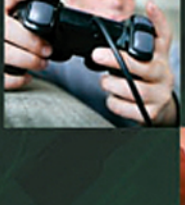
MULTIMEDIA OVER IP *and* WIRELESS NETWORKS

COMPRESSION, NETWORKING, AND SYSTEMS

MIHAELA VAN DER SCHAAR • PHILIP A. CHOU



MULTIMEDIA OVER IP AND WIRELESS NETWORKS

This page intentionally left blank

MULTIMEDIA OVER IP AND WIRELESS NETWORKS

COMPRESSION, NETWORKING, AND SYSTEMS

Edited by

Philip A. Chou
Microsoft Corporation

Mihaela van der Schaar
University of California, Los Angeles

ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Academic Press is an imprint of Elsevier

ACADEMIC PRESS

Academic Press is an imprint of Elsevier
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA
525 B Street, Suite 1900, San Diego, California 92101-4495, USA
84 Theobald's Road, London WC1X 8RR, UK

This book is printed on acid-free paper. ♾

Library of Congress Cataloging-in-Publication Data
Application submitted

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library.

Multimedia over IP and wireless networks : compression, networking, and systems / edited by Philip A. Chou, Mihaela van der Schaar.
p. cm.
ISBN-10: 0-12-088480-1
ISBN-13: 978-0-12-370856-4
1. Multimedia communications. 2. Computer networks. 3. Multimedia systems. I. Chou, Philip A. II. Schaar, Mihaela van der.
TK5105.15.M95 2007
006.7–dc22
2007003425

ISBN 13: 978-0-12-088480-3
ISBN 10: 0-12-088480-1

For information on all Academic Press publications
visit our Web site at www.books.elsevier.com

Printed in the United States of America
07 08 09 10 11 10 9 8 7 6 5 4 3 2 1

Table of Contents

ABOUT THE EDITORS

Mihaela van der Schaar received her Ph.D. degree from Eindhoven University of Technology, Eindhoven, The Netherlands, in 2001. She is currently an Assistant Professor in the Electrical Engineering Department at UCLA. Prior to this, between 1996 and June 2003 she was a senior researcher at Philips Research in the Netherlands and the USA, where she led a team of researchers working on multimedia coding, processing, networking, and streaming algorithms and architectures. She has published extensively on multimedia compression, processing, communications, networking, and architectures and holds 28 granted U.S. patents and several more pending. Since 1999, she has been an active participant to the ISO Motion Picture Expert Group (MPEG) standard, to which she made more than 50 contributions and for which she received three ISO recognition awards. She chaired the ad hoc group on MPEG-21 Scalable Video Coding for three years, and also co-chaired the MPEG ad hoc group on Multimedia Test-bed. She is a senior member of IEEE, and was also elected as a Member of the Technical Committee on Multimedia Signal Processing (MMSP TC) and Image and Multidimensional Signal Processing Technical Committee (IMDSP TC) of the IEEE Signal Processing Society. She was an Associate Editor of IEEE Transactions on Multimedia and SPIE Electronic Imaging Journal from 2002 to 2005. Currently, she is an Associate Editor of IEEE Transactions on Circuits and Systems for Video Technology and an Associate Editor of IEEE Signal Processing Letters. She served as a General Chair for the Picture Coding Symposium (PCS) in 2004. She received the NSF CAREER Award in 2004, the IBM Faculty Award in 2005, the Okawa Foundation Award in 2006, and the Best Paper Award for her paper published in 2005 in the IEEE Transactions on Circuits and Systems for Video Technology.

Philip A. Chou received a B.S.E. degree from Princeton University, Princeton, NJ, in 1980, and an M.S. degree from the University of California, Berkeley, in

1983, both in electrical engineering and computer science, and a Ph.D. degree in electrical engineering from Stanford University in 1988. From 1988 to 1990, he was a Member of Technical Staff at AT&T Bell Laboratories in Murray Hill, NJ. From 1990 to 1996, he was a Member of Research Staff at the Xerox Palo Alto Research Center in Palo Alto, CA. In 1997, he was the manager of the compression group at VXtreme in Mountain View, CA, before it was acquired by Microsoft in 1997. From 1998 to the present, he has been a Principal Researcher with Microsoft Research in Redmond, Washington, where he currently manages the Communication and Collaboration Systems research group. Dr. Chou also served as a Consulting Associate Professor at Stanford University from 1994 to 1995, an Affiliate Associate Professor at the University of Washington since 1998, and an Adjunct Professor at the Chinese University of Hong Kong since 2006. Dr. Chou's research interests are data compression, information theory, communications, and pattern recognition, with applications to video, images, audio, speech, and documents. Dr. Chou served as an Associate Editor in source coding for the IEEE Transactions on Information Theory from 1998 to 2001 and as a Guest Associate Editor for special issues in the IEEE Transactions on Image Processing and the IEEE Transactions on Multimedia in 1996 and 2004, respectively. From 1998 to 2004, he was a Member of the IEEE Signal Processing Society's Image and Multidimensional Signal Processing Technical Committee (IMDSP TC). He served as Program Committee Chair for the inaugural NetCod 2005 workshop, and he currently serves on the organizing committee for ICASSP 2007. He is a Fellow of the IEEE, a member of Phi Beta Kappa, Tau Beta Pi, Sigma Xi, and the IEEE Computer, Information Theory, Signal Processing, and Communications Societies, and was an active member of the MPEG committee. He is the recipient, with Anshul Seghal, of the 2002 ICME Best Paper award, and he is the recipient, with Tom Lookabaugh, of the 1993 Signal Processing Society Paper award.

ABOUT THE AUTHORS

John Apostolopoulos received his B.S., M.S., and Ph.D. degrees in EECS from Massachusetts Institute of Technology (MIT). He joined Hewlett-Packard Laboratories in 1997, where he is a Principal Research Scientist and Project Manager for the Streaming Media Systems Group. He also teaches and conducts joint research at Stanford University, where he is a Consulting Assistant Professor in EE. He received a Best Student Paper Award for part of his Ph.D. thesis, the Young Investigator Award (Best Paper Award) at VCIP 2001 for his paper on multiple description video coding and path diversity for reliable video communication over lossy packet networks, and in 2003 was named "one of the world's top 100 young (under 35) innovators in science and technology" (TR100) by Technology Review. He contributed to both the U.S. Digital Television and JPEG-2000 Security (JPSEC) standards. His research interests include improving the reliability, fidelity, scalability, and security of media communication over wired and wireless packet networks.

Kai Chen received his Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign in 2004. He received his M.S. and B.S. degrees from University of Delaware and Tsinghua University, respectively. He is currently working at Google Inc.

Jim Chou received B.S. and M.S. degrees in electrical engineering from the University of Illinois at Urbana-Champaign in 1995 and 1997, respectively. He received the Ph.D. degree in electrical engineering from the University of California, Berkeley in 2002. He has worked at TRW, Bytemobile, and Sony Research in the past. Jim holds two U.S. patents and has several patents pending. Currently, Jim is a Video Architect at C2 Microsystems. His research interests include coding theory, wireless video transmission, digital watermarking, and estimation and detection theory.

Philip A. Chou is Principal Researcher and Manager of the Communication and Collaboration Systems group at Microsoft Research. He also holds affiliate pro-

fessor positions at the University of Washington and the Chinese University of Hong Kong. Prior to coming to Microsoft, Dr. Chou was Compression Group Manager at VXtreme (a startup company acquired by Microsoft) in 1997, a Member of Research Staff at the Xerox Palo Alto Research Center from 1990 to 1996, a Consulting Associate Professor at Stanford from 1994 to 1995, and a Member of the Technical Staff at AT&T Bell Laboratories from 1988 to 1990. Dr. Chou received a Ph.D. from Stanford University in 1988, an M.S. from the University of California, Berkeley, in 1983, and a B.S.E. from Princeton University in 1980. His research interests include data compression, information theory, communications, and pattern recognition, with applications to video, images, audio, speech, and documents. Dr. Chou is a Fellow of IEEE.

Dinei Florêncio received B.S. and M.S. degrees from the University of Brasília, Brazil, and a Ph.D. degree from the Georgia Institute of Technology, Atlanta, all in electrical engineering. He has been a researcher in the communication and collaboration systems group at Microsoft Research since 1999. From 1996 to 1999, he was a Member of Research Staff at the David Sarnoff Research Center. From 1994 to 1996, he was an Associated Researcher with AT&T Human Interface Lab (now part of NCR), and an intern at the (now defunct) Interval Research in 1994. He is a Senior Member of the IEEE. He has published over 25 referred papers, has been granted 20 U.S. patents, and has received the 1998 Sarnoff Achievement Award.

Bernd Girod is Professor of Electrical Engineering and (by courtesy) Computer Science in the Information Systems Laboratory of Stanford University, California. He was Chaired Professor of Telecommunications in the Electrical Engineering Department of the University of Erlangen-Nuremberg from 1993 to 1999. His research interests are in the areas of video compression and networked media systems. Prior visiting or regular faculty positions include the Massachusetts Institute of Technology, Georgia Institute of Technology, and Stanford University. He has been involved with several startup ventures as founder, director, investor, or advisor, among them Vivo Software, 8x8 (Nasdaq: EGHT), and RealNetworks (Nasdaq: RNWK). Since 2004, he has served as the Chairman of the new Deutsche Telekom Laboratories in Berlin. He received an Engineering Doctorate from University of Hannover, Germany, and an M.S. degree from Georgia Institute of Technology. Professor Girod is a Fellow of IEEE.

Raouf Hamzaoui received an M.Sc. degree in mathematics from the University of Montreal, Canada, in 1993, the Dr. rer. nat. degree from the Faculty of Applied Sciences of the University of Freiburg, Germany, in 1997, and the Habilitation degree in computer science from the University of Konstanz, Germany, in 2004.

From 2000 to 2002, he was an Assistant Professor with the Department of Computer Science of the University of Leipzig, Germany. From 2002 to August 2006, he was an Assistant Professor with the Department of Computer and Information Science of the University of Konstanz. Since September 2006, he has been a Professor for Media Technology in the School of Engineering and Technology at De Montfort University, Leicester, United Kingdom. His research interests include image and video compression, multimedia communication, channel coding, and algorithms.

Mark Kalman received a B.S. in Electrical Engineering and a B.Mus. in Composition from Johns Hopkins University in 1997. He completed his M.S. and Ph.D. degrees, both in Electrical Engineering, at Stanford University in 2001 and 2006, respectively. He is currently with Pure Digital Technologies, Inc., in San Francisco, California.

Syed Ali Khayam received his B.E. degree in Computer Systems Engineering from National University of Sciences and Technology (NUST), Pakistan, in 1999 and his M.S. degree in Electrical Engineering from Michigan State University (MSU) in 2003. He received his Ph.D. from MSU in December 2006. He worked at Communications Enabling Technologies from October 2000 to August 2001. His research interests include analysis and modeling of statistical phenomena in computer networks, network security with emphasis on detection and mitigation of self-propagating malware, cross-layer design for wireless networks, and real-time multimedia communications.

Yen-Chi Lee received a B.S. and M.S. degrees in Computer Science and Information Engineering from National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 1997 and 1999, respectively, and a Ph.D. degree in Electrical and Computer Engineering from Georgia Institute of Technology, Atlanta, in 2003. In 2003, he joined Nokia Research Center, Irving, TX, as a research engineer, where he conducted research on video teleconferencing over GSM GPRS/EGPRS networks. He has been with Qualcomm Inc., San Diego, CA, as a video system engineer since 2004. His current research focuses on the areas of video compression techniques and real-time wireless video communications; particularly, error-resilient video coding and error control, low-delay video rate control, and channel rate adaptation. Yen-Chi has published 16 research papers and currently holds 14 pending patent applications.

Jin Li is currently a Senior Researcher at Microsoft Research (MSR) Redmond. He received his Ph.D. from Tsinghua University in 1994. Before moving to Redmond, he worked at the University of Southern California, the Sharp Laboratories

of America, and MSR Asia. Since 2000, Dr. Li has also served as an adjunct professor at Tsinghua University. Dr. Li has more than 80 referred conference and journal papers in a diversified research field of media compression and communication and peer-to-peer content delivery. He holds 18 issued U.S. patents, with many more pending. Dr. Li is an Area Editor for the Journal of Visual Communication and Image Representation (Academic Press) and an Associate Editor of IEEE Transactions on Multimedia. He is a Senior Member of IEEE. He was the recipient of the 1994 Ph.D. Thesis Award from Tsinghua University and the 1998 Young Investigator Award from SPIE Visual Communication and Image Processing.

Shipeng Li received B.S. and M.S. degrees from the University of Science and Technology of China (USTC), Hefei, China, in 1988 and 1991, respectively, and a Ph.D. degree from Lehigh University, Bethlehem, PA, in 1996, all in electrical engineering. He was with the Electrical Engineering Department, USTC, from 1991 to 1992. He was a Member of Technical Staff with Sarnoff Corporation, Princeton, NJ, from 1996 to 1999. He has been a Researcher with Microsoft Research Asia, Beijing, China, since May 1999 and has contributed to some technologies in MPEG-4 and H.264. His research interests include image/video compression and communications, digital television, multimedia, and wireless communication.

Yi Liang's expertise is in the areas of networked multimedia systems, real-time voice and video communication, and low-latency media streaming over wire-line and wireless networks. Currently holding positions at Qualcomm CDMA Technologies, San Diego, CA, he is responsible for the design and development of video and display system architecture for multimedia handset chipsets. From 2000 to 2001, he conducted research with Netergy Networks, Inc., Santa Clara, CA, on voice-over-IP systems that provide superior quality over best-effort networks. From 2001 to 2003, he led the Stanford-Hewlett-Packard Labs low-latency video streaming project, in which he and his colleagues developed error-resilience techniques for rich-media-communication-over-IP networks at very low latency. In the summer of 2002 at Hewlett-Packard Labs, Palo Alto, CA, he developed an accurate loss-distortion model for compressed video and contributed in the development of the pioneering mobile streaming media content delivery network (MSM-CDN) that delivers rich media over 3G wireless. Yi Liang received a Ph.D. degree in Electrical Engineering from Stanford University in 2003 and a B.Eng. degree from Tsinghua University, Beijing, China, in 1997.

Dmitri Loguinov received a B.S. degree (with honors) in computer science from Moscow State University, Russia, in 1995 and a Ph.D. degree in computer science from the City University of New York in 2002. Since 2002, he has been

an Assistant Professor of Computer Science with Texas A&M University, College Station. His research interests include peer-to-peer networks, Internet video streaming, congestion control, topology modeling, and Internet traffic measurement.

Abhik Majumdar received a B.Tech. degree from the Indian Institute of Technology (IIT), Kharagpur, and M.S. and Ph.D. degrees from the University of California, Berkeley, in 2000, 2003, and 2005, respectively, all in Electrical Engineering. He is currently with Pure Digital Technologies, San Francisco, CA. His research interests include multimedia compression and networking and wireless communications. Dr. Majumdar was awarded the Institute Silver Medal from I.I.T. Kharagpur for outstanding achievement in the graduating class of 2000.

Klara Nahrstedt is a Full Professor at the University of Illinois at Urbana-Champaign, Computer Science Department. Her research interests are directed toward multimedia distributed systems, quality of service (QoS) management in wired and mobile ad hoc networks, QoS-aware resource management in distributed multimedia systems, QoS-aware middleware systems, quality of protection in multimedia systems, and tele-immersive applications. She is the co-author of the widely used multimedia book *Multimedia: Computing, Communications and Applications*, published by Prentice Hall in 1995, and the multimedia book *Multimedia Systems*, published by Springer-Verlag in 2004. She is the recipient of the Early NSF Career Award, the Junior Xerox Award, and the IEEE Communication Society Leonard Abraham Award for Research Achievements. She is the Editor-in-Chief of the ACM/Springer Multimedia Systems Journal, and the Ralph and Catherine Fisher Professor. Klara Nahrstedt received her B.A. in mathematics from Humboldt University, Berlin, in 1984, and an M.Sc. degree in numerical analysis from the same university in 1985. She was a research scientist in the Institute for Informatik in Berlin until 1990. In 1995, she received her Ph.D. from the University of Pennsylvania in the Department of Computer and Information Science. She is a Member of ACM and a Senior Member of IEEE.

Antonio Ortega received the Telecommunications Engineering degree from the Universidad Politecnica de Madrid, Spain, in 1989 and the Ph.D. in Electrical Engineering from Columbia University, New York, NY in 1994. His Ph.D. work was supported by a Fulbright Scholarship. In 1994, he joined the Electrical Engineering-Systems department at the University of Southern California, where he is currently a Professor and Associate Chair of the Department. He is a Senior Member of IEEE, and a Member of ACM. He has been Chair of the Image and Multidimensional Signal Processing Technical Committee (IMDSP TC) and

a member of the Board of Governors of the IEEE SPS (2002). He was the Technical Program Co-chair of ICME 2002 and has served as Associate Editor for the IEEE Transactions on Image Processing and the IEEE Signal Processing Letters. He received the National Science Foundation (NSF) CAREER award, the 1997 IEEE Communications Society Leonard G. Abraham Prize Paper Award, and the IEEE Signal Processing Society 1999 Magazine Award. His research interests are in the areas of multimedia compression and communications. His recent work focuses on distributed compression, multiview coding, compression for recognition and classification applications, error-tolerant compression, and information representation for wireless sensor networks.

Béatrice Pesquet-Popescu is an Associate Professor at ENST Paris, where she is currently the leader of the Multimedia Group. Her current research interests are in scalable and robust video coding, adaptive wavelets, and multimedia applications. EURASIP gave her a Best Student Paper Award in the IEEE Signal Processing Workshop on Higher-Order Statistics in 1997; in 1998, she received a Young Investigator Award granted by the French Physical Society, and she received, together with D. Turaga and M. van der Schaar, the 2006 IEEE Circuits and Systems Society CSVT Transactions Best Paper Award for the paper "Complexity Scalable Motion Compensated Wavelet Video Encoding." She has authored more than one hundred book chapters, journal articles, and conference papers in the field and holds more than 20 patents in wavelet-based video coding. She is a Member of the IEEE Multimedia Signal Processing Technical Committee, an elected EURASIP AdCom Member, and a Senior Member of IEEE.

Rohit Puri received a B.Tech. degree from the Indian Institute of Technology, Bombay, the M.S. degree from the University of Illinois at Urbana-Champaign, and a Ph.D. degree from the University of California, Berkeley, in 1997, 1999, and 2002, respectively, all in electrical engineering. From 2003 to 2004, he was with Sony Electronics Inc., San Jose, CA. He was then with the EECS Department, University of California, Berkeley, as a Research Engineer. He is currently a Senior Video Architect at PortalPlayer Inc., San Jose, CA. His research interests include multimedia compression, distributed source coding, multiple descriptions coding, and multi-user information theory. Dr. Puri was awarded the Institute Silver Medal by the Indian Institute of Technology, Bombay, for outstanding achievement in the graduating class, in 1997. He was a recipient of the 2004 Eliahu I. Jury Award at the University of California, Berkeley, for the best doctoral thesis in the area of systems, signal processing, communications, and controls.

Hayder Radha is a Professor of Electrical and Computer Engineering at Michigan State University (MSU). He received his Ph.M. and Ph.D. degrees from Columbia University in 1991 and 1993, an M.S. degree from Purdue University

in 1986, and a B.S. degree (honors) from MSU in 1984 (all in electrical engineering). From 1996 to 2000, he worked for Philips Research as a Principal Member of Research Staff and Consulting Scientist in the Video Communications Research Department. From 1986 to 1996, he worked at Bell Labs in the areas of digital communications, image processing, and broadband multimedia networking. He served as Co-chair and Editor of the Broadband and LAN Video Coding Experts Group of the ITU-T. He was a Philips Research Fellow, and he is a recipient of the Bell Labs Distinguished Member of Technical Staff, AT&T Circle of Excellence, College of Engineering Withrow Distinguished Scholar, and the Microsoft Research Content and Curriculum Awards.

Kannan Ramchandran received M.S. and Ph.D. degrees from Columbia University in Electrical Engineering in 1984 and 1993, respectively. From 1984 to 1990, he was a Member of Technical Staff at AT&T Bell Labs in the telecommunications R&D area. From 1993 to 1999, he was on the faculty of the Electrical and Computer Engineering Department at the University of Illinois at Urbana-Champaign and was a Research Assistant Professor at the Beckman Institute and the Coordinated Science Laboratory. Since fall 1999, he has been an Associate Professor in the Electrical Engineering and Computer Sciences department, University of California, Berkeley. His current research interests include distributed algorithms for signal processing and communications, multi-user information theory, wavelet theory and multiresolution signal processing, and unified algorithms for multimedia signal processing, communications, and networking. Dr. Ramchandran was a recipient of the 1993 Eliahu I. Jury Award at Columbia University for the best doctoral thesis in the area of systems, signal processing, and communications. His research awards include the National Science Foundation (NSF) CAREER award in 1997, ONR and ARO Young Investigator Awards in 1996 and 1997, and the Okawa Foundation Award at the University of California, Berkeley, in 2000. In 1998, he was selected as a Henry Magnusky Scholar by the Electrical and Computer Engineering department at the University of Illinois, an honor that recognizes excellence among junior faculty. He is the co-recipient of two Best Paper Awards from the IEEE Signal Processing Society, has been a Member of the IEEE Image and Multidimensional Signal Processing Committee and the IEEE Multimedia Signal Processing Committee, and has served as an Associate Editor for the IEEE Transactions on Image Processing.

Samarth Shah received his B.E. degree in Computer Science and Engineering from the University of Madras, India, in 1998 and completed his Ph.D. in Computer Science at the University of Illinois at Urbana-Champaign in 2005. Since 2005, he has been working at Motorola Inc. in Libertyville, Illinois, in the area of VoIP-over-WiFi.

Vladimir Stanković received the Dipl.-Ing. degree in electrical engineering from the University of Belgrade, Serbia, in 2000, and the Dr.-Ing. degree from the University of Leipzig, Germany, in 2003. From 2002 to 2003, he was with the Department of Computer and Information Science, University of Konstanz, Germany. From June 2003 to February 2006, he was with the Department of Electrical and Computer Engineering at Texas A&M University, College Station, first as a Postdoctoral Research Associate and then as a Research Assistant Professor. In February 2006, Dr. Stanković joined the Department of Communication Systems, Lancaster University, Lancaster, United Kingdom, as a lecturer. His research focuses on multimedia networking, network information theory, and wireless communications.

Eckehard Steinbach (IEEE M'96) studied Electrical Engineering at the University of Karlsruhe, Germany, the University of Essex, United Kingdom, and ESIEE in Paris. From 1994 to 2000, he was a member of the research staff of the Image Communication Group at the University of Erlangen-Nuremberg (Germany), where he received an Engineering Doctorate in 1999. From February 2000 to December 2001, he was a Postdoctoral Fellow with the Information Systems Laboratory of Stanford University. In February 2002, he joined the Department of Electrical Engineering and Information Technology of Munich University of Technology, Germany, where he is currently an Associate Professor for Media Technology. His current research interests are in the area of networked and interactive multimedia systems.

Thomas Stockhammer has been working at the Munich University of Technology, Germany and was visiting researcher at Rensselear Polytechnic Institute (RPI), Troy, NY and at the University of San Diego, California (UCSD). He has published more than 80 conference and journal papers, is member of different program committees, and holds several patents. He regularly participates and contributes to different standardization activities, such as JVT, IETF, 3GPP, ITU, and DVB, and has co-authored more than 100 technical contributions. He is acting chairman of the video ad hoc group of 3GPP SA4. He is also co-founder and CEO of Novel Mobile Radio (NoMoR) Research, a company working on the simulation and emulation of future mobile networks. He is working as a research and standardization consultant for Siemens Mobile Devices and now consults for Digital Fountain, the leading provider for forward error correction. His research interests include video transmission, cross-layer and system design, forward error correction, content delivery protocols, rate–distortion optimization, information theory, and mobile communications.

Wai-Tian Tan joined Hewlett-Packard Laboratories in December 2000, where he is a member of the Streaming Media Systems Group. He received a B.S. degree

from Brown University in 1992, an M.S.E.E. degree from Stanford University in 1993 and a Ph.D. degree from the University of California, Berkeley, in 2000. He worked for Oracle Corporation from 1993 to 1995. His research focuses on adaptive media streaming, both at the end-point and inside the delivery infrastructure.

Chia-Yuan (Andy) Teng was born in Taipei, Taiwan, China, in 1964. He received a college diploma from National Taipei Institute of Technology, Taipei, Taiwan, in 1984, a M.S. degree in Electrical Engineering from National Sun Yat-Sen University, Kaoshiung, Taiwan, in 1989, and a Ph.D. degree in Electrical Engineering and Computer Science from the University of Michigan, Ann Arbor, in 1996. In 1989, he was with the Industrial Technology Research Institute (ITRI), Hsinchu, Taiwan. From 1990 to 1992, he was a Faculty Member of the Department of EE, Chien-Hsin Institute of Technology, Chunli, Taiwan. From 1996 to 1998, he was with the Corporate Research, Thomson Multimedia, where he participated in the standardization and research of digital TV, satellite, and cable systems. From 1998 to 2004, he was with the San Diego R&D Center, Nokia Mobile Phones, where he was a Technical Team Leader in DSP entity and involved in the development and design for multimedia, streaming, and DSP firmware. Dr. Teng joined Qualcomm Corporation in Aug. 2004, where he is currently a Staff Engineer/Manager in the Video Systems Group. His research interests include video/image coding, video/image processing, multimedia streaming, Internet protocols, and video telephony.

Mitchell Trott received B.S. and M.S. degrees in Systems Engineering from Case Western Reserve University in 1987 and 1988, respectively, and a Ph.D. in electrical engineering from Stanford University in 1992. He was an Associate Professor in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology (MIT) from 1992 until 1998, and director of research at ArrayComm from 1997 through 2002. He is now a member of the Streaming Media Systems Group at Hewlett-Packard Laboratories. His research interests include streaming media systems, multi-user and wireless communication, and information theory.

Mihaela van der Schaar is currently an Assistant Professor in the Electrical Engineering Department at UCLA. She has published extensively on multimedia compression, processing, communications, networking, and architectures and holds 28 granted U.S. patents. Since 1999, she has been an active participant to the ISO Motion Picture Expert Group (MPEG) standard, to which she made more than 50 contributions and for which she received three ISO recognition awards. She was an Associate Editor of IEEE Transactions on Multimedia and SPIE Electronic Imaging Journal and is currently an Associate Editor of IEEE Transactions

on Circuits and System for Video Technology and of IEEE Signal Processing Letters. She received the NSF CAREER Award in 2004, the IBM Faculty Award in 2005, the Okawa Foundation Award in 2006, and the Best Paper Award for her paper published in 2005 in the IEEE Transactions on Circuits and Systems for Video Technology.

Huisheng Wang received the B.Eng. degree from Xi'an Jiaotong University, China, in 1995 and the M.Eng. degree from Nanyang Technological University, Singapore, in 1998, both in electrical engineering. She is currently pursuing her Ph.D. degree in the Department of Electrical Engineering-Systems at the University of Southern California, Los Angeles. From 1997 to 2000, she worked in Creative Technology Ltd., Singapore as a R&D software engineer. She was also a research intern at La Jolla Lab, ST Microelectronics, San Diego, CA, and at HP Labs, Palo Alto, CA. Her research interests include signal processing, multimedia compression, networking, and communications.

Susie Wee is the Director of the Mobile and Media Systems Lab in Hewlett-Packard Laboratories (HP Labs). She is responsible for research programs in multimedia, networked sensing, next-generation mobile multimedia systems, and experience design. Her lab has activities in the U.S., Japan, and the United Kingdom, and includes collaborations with partners around the world. Wee's research interests broadly embrace the design of mobile streaming media systems, secure scalable streaming methods, and efficient video delivery algorithms. In addition to her work at HP Labs, Wee is a Consulting Assistant Professor at Stanford University. She received Technology Review's Top 100 Young Investigators Award in 2002, served as an Associate Editor for the IEEE Transactions on Image Processing and IEEE Transactions on Circuits, Systems, and Video Technologies. She is currently a Co-Editor of the JPEG-2000 Security standard (JPSEC). Wee received her B.S., M.S., and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology (MIT).

Zixiang Xiong received a Ph.D. degree in Electrical Engineering in 1996 from the University of Illinois at Urbana-Champaign. He is currently an Associate Professor in the Department of Electrical and Computer Engineering at Texas A&M University, College Station. His research interests are network information theory, code designs and applications, networked multimedia, and genomic signal processing.

Yuan Xue received her B.S. in Computer Science from Harbin Institute of Technology, China in 1998 and her M.S. and Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 2002 and 2005, respectively. Currently, she is an Assistant Professor at the Department of Electrical Engineering

and Computer Science of Vanderbilt University. She is a recipient of the Vodafone fellowship. Her research interests include wireless and sensor networks, peer-to-peer and overlay systems, QoS support, and network security. She is a Member of the IEEE and ACM.

Wanghong Yuan received his B.S. and M.S. degrees in 1996 and 1999, respectively, from the Department of Computer Science, Beijing University, and his Ph.D. degree in 2004 from the Department of Computer Science, University of Illinois at Urbana-Champaign. He is a software engineer at Microsoft Corporation. Before joining Microsoft, he was a research engineer at DoCoMo USA Labs from 2004 to 2006. His research and development interests include operating systems, networks, multimedia, and real-time systems, with an emphasis on the design of energy-efficient and QoS-aware operating systems.

Waqar Zia received his B.Sc. degree in electrical engineering from the University of Engineering and Technology, Taxila, Pakistan in 2000. He worked on embedded digital video processing for three years in Streaming Networks Ltd., Islamabad, Pakistan. He received his M.Sc. degree in Information and Communication Systems from Hamburg University of Technology, Germany, in 2005. He then started working on his Ph.D. under the supervision of Prof. Klaus Diepold and Thomas Stockhammer at the Technical University of Munich, Germany. His work focuses on complexity-constrained error-robust video communication on handheld devices. He has also actively participated in recent 3GPP standardization and has co-authored several technical contributions along with pursuing his research work.

This page intentionally left blank

PART A

OVERVIEW

This page intentionally left blank

1

Multimedia Networking and Communication: Principles and Challenges

Mihaela van der Schaar and Philip A. Chou

In case you haven't noticed, multimedia communication over IP and wireless networks is exploding. Applications such as BitTorrent, used primarily for video downloads, now take up the lion's share of all traffic on the Internet. Music file sharing, once on the legal cutting edge of massive copyright infringement on college campuses around the world, has moved into the mainstream with significant legal downloads of music and video to devices such as the iPod and numerous other portable media players. Multimedia podcasting to client computers and portable devices is a phenomenon exploding in its own right. Internet radio, pioneered in the late 1990s, is now being joined in a big way by peer-to-peer television such as CoolStreaming and PPLive. Audio and video on demand over the Internet, also available since the late 1990s on the Web sites of well-funded organizations such as CNN.com and MSNBC.com, are now at the core of a multitude of new music and video businesses from Napster to iTunes to MTV's Urge service, and will be expanding imminently to full-length movie delivery on demand. Moreover, Web sites such as YouTube have made publishing videos on demand available to anyone with a home video camera, which these days is nearly everyone with a mobile phone. Indeed, most mobile phones today can actively download *and upload* both photos and videos, sometimes in real time. Internet telephony is exploding, with popular applications such as Skype and others enabling wideband voice and video conferencing over the Internet. In general, voice over IP (VoIP) is revolutionizing the telecommunications industry, as circuit-switched equipment from PBX to long haul equipment is being replaced by soft IP switches. Enhanced television is also being delivered into the living room over IP networks by traditional telephone providers through DSL.

Once inside the home, consumer electronics manufacturers, and increasingly, the computer industry and its partners, are distributing audio and video over WiFi to monitors and speaker systems around the house. Now that the analog-to-digital revolution is nearly complete, we are undergoing an all-media-over-IP revolution, with radio, television, telephony, and stored media all currently being delivered over IP wireline and wireless networks. To top it all off, brand new types of media, such as game data for interactive gaming over the Internet, are strongly emerging.

Despite having unleashed a plethora of new multimedia applications, the Internet and wireless networks provide only limited support for multimedia. The Internet and wireless networks have inherently unpredictable and variable conditions. If averaged over time, this variability may not significantly impact delay-insensitive applications such as file transfer. However, variations in network conditions can have considerable consequences for real-time multimedia applications and can lead to unsatisfactory user experience. Multimedia applications tend to be *delay sensitive, bandwidth intense, and loss tolerant*. These properties can change the fundamental principles of communication design for these applications.

The concepts, theories, and solutions that have traditionally been taught in information theory, communication, and signal processing courses may not be directly applicable to highly time-varying channel conditions, adaptive and delay-sensitive multimedia applications, and interactive multiuser transmission environments. Consequently, in recent years, the area of multimedia communication and networking has emerged not only as a very active and challenging integrative research topic across the borders of signal processing and communication, but also as a core curriculum that requires its own set of fundamental concepts and algorithms that differ from those taught in conventional signal processing and communication courses.

This book aims at providing the reader with an in-depth understanding of the theoretical foundations, key design principles, algorithms, and existing standards for multimedia communication and networking.

This introductory chapter provides motivation for studying the topic of multimedia communication, the addressed applications, and associated challenges. Subsequently, a road map of the various chapters is provided. A suggested use for graduate instruction and self-study is also provided.

1.1 DIMENSIONS OF MULTIMEDIA COMMUNICATION

1.1.1 Multimedia Communication Applications

The emergence of communication infrastructures such as the Internet and wireless networks enabled the proliferation of the aforementioned multimedia applications. These applications range from simple music downloading to a portable

device, to watching TV through the Internet on a laptop, or to viewing movie trailers posted on the Web via a wireless link. Some of these applications are new to the Internet revolution, while others may seem more traditional, such as sending VoIP to an apparently conventional telephone, sending television over IP to an apparently conventional set top box, or sending music over WiFi to an apparently conventional stereo amplifier.

An obvious question that comes to mind when considering all the aforementioned applications is how to jointly discuss these applications. What do they have in common and how do they differ? To provide an answer to this seemingly simple question, we will discuss the various dimensions of these multimedia communication applications.

1.1.2 Streaming Versus Downloading

Conventional downloading applications (e.g., file transfer such as FTP) involve downloading a file before it is viewed or consumed by a user. Examples of such multimedia downloading applications are downloading an MP3 song to a portable device, downloading a video file to a computer via BitTorrent, or downloading a podcast. (Despite its name, podcasting is a "pull" technology with which a Web site is periodically polled for new multimedia content.) Downloading is usually a very robust way to deliver media to a user. However, downloading has two potentially important disadvantages for multimedia applications. First, a large buffer is required whenever a large media file (e.g., an MPEG-4 movie) is downloaded. Second, the amount of time required for the download can be relatively large, thereby requiring the user to wait minutes or even hours before being able to consume the content. Thus, while downloading is simple and robust, it provides only limited flexibility both to users and to application designers.

An alternative to downloading is streaming. Streaming applications split the media bit stream into separate chunks (e.g., packets), which can be transmitted independently. This enables the receiver to decode and play back the parts of the bit stream that are already received. The transmitter continues to send multimedia data chunks while the receiver decodes and simultaneously plays back other, already received parts of the bit stream. This enables low delay between the moment data is sent by the transmitter to the moment it is viewed by the user. Low delay is of paramount importance for interactive applications such as video conferencing, but it is also important both for video on demand, where the user may desire to change channels or programs quickly, and for live broadcast, where the content length is unbounded a priori, but the delay must be finite. Another advantage of streaming is its relatively low storage requirements and increased flexibility for the user, compared to downloading. However, streaming applications, unlike downloading applications, have deadlines and other timing requirements to ensure

continuous real-time media playout. This leads to new challenges for designing communication systems to best support multimedia streaming applications.

1.1.3 Streaming Media on Demand, Live Broadcast, and Real-Time Communication

Multimedia streaming applications can be partitioned into three classes by delay tolerance. Interactive audio and video telephony, teleconferencing, and gaming have extremely low delay tolerance, usually no more than 200 ms of end-to-end delay for comfortable interaction. In contrast, live broadcast applications (e.g., Internet radio), which typically have no interactivity, have a large delay tolerance, say up to 30 s, because the delay cannot be detected without interactivity and without a reference, such as a neighbor who is listening to a conventional radio. (Cheers coming from a neighbor's apartment 30 s before a goal can certainly ruin the surprise!) Intermediate in terms of delay tolerance is the application of streaming media on demand, which has only moderate interactivity requirements, such as channel changing and VCR-like control. The differences in delay tolerance among these three classes of multimedia applications have profound effects on their design, particularly with respect to error recovery. Low-delay, low bit rate applications such as telephony can afford only error-resilience techniques, whereas high-delay or high bandwidth applications can afford complete error recovery using either forward error correction or retransmission-based techniques.

It is worth noting here that although applications in all three classes play out multimedia in real time, the phrase "real-time communication" is commonly used only for the first application, that is, audio and video telephony, conferencing, and gaming, whereas the phrase "streaming" is often associated only with the latter two applications.

1.1.4 Online Versus Off-Line Encoding

Another essential difference between multimedia communication applications is whether the content is encoded online, as in the case of real-time communication or live broadcast applications, or is encoded off-line, as in the case of streaming media on demand. The advantage of online encoding is that the communication channel can be monitored and the source and channel coding strategies can be adapted correspondingly. For instance, the receiver can inform the transmitter of the information that is lost and the encoder can adjust correspondingly. The advantage of off-line encoding is that the content can be exhaustively analyzed and the encoding can be optimized (possibly in nonreal time over several passes of the data) for efficient transmission.

1.1.5 Receiver Device Characteristics

The constraints of the receiver devices on which the various applications are consumed by the end user also have an important impact on multimedia communication. In particular, the available storage, power, and computational capabilities of the receiving device need to be explicitly considered when designing complete multimedia communication solutions. For instance, the design of multimedia compression, scheduling, and error protection algorithms at the receiver should explicitly consider the ability of the receiver to cope with packet loss. Also, receiver-driven streaming applications can enable the end device to proactively decide which parts of the compressed bit streams should be transmitted depending on the display size and other factors.

1.1.6 Unicast, Multicast, and Broadcast

Multimedia communication can be classified into one of three different categories: unicast, multicast, and broadcast, depending on the relationship between the number of senders and receivers. Unicast transmission connects one sender to one receiver. Examples of such applications include downloading, streaming media on demand, and point-to-point telephony. A main advantage of unicast is that a back channel can be established between the receiver and the sender. When such a back channel exists, the receiver can provide feedback to the sender about experienced channel conditions, end-user requirements, end-device characteristics, and so on, which can be used accordingly to adapt compression, error protection, and other transmission strategies.

Multicast transmission connects the sender to multiple receivers that have elected to participate in the multicast session, over IP multicast or application level multicast. Multicast is more efficient than multiple unicasts in terms of network resource utilization and server complexity. However, a disadvantage of multicast compared to unicast is that the sender cannot target its transmission toward a specific receiver.

Broadcast transmission connects a sender to all receivers that it can reach through the network. An example is broadcast over a wireless link or a shared Ethernet link. As in multicast, the communication channel may be different for different receivers. In this book, when we refer to the live broadcast application, we are usually talking about a solution in which a live signal is actually multicast over the network.

1.1.7 Metrics for Quantifying Performance

Unlike conventional communication applications, multimedia communication applications cannot be simply evaluated in terms of the achieved throughput, packet

loss rates, or bit error rates, as these applications are delay sensitive and not all the various transmitted bits are "created equal," that is, have the same importance. Instead, multimedia performance needs to be quantified in terms of metrics such as the perceived quality or objective metrics such as the Peak Signal-to-Noise Ratio (PSNR) between transmitted and received media data. Hence, the importance of each bit or packet of multimedia data depends on its delay requirements (i.e., when it needs to be available at the receiver side) and impact on the resulting PSNR. These new evaluation criteria fundamentally change the design principles for multimedia communication systems compared to communication systems for traditional delay-insensitive, loss-intolerant applications.

1.2 ORGANIZATION OF THE BOOK

This book aims at providing an in-depth understanding of the theoretical foundations, key design principles, algorithms, and existing standards for the aforementioned multimedia networking and communication scenarios. The book is divided into five major parts.

The first part of the book discusses how multimedia data can be efficiently compressed to enable optimized transmission over the Internet and wireless networks. Unlike traditional compression techniques such as MPEG-2, which were designed solely for storage (e.g., on DVD disks) or transmission over error-free networks with relatively large and guaranteed bandwidth, compression schemes that enable efficient multimedia communication over the Internet and wireless networks need to have the ability to cope with different channel conditions, characterized by different bit error rates, packet loss rates, access bandwidths, or time-varying available bandwidths. Chapter 2 discusses error-resilient techniques for video transmission over such error-prone networks, while Chapter 3 presents algorithms and solutions for error-resilient audio transmission. To cope with the changes in bandwidth, Chapter 4 provides a thorough analysis of the various mechanisms for bandwidth adaptation, as the network often offers heterogeneous, time-varying channel conditions. To effectively cope with adaptive streaming applications or multicasting applications, where a variety of receivers would like to simultaneously access the same multimedia content, Chapter 5 introduces existing and emerging scalable video coding algorithms, while Chapter 6 discusses scalable audio coding.

The second part of the book focuses on efficient solutions for bit stream transmission over IP networks. Chapter 7 introduces the fundamentals of channel protection needed to insulate bit streams from the error-prone nature of the channels over which they are transmitted. Chapter 8 discusses how to effectively model and characterize the complex communication channels within networks such as the Internet. Having an accurate model of the channel becomes paramount when finding an efficient trade-off between the bit rates allocated to source and channel

protection. Chapter 9 focuses on Forward Error Correction (FEC) mechanisms aimed at effectively protecting multimedia bit streams at the application layer. These solutions can successfully exploit the available knowledge of the multimedia bit streams. Chapter 10 focuses on the corresponding retransmission-based mechanisms. Unlike FEC mechanisms, retransmission-based mechanisms can be instantaneously adapted to each channel realization. However, the retransmission-based algorithms are not well suited to the multicast case or the live broadcast scenario, where many receivers are connected to a single sender. FEC mechanisms must be used here instead.

The third part of the book focuses on multimedia transmission over wireless networks. Chapter 11 discusses MAC-centric channel models characterizing the specific behavior of wireless networks, thereby offering insights into the challenges associated with multimedia streaming over such networks. Chapter 12 shows how to cope with these challenges, how the various layers of the protocol stack can collaborate to ensure efficient wireless multimedia communication, and how the cross-layer design deployed at one station influences multiuser interaction and fairness in such environments. Chapter 13 provides various solutions for providing the necessary quality of service guarantees in such wireless environments.

The fourth part of the book discusses efficient multimedia system design, which is essential for ensuring that the streaming solutions are efficiently optimized and deployed. Chapter 14 presents approaches to streaming media on demand as well as live broadcast, while Chapter 15 presents approaches to real-time communication applications such as telephony and conferencing. To ensure the continuous playout of multimedia despite packet loss and jitter, Chapter 16 exploits the "time elastic" behavior of these applications by discussing the concept of adaptive media playout.

The final part of the book presents several advanced topics on multimedia communication. Chapter 17 discusses how multimedia compression and transmission algorithms can take advantage of the multipath diversity existing in the Internet and wireless networks. Chapter 18 presents distributed video coding principles, algorithms, and their applications to, for example, low-cost encoding for multimedia streaming. Chapter 19 introduces the capabilities, architectures, and design principles of building overlays on top of the existing Internet and wireless infrastructures for enhanced support to multimedia applications.

1.3 SUGGESTED USE FOR GRADUATE INSTRUCTION AND SELF-STUDY

This book is intended as a textbook for a graduate-level course on multimedia networking and communication or as reference text for researchers and engineers working in the areas of multimedia communication, multimedia compression,

multimedia systems, wireless communication, and networking. This book can be used for either a semester-length course or a quarter-length course if some of the advanced topics are left for self-study or as part of a research project associated with the class.

One of the best ways to understand the challenges and theory for multimedia communication and networking discussed in this book is through the completion of a multimedia-related project. This is because the importance of the various principles and techniques taught in such a course, as well as their interrelationships, become apparent when solving "real" multimedia communication problems. Students should be encouraged to choose a project topic related to their interests and/or research backgrounds. The summary and further reading sections concluding the various chapters can be used as a starting point for defining relevant class projects. For instance, students having a background on wireless communication can choose a project topic on cross-layer wireless multimedia transmission or multimedia transmission over multihop wireless networks, students having interests on information theory can select projects on joint source-channel coding or distributed source coding, and students with a background on signal, speech, or image processing can investigate topics related to robust multimedia compression, scalable coding, error concealment, or adaptive media playout.

1.4 SUPPLEMENTARY MATERIAL FOR THE BOOK

Supplementary material for this book can be found at `http://books.elsevier.com/companions/0120884801`. This includes an additional chapter to this book, Chapter 20, which presents state-of-the-art techniques for multimedia transmission over peer-to-peer networks. Also, the Web page contains slides, exercises, and additional material for the various chapters, which can be used by potential instructors for a class on multimedia communication and networking. For feedback about the book or the material posted on this Web site, the reader can contact the coeditors of this book, Mihaela van der Schaar (mihaela@ee.ucla.edu) and Phil Chou (pachou@microsoft.com).

ACKNOWLEDGMENTS

The coeditors acknowledge the contributions by our incredibly talented team of chapter authors. We also acknowledge Hyunggon Park, Yiannis Andreopoulos, Andres I. Vila Casado, Cong Shen, Miguel Griot, Jonas B. Borgstrom, and Nicholas Mastronarde, all graduate students in the Electrical Engineering Department at UCLA, for reading initial drafts of the book and providing constructive feedback. We acknowledge the patience and careful editorial work of Chuck Glaser, Rick Adams, Rachel Roumeliotis, and the staff at Elsevier. In addition, Mihaela van der Schaar would like to thank her husband for his help and support, as well as for the many discussions they had on this book.

PART **B**

COMPRESSION

This page intentionally left blank

2

Error-Resilient Coding and Decoding Strategies for Video Communication

Thomas Stockhammer and Waqar Zia

2.1 INTRODUCTION

Video is becoming more and more popular for a large variety of applications and networks. Internet and wireless video, especially, has become part of our daily lives. However, despite many advances in terms of bandwidth and capacity enhancements in different networks, the data transmission rate will always be a scarce resource due to physical limitations, especially for high quality high bit rate applications. Therefore, good compression is as important as ever. Furthermore, real-time delivery of multimedia data is required for several application scenarios, such as conversational applications, streaming, broadcast, or video-on-demand services. Under such real-time constraints, unfortunately the Quality-of-Service (QoS) available in current and next generation networks is in general not sufficient to guarantee error-free delivery to all receivers. Therefore, in addition to the capability of easy integration into existing and future networks, video codecs must provide means of dealing with various transmission impairments. In communication environments, standardized solutions are desirable at terminals to ensure compatibility. That is why video coding standards such as MPEG-4 and H.264/AVC have become popular and attractive for numerous network environments and application scenarios. These standards, like numerous previous standards and more recent standards such as VCI, use a hybrid coding approach, namely Motion Compensated Prediction (MCP). MCP is combined with transform coding of the residual. We will focus on MCP-coded video in the remainder of this chapter and mainly concentrate on tools and features integrated in the latest video coding standard H.264/AVC [19,45] and its test model software JM.

We will focus on specific tools for improved error resilience within standard-compliant MCP-coded video. More advanced error-resilience features, such as multiple description coding, distributed video coding, and combinations with network prioritization and forward error correction, are left to the remaining chapters of this book and the references therein. It is assumed that the reader has some basic knowledge of the encoding and decoding algorithms of MCP-coded video, for example, as discussed in [14].

2.2 VIDEO COMMUNICATION SYSTEMS

2.2.1 End-to-End Video Transmission

Figure 2.1 provides an abstraction of a video transmission system. In order to keep this work focused, we have excluded capturing and display devices, user interfaces, and security issues; most computational complexity issues are also ignored. Components that enhance system performance, for example, a feedback channel, will also be introduced later in this chapter. In contrast to still images, video frames inherently include relative timing information, which has to be maintained to assure perfect reconstruction at the receiver's display. Furthermore, due to significant amounts of spatiotemporal statistical and psychovisual redundancy in natural video sequences, video encoders are capable of reducing the actual amount of transmitted data significantly. However, excessive lossy compression results in noticeable, annoying, or even intolerable artifacts in the decoded video. A trade-off between *rate* and *distortion* is always necessary. Real-time transmission of video adds additional challenges. According to Figure 2.1, the video encoder generates data units containing the compressed video stream, which is stored in an

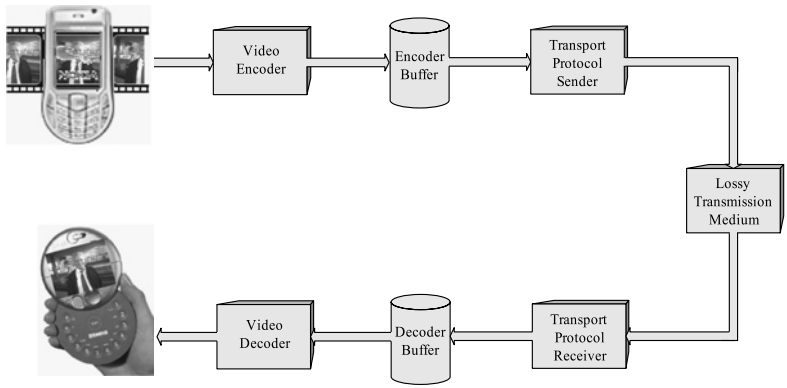


FIGURE 2.1: Simplified lossy video transmission system.

encoder buffer before the transmission. The transmission system may delay, lose, or corrupt individual data units. Furthermore, each processing and transmission step adds some delay, which can be fixed, deterministic, or random. The encoder buffer and the decoder buffer compensate for variable bit rates produced by the encoder as well as channel delay variations to keep the end-to-end delay constant and to maintain the time line at the decoder. Nevertheless, in general the initial playout delay cannot be too excessive and strongly depends on the application constraints.

2.2.2 Video Applications

As discussed in Chapter 1, digitally coded video is used in a wide variety of applications, in different transmission environments. These applications can operate in completely different bit rate ranges. For example, HDTV applications require data rates in the vicinity of 20 Mbit/s, whereas simple download-and-play services such as MMS on mobile devices might be satisfied with 20 Kbit/s, three orders of magnitude less. However, applications themselves have certain characteristics, which are of importance for system design. For example, they can be distinguished by the maximum tolerable end-to-end delay and the possibility of online encoding (in contrast to the transmission of pre-encoded content). In particular, real-time services, such as broadcasting, unicast streaming, and conversational services, come with significant challenges, because generally, reliable delivery of all data cannot be guaranteed. This can be due to the lack of a feedback link in the system or due to constraints on the maximum end-to-end delay. Among these applications, conversational applications with end-to-end delay constraints of less than 200 to 250 ms are most challenging for the system design.

2.2.3 Coded Video Data

In contrast to analog audio, for example, compressed digital video cannot be accessed at any random point due to variable-length entropy coding as well as the syntax and semantics of the encoded video stream. In general, coded video can be viewed as a sequence of data units, referred to as access units in MPEG-4 or network abstraction layer (NAL) units in H.264. The data units themselves are self-contained, at least on a syntactic level, and they can be labeled with data unit-specific information; for example, their relative importance for video reconstruction quality. However, on a semantic level, due to spatial and temporal prediction, the independent compression of data units cannot be guaranteed without significantly harming compression efficiency. A concept of directed acyclic dependency graphs on data units has been introduced in [6], which formalizes these issues. The data units themselves are either directly forwarded to a packet network or encapsulated into a bit or byte stream format containing unique synchronization codes and then injected into a circuit-switched network.

2.2.4 Transmission Impairments

The process of introduction of errors and its effects are markedly different in IP and wireless-based networks. For wireless networks, fading and interference cause *burst errors* in the form of multiple lost bits, while congestion can result in lost packets in an IP network. Nowadays, even for wireless networks, systems include means to detect the presence of errors on physical layer segments and the losses are indicated to higher layers. Intermediate protocol layers such as the User Datagram Protocol (UDP) [32] might decide to completely drop erroneous packets and the encapsulated data units.

Furthermore, video data packets are treated as lost if they are delayed more than a tolerable threshold defined by the application. Hence for the remainder of this chapter we will concentrate on the effects of entire data units lost and present means to deal with such losses in video applications. Detailed description of the processes of introduction of losses in IP and wireless-based networks will be given in Chapter 8 and Chapter 11, respectively.

2.2.5 Data Losses in MCP-Coded Video

Figure 2.2 presents a simplified yet typical system when MCP video is transmitted over error-prone channels. Assume that all macroblocks (MBs) of one frame s_t are contained in a single packet $\mathcal{P}_t$, for example, in an NAL unit in the case of H.264/AVC. Furthermore, assume that this packet is transmitted over a channel that forwards correct packets to the decoder, denoted as $\mathcal{C}_t = 1$, and perfectly detects and discards corrupted packets at the receiver, denoted as $\mathcal{C}_t = 0$.

In case of successful transmission, the packet is forwarded to the regular decoder operation. The prediction information and transform coefficients are re-

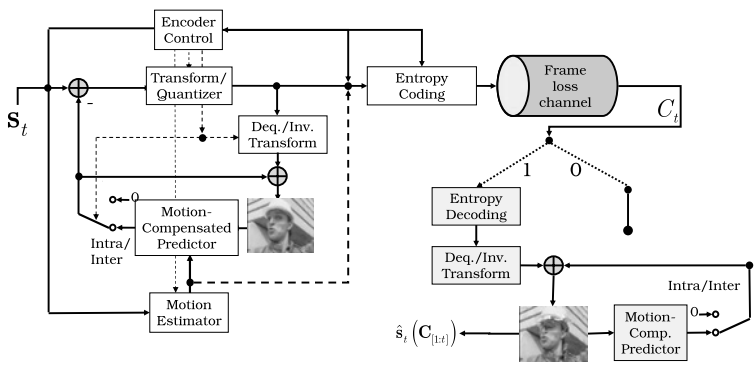


FIGURE 2.2: Simplified lossy video transmission system.

trieved from the coded bit stream to reconstruct frame $\hat{s}_{t-1}$. The frame is forwarded to the display buffer and also to the reference frame buffer to be used in the MCP process to reconstruct following inter-coded frames, for example, $\hat{s}_t$. In the less favorable case that the coded representation of the frame is lost, that is, at our reference time $t = 0$, $\mathcal{C}_t = 0$, so-called *error concealment* is necessary. In the simplest form, the decoder just skips the decoding operation and the display buffer is not updated, that is, the displayed frame is still $\hat{s}_{t-1}$. The viewer will immediately recognize the loss of fluent motion since a continuous display update is not maintained.

However, in addition to the display buffer, the reference frame buffer is also not updated as a result of this data loss. Even in case of successful reception of packet $\mathcal{P}_{t+1}$, the inter-coded frame $\hat{s}_{t+1}$ reconstructed at the decoder will in general not be identical to the reconstructed frame $\tilde{s}_{t+1}$ at the encoder. The reason is obvious, as the encoder and the decoder refer to a different reference signal in the MCP process, resulting in a so-called reconstruction *mismatch*. Therefore, there will again be a mismatch in the reference signal when decoding $\hat{s}_{t+2}$. Hence it is obvious that the loss of a single packet $\mathcal{P}_t$ affects the quality of all the inter-coded frames $\hat{s}_{t+1}$, $\hat{s}_{t+2}$, $\hat{s}_{t+3}$, ... This phenomenon, present in any predictive coding scheme, is called *error propagation*. If predictive coding is applied in the spatial and temporal domains, it is referred to as *spatiotemporal error propagation*.

Therefore, for MCP-coded video, the reconstructed frame at the receiver, $\hat{s}_t$, not only depends on the actual channel behavior $\mathcal{C}_t$, but on the previous channel behavior $\mathcal{C}_{[1:t]} = \{\mathcal{C}_1, \ldots, \mathcal{C}_t\}$ and we write $\hat{s}_t(\mathcal{C}_{[1:t]})$. An example for error propagation is shown in Figure 2.3. The top row presents the sequence with perfect reconstruction; in the bottom row only packet $\mathcal{P}_t$ at time $t = 0$ is lost. Although the remaining packets are again correctly received, the error propagates and is still visible in decoded frame $\hat{s}_{t=8}$. At time $t = 9$, the encoder transmits an intra-coded image, and since no temporal prediction is used for coding this image, temporal error propagation is terminated at this time. It should be noted, however, that even

FIGURE 2.3: Example for error propagation in a typical hybrid video coding system.

with inter-coded images, the effect of a loss is reduced with every correct reception. This is because inter-coded frames might consist of intra-coded regions that do not use temporal prediction. An encoder might decide to use intra-coding when it finds that temporal prediction is inefficient for coding a certain image region.

Following the intra image at $t = 9$, the decoder will be able to perfectly reconstruct the encoded images until another data packet is lost for $t > 9$.

Therefore, a video coding system operating in environments where data units might get lost should provide one or several of the following features:

1. means that allow completely avoiding transmission errors,
2. features that allow minimizing the visual effects of errors in a frame, and
3. features to limit spatial as well as spatiotemporal error propagation in hybrid video coding.

In the remainder of this chapter we restrict ourselves to forward predictive MCP video coding, although most of the concepts generalize to any kind of dependencies. A formal description of packetized video with slice structured coding and error concealment, as well as the extension of operational encoder control for error-prone video transmission, are discussed in Section 2.3.

2.3 ERROR-RESILIENT VIDEO TRANSMISSION

2.3.1 System Overview

The operation of an MCP video coding system in a transmission environment is depicted in Figure 2.4. It extends the simplified presentation in Figure 2.2 by the

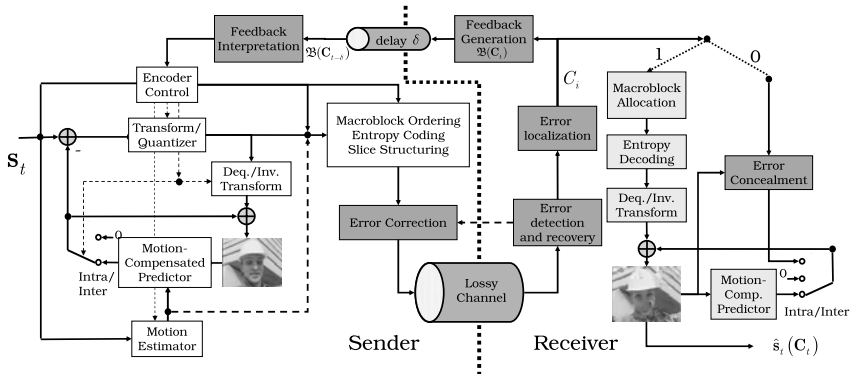


FIGURE 2.4: MCP video coding in packet lossy environment: Error-resilience features and decoder operations.

addition of typical features used when transmitting video over error-prone channels. However, in general, for specific applications not all features are used, but only a suitable subset is extracted. Frequently, the generated video data belonging to a single frame is not encoded as a single data unit, but MBs are grouped in data units and the entropy coding is such that individual data units are syntactically accessible and independent. The generated video data might be processed in a transmission protocol stack and some kind of error control is typically applied, before the video data is transmitted over the lossy channel. Error control features include Forward Error Correction (FEC), Backward Error Correction (BEC), and any prioritization methods, as well as any combinations of those. At the receiver, it is essential that erroneous and missing video data are detected and localized. Commonly, video decoders are fed only with correctly received video data units, or at least with an error indication, that certain video data has been lost. Video data units such as NAL units in H.264 are self-contained and therefore the decoder can assign the decoded MBs to the appropriate locations in the decoded frames. For those positions where no data has been received, error concealment has to be applied. Advanced video coding systems also allow reporting the loss of video data units from the receiver to the video encoder. Depending on the application, the delay, and the accurateness of the information, an online encoder can exploit this information in the encoding process. Likewise, streaming servers can use this information in their decisions. Several of the concepts briefly mentioned in this high-level description of an error-resilient video transmission system will be elaborated and investigated in more detail in remaining sections.

2.3.2 Design Principles

Video coding features such as MB assignments, error control methods, or exploitation of feedback messages can be used exclusively or jointly for error robustness purposes, depending on the application. It is necessary to understand that most error-resilience tools decrease compression efficiency. Therefore, the main goal when transmitting video goes along the spirit of Shannon's famous separation principle [38]: Combine compression efficiency with link layer features that completely avoid losses such that the two aspects, compression and transport, can be completely separated. Nevertheless, in several applications and environments, particularly in low delay situations, error-free transport may be impossible. In these cases, the following system design principles are essential:

1. *Loss correction below codec layer*: Minimize the amount of losses in the wireless channel without completely sacrificing the video bit rate.
2. *Error detection*: If errors are unavoidable, detect and localize erroneous video data.
3. *Prioritization methods*: If losses are unavoidable, at least minimize losses for very important data.

4. *Error recovery and concealment*: In case of losses, minimize the visual impact of losses on the actually distorted image.
5. *Encoder–decoder mismatch avoidance*: In case of losses, limit or completely avoid encoder and decoder mismatch to avoid the annoying effects of error propagation.

This chapter will focus especially on the latter three design principles. However, for completeness, we include a brief overview on the first two aspects. The remainder of this book will treat many of these advanced issues.

2.3.3 Error Control Methods

In wireless systems, below the application layer, error control such as FEC and retransmission protocols are the primary tools for providing QoS. However, the trade-offs among reliability, delay, and bit rate have to be considered. Nevertheless, to compensate the shortcomings of non-QoS-controlled networks, for example, the Internet or some mobile systems, as well as to address total blackout periods caused, for example, by network buffer overflow or a handoff between transmission cells, error control features are introduced at the application layer. For example, broadcast services make use of application-layer FEC schemes. For point-to-point services, selective application layer retransmission schemes have been proposed. For delay-uncritical applications, the Transmission Control Protocol (TCP) [31,40] can provide QoS. The topics of channel protection techniques and FEC will be covered in detail in Chapter 7 and Chapter 9, respectively. We will not deal with these features in the remainder of this chapter, but concentrate on video-related signal processing to introduce reliability and QoS.

2.3.4 Video Compression Tools Related to Error Resilience

Video coding standards such as H.263, MPEG-4, and H.264 only specify the decoder operation in case of reception of an error-free bit stream as well as the syntax and semantics of the video bit stream. Consequently, the deployment of video coding standards still provides a significant amount of freedom for encoders and decoding of erroneous bit streams. Depending on the compression standard used, different compression tools are available that offer some room for error-resilient transmission.

Video compression tools have evolved significantly over time in terms of the error resilience they offer. Early video compression standards, such as H.261, had very limited error-resilience capabilities. Later standards, such as MPEG-1 and MPEG-2, changed little in this regard, since they were tailored mostly for storage applications. With the advent of H.263, things started changing dramatically. The resilience tools of the first version of H.263 [18] had only marginal improvements over MPEG-1; however, later versions of H.263 (referred as H.263+ and H.263++,

respectively) introduced several new tools that were tailored specifically for the purpose of error resilience and will be discussed in this section. These tools resulted in a popular acceptance of this codec; it replaced H.261 in most video communication applications. In parallel to this work, the new emerging standard MPEG-4 Advanced Simple Profile (ASP) [17] opted for an entirely different approach. Some sophisticated resilience tools, such as Reversible Variable Length Coding (RVLC) and resynchronization markers, were introduced. However, despite their strong concept, these tools did not gain wide acceptance. One of the reasons for this is that these tools target to solve the issues of lower layers in the application layer, which is not a widely accepted approach. For example, RVLC can be used at the decoder to reduce the impact of errors in a corrupted data packet. However, as discussed in Section 2.2.4, errors on the physical layer can be detected and lower layers might discard these packets instead of forwarding them to the application.

Up to date, the final chapter in error-resilient video coding is H.264/AVC. This standard is equipped with a wide range of error-resilience tools. Some of these tools are modified and enhanced forms of the ones introduced in H.263++. The following section gives a brief overview of these tools as they are formulated in H.264/AVC and the concepts behind these. Considering the rapid pace of evolution of these tools, it is also important to know the origin of these tools in previous standards.

Some specific error-resilience features such as error-resilient entropy coding schemes and arbitrary slice ordering will not be discussed. The interested reader is referred to [43,60]. It is also worth considering that most features are general enough to be used for multiple purposes rather than being assigned to a specific application. Some of the tools have a dual purpose of increased compression efficiency along with error resilience, which seems to be contradictory initially, but this ambiguity will be resolved. In later sections of this chapter, we will present some of these tools in action in different applications and measure their impact on system performance.

Slice Structured Coding

For typical digital video transmission over networks, it is not suitable to transmit all the compressed data belonging to a complete coded frame in a single data packet for a variety of reasons. Most importantly, variations are expected in the sizes of such data packets because of a varying amount of redundancy in different frames of a sequence. In this case the lower layers have to subdivide the packet to make it suitable for transmission. In case of a loss of a single such division, the decoder might be unable to decode an entire frame with only one synchronization point available for an entire coded frame.

To overcome this issue, *slices* provide spatially distinct resynchronization points within the video data for a single frame (Figure 2.5). A number of MBs are grouped together; this is accomplished by introducing a slice header, which contains syntactic and semantic resynchronization information. The concept of slices (referred to as group of blocks [GOB] in H.261 and H.263) exists in different forms in different standards. Its usage was limited to encapsulate individual rows of MBs in H.263 and MPEG-2. In this case, slices will still result in variable sized data units because of the varying amount of redundancy in different regions of a frame. Slices take their most flexible and advanced form in H.264/AVC. The encoder can select the location of the synchronization points at any MB boundary. Intra prediction and motion vector prediction are not allowed over slice boundaries. An arbitrary number of MBs can be assigned to a slice, which results in different modes of operation. For example, the encoder can decide to allocate either a fixed number of MBs or a fixed number of bits to a slice. The later mode of operation, with a predefined data size of a slice, is especially useful from a network perspective, since the slice size can be better matched to the packet size supported by the network layer. In this case, a loss of a data unit on network layer will result in a loss of a discrete number of slices, and a considerable portion of a picture might remain unaffected by the loss.

Hence in H.264/AVC, slices are the basic output of the video encoder and form an independently accessible entity. Provision of access to those units is provided either by the use of unique synchronization markers or by the appropriate encapsulation in underlying transport protocols. The details of slice structured coding modes and the implications are discussed in Section 2.4.2.

FIGURE 2.5: A sketch of a picture divided into several slices, demarcated by gray boundaries.

Flexible MB Ordering

In previous video compression standards, such as MPEG-1, MPEG-2 and H.263, MBs are processed and transmitted in raster–scan order, starting from the top-left corner of the image to the bottom right. However, if a data unit is lost, this usually results in the loss of a connected area in a single frame.

In order to allow a more flexible transmission order of MBs in a frame in H.264/AVC, Flexible Macroblock Ordering (FMO) allows mapping of MBs to so-called *slice groups*. A slice group itself may contain several slices. For example, in Figure 2.6, each shaded region (a slice group) might be subdivided into several slices. Hence *slice group* can be thought of as an entity similar to a picture consisting of slices in the case when FMO is not used. Therefore, MBs may be transmitted out of raster–scan order in a flexible and efficient way. This can be useful in several cases. For example:

- Several concealment techniques at the decoder rely on the availability of correctly received neighbor MBs to conceal a lost MB. Hence a loss of collocated image areas results in poor concealment. Using FMO, spatially collocated image areas can be interleaved in different slices. This will result in a greater probability that neighboring MB data is available for concealing the lost MB.
- There might exist a Region Of Interest (ROI) within the images of a video sequence, for example, the face of the caller in a video conferencing system. Such regions can be mapped to a separate slice group than the background to offer better protection against losses in the network layer.

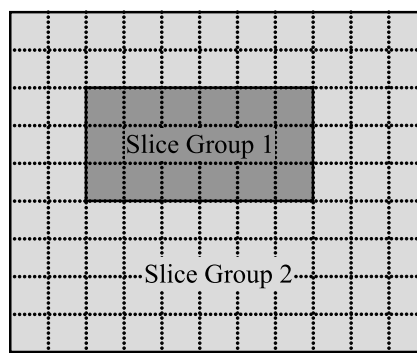


FIGURE 2.6: MBs of a picture (dotted lines) allocated to two slice groups. Light-gray MBs belong to one slice group, and dark-gray MBs belong to the other.

A description of different modes and specific applications of FMO are given in Section 2.4.2.

Scalability

Scalable coding usually refers to a source coder that simultaneously provides encoded version of the same data source at different quality levels by extracting a lower quality reconstruction from a single binary description. Scalable coding can be realized using *embedded bit streams*, that is, the bit stream of a lower resolution is embedded in the bit stream of higher resolution. Unlike one-dimensional sources such as speech or audio, where usually the quality levels are defined by the quantization distortion, for video the quality can be changed in basically three dimensions, namely spatial resolution, temporal resolution or frame rate, and quantization distortion. Scalable video coding is realized in standards in many different variants and will be extensively treated in Chapter 5. Commonly, scalability is synonymously used with a specific type of scalability referred to as *successive refinement*. This specific case addresses the view point that information is added such that the initial reproduction is refined. In this case, the emphasis is on a good initial reproduction.

Data Partitioning

The concept of *data partitioning* originates from the fact that loss of some syntax elements of a bit stream results in a larger degradation of quality compared to others. For example, the loss of MB mode information or motion vector (MV) information will, for most cases, result in a larger distortion compared to loss of a high-frequency transform coefficient. This is intuitive, since, for example, MB mode information is required for interpreting all the remaining MB information at the decoder.

In the case of data loss in the network, data partitioning results in the so-called *graceful degradation* of video quality. Graceful degradation targets the reduction of perceived video quality that is, to some extent, proportionate to the amount of data lost. In this case, the emphasis is on a good final reproduction quality, but at least an intermediate reconstruction is possible.

The concept of categorizing syntax elements in the order of their importance started with MPEG-4 and H.263++. For these standards, video coded data was categorized into header information, motion information, and texture information (transformed residual coefficients), listed here in the order of their importance. Figure 2.7 shows the interleaved structure of data when using the data partitioning mode. For example, combining this concept with that of RVLC and resynchronization markers, it could be possible to retrieve most of header and MV information even for the case of data lost within the transform coefficients partition.

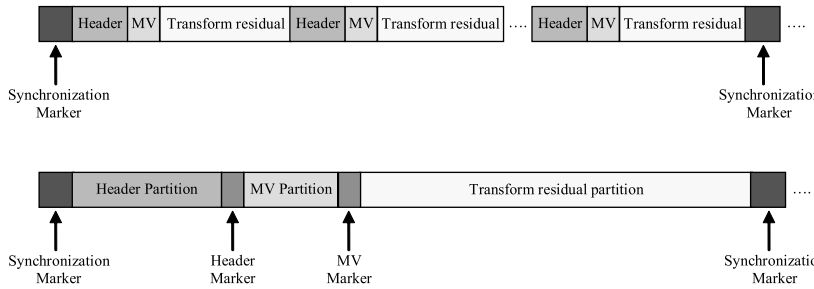


FIGURE 2.7: The layout of a compressed video data without using data partitioning (top) and with data partitioning (bottom) in H.263++. A packet starts with a synchronization marker, while for the data partitioning mode, two additional synchronization points are available, such as the header marker and the MV marker.

In the H.264/AVC data partitioning mode, each slice can be segmented into header and motion information, intra information, and inter texture information by simply distributing the syntax elements to individual data units. Typically, the importance of the individual segments of the partition is in the order of the list. In contrast to MPEG-4, H.264/AVC distinguishes between inter- and intra-texture information because of the more important role of the latter in error mitigation. The partitions of different importance can be protected with Unequal Error Protection (UEP), with the more important data being offered more protection and vice versa. Due to this reordering only on the syntax level, coding efficiency is not sacrificed, but obviously the loss of individual segments still results in error propagation with similar but typically less severe effects as those shown in Figure 2.3. Some detailed investigations of synergies of data partition and UEP can be found in [13,24,42].

Redundant Slices

An H.264/AVC encoder can transmit a redundant version of a normally transmitted slice using possibly different encoding parameters. Such a *redundant slice* can be simply discarded by the decoder during its normal operation. However, in the case when the original slice is lost, this redundant data can be used to reconstruct the lost regions. For example, in a system with frequent data losses, an H.264/AVC encoder can exploit this unique feature to send the redundant, coarsely quantized version of an ROI along with the regular representation of it. Hence the decoder will be capable of displaying the lost ROI, albeit at a lower quality. It is worthwhile to notice that this will still result in an encoder–decoder

mismatch of reference pictures, since the encoder being unaware of the loss uses the original slice as a reference, but this effect will be less severe compared to the case when this tool is not used.

Flexible Reference Frame Concept

Standards such as H.263 version 1 and MPEG-2 allow only a single reference frame for predicting a P-type frame and at most two frames for predicting a B-type frame. However, there is a possibility of significant statistical dependencies between other pictures as well. Hence using more frames than just the recent frame as reference has a dual advantage: increased compression efficiency and improved error resilience at the same time. Here we focus on the latter effect exclusively. This concept has been recognized as especially useful for transmission over error-prone channels.

In prior codecs, if the encoder is aware of the only reference picture being lost at the decoder, the only available option to limit error propagation was to transmit intra-coded information. However, intra-coded data has significantly large size compared to temporally predicted data, which results in further delays and losses on the network. H.263+ and MPEG-4 proposed tools, such as the Reference Picture Selection (RPS), allows flexible selection of a reference picture on a slice or GOB bases. Hence temporal prediction is still possible from other correctly received frames at the decoder. This results in improved error resilience by avoiding using corrupted picture areas as reference. In H.264/AVC, this restrictive concept has been generalized to allow reference frames to be selected in a flexible way on an MB basis (Figure 2.8). There is also the possibility of using two weighted reference signals for MB inter prediction. Frames can be kept in short-term and long-term memory buffers for future reference. This concept can be exploited by the encoder for different purposes, for compression efficiency, for bit rate adaptivity, and for error resilience.

Flexible reference frames can also be used to enable *subsequences* in the compressed stream to effectively enable temporal scalability. The basic idea is to use a subsequence of "anchor frames" at a lower frame rate than the overall sequence frame rate, shown as P frames in Figure 2.9. Other frames are inserted in between these frames to achieve the overall target frame rate, shown as P' frames in Figure 2.9. Here, as an example, every third frame is a P frame. These P' frames can use the low frame rate P frames as reference, but *not* the other way around. This is shown by the chain of prediction arcs in Figure 2.9. If such a P' frame is lost, the error propagates only until the next P is received. Hence P frames are more important to protect against error propagation than P' frames, and some prioritization techniques at lower layers can make use of this fact. This concept is similar to using B frames in prior standards, except that a one-directional predic-

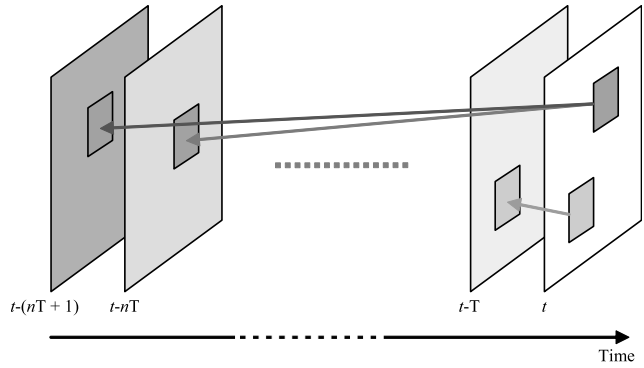


FIGURE 2.8: A sketch of an H.264/AVC inter-predicted frame at a given time t, with different MBs referencing different frames. The frame interval in this sketch is T.

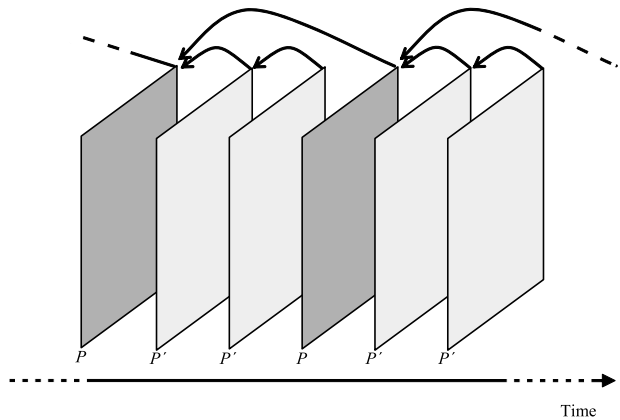


FIGURE 2.9: H.264/AVC inter prediction with subsequences. Arcs show the reference frame used for prediction.

tion chain avoids any buffering overhead as with the bidirectionally predicted B pictures.

Some use cases of the flexible concept specifically for error-resilience purposes are presented in Section 2.5. More details on this mode can be studied in Subsection 2.5.4 and [9,63,64,69].

Intra Information Coding

Even though temporal redundancy might exist in a frame, it is still necessary to have the possibility of switching off temporal prediction in hybrid video coding. This feature enables random access and also provides error robustness. Any video coding standard allows encoding image regions in *intra mode*, such as without reference to a previously coded reference frame. In a straightforward way, completely intra-coded frames might be inserted. These frames will be referred to as "intra frames" in the remainder of this chapter. In H.264/AVC, the flexible reference frame concept allows the usage of several reference frames; not limited to just the temporally preceding frame. Hence in H.264/AVC, intra frames are further distinguished as Instantaneous Decoder Refresh (IDR) frames and "open GOP" intra frames, whereby the latter do not provide the random access property as possibly frames "before" the intra frame are used as reference for "later" predictively coded frames (Figure 2.10).

In addition, intra information can be introduced for just parts of a predictively coded image. Again most video coding standards allow encoding of single MBs for regions that cannot be predicted efficiently or due to any other case the encoder decides for nonpredictive mode. H.264/AVC intra-coded MBs gain significant compression by making use of spatial prediction from neighboring blocks. To limit error propagation, in H.264/AVC this intra mode can be modified such that intra prediction from inter-coded MBs is disallowed. In addition, encoders can also guarantee that MB intra updates result in Gradual Decoding Refresh (GDR), that is, entirely correct output pictures after a certain period of time. Some advanced techniques for the purpose of error resilience, based on intra updates, and their impact on system performance will be discussed in Section 2.5.3.

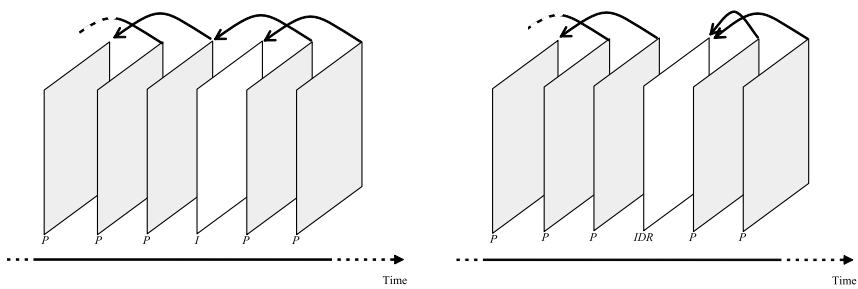


FIGURE 2.10: Inter prediction with open GOP intra "*I*" (left) and *IDR* (right). Temporal prediction (shown by arcs) is not allowed from the frames coded before an IDR frame.

Switching Pictures

H.264/AVC includes a feature that allows applying predictive coding even in the case of different reference signals. This unique feature is enabled by introducing Switching-Predictive (SP) pictures for which the MCP process is performed in the transform domain rather than in the spatial domain and the reference frame is quantized—usually with a finer quantizer than that used for the original frame—before it is forwarded to the reference frame buffer. These so-called primary SP (PSP) frames, which are introduced to the encoded bit stream, are in general slightly less efficient than regular P-frames but significantly more efficient than regular I-frames. The major benefit results from the fact that this quantized reference signal can be generated mismatch free using any other prediction signal. In case that this prediction signal is generated by predictive coding, the frames are referred to as secondary SP (SSP) pictures, which are usually significantly less efficient than P-frames, as an exact reconstruction is necessary. To generate this reference signal without any predictive signal, so-called Switching Intra (SI) pictures can be used. SI pictures are only slightly less inefficient than common intra-coded pictures and can also be used for adaptive error-resilience purposes. Further details on this unique feature within H.264/AVC are covered in Chapter 4 and [22].

2.4 RESYNCHRONIZATION AND ERROR CONCEALMENT

2.4.1 Formalization of H.264 Packetized Video

By the use of slices and slice groups as introduced in Section 2.3, video coding standards, particularly H.264/AVC, provide a flexible and efficient syntax to map the N_{MB} MBs of each frame s_t of the image sequence to individual data units. The encoding of s_t results in one or more data units $\mathcal{P}_i$ with sequence number i. The video transmission system considered is shown in Figure 2.4. Assume that each data unit $\mathcal{P}_i$ is transmitted over a channel that either delivers the data unit $\mathcal{P}_i$ correctly, indicated by $\mathcal{C}_i = 1$, or loses the data unit, that is, $\mathcal{C}_i = 0$. A data unit is also assumed to be lost if it is received after its nominal Decoding Time Stamp (DTS) has expired. We do not consider more complex concepts with multiple decoding deadlines, also referred to as Accelerated Retroactive Decoding [11, 21], in which late data units are processed by the decoder to at least update the reference buffer, resulting in reduced long-term error propagation.

At the receiver, due to the coding restriction of slices and slice groups, as well as with the information in slice headers, the decoder is able to reconstruct the information of each correctly received data unit and its encapsulated slice. The decoded MBs are then distributed according to the mapping $\mathcal{M}$ in the frame. For all MBs positions, for which no data has been received, appropriate error

concealment has to be invoked before the frame is forwarded to the reference and display buffer. The decoded source $\hat{s}_t$ obviously depends on the channel behavior for all the data units $\mathcal{P}_i$ corresponding to the current frame s_t, but due to the predictive coding and error propagation in general, it also depends on the channel behavior of all previous data units, $\mathcal{C}_t \triangleq \mathcal{C}_{[1:i_t]}$. This dependency is expressed as $\hat{s}_t(\mathcal{C}_t)$.

Due to the bidirectional nature of conversational applications, a low-delay, low-bit rate, error-free feedback channel from the receiver to the transmitter, as indicated in Figure 2.4, can be assumed, at least for some applications. This feedback link allows sending some back channel messages. These messages make the transmitter aware of the channel conditions so that it may react to these conditions. These messages are denoted as $\mathcal{B}(\mathcal{C}_t)$. The exact definition and applications of such messages are described in Section 2.5. In our framework we model the feedback link as error free, but the feedback message delay is normalized to the frame rate such that $\mathcal{B}(\mathcal{C}_{t-\delta})$ expresses a version of $\mathcal{B}(\mathcal{C}_t)$ delayed by δ frames, with $\delta = 0, 1, 2, \ldots$. The exploitation of this feedback link and different types of messages having assigned specific semantics in the encoding process are discussed later.

2.4.2 Video Packetization Modes

At the encoder the application of slice structured coding and FMO allows limiting the amount of lost data in case of transmission errors. Especially with the use of FMO, the mapping of MBs to data units basically provides arbitrary flexibility. However, there exist a few typical mapping modes, which are discussed in the following.

Without the use of FMO, the encoder typically can choose between two slice coding options: one with a constant number of MBs, $N_{\mathrm{MB/DU}}$, within one slice resulting in an arbitrary size, and one with the slice size bounded to some maximum number of bytes $S_{\max}$, resulting in an arbitrary number of MBs per slice. Whereas with the former mode, the similar slice types as present in H.263 and MPEG-2 can be formed, the latter is especially useful for introducing some QoS, as commonly the slice size and the resulting packet size determine the data unit loss rate in wireless systems. Examples of the two different packetization modes and the resulting locations of the slice boundaries in the bit stream are shown in Figure 2.11. With the use of FMO, the flexibility of the packetization modes is significantly enhanced, as shown in the examples in Figure 2.12. Features such as slice interleaving, dispersed MB allocation using checkerboard-like patterns, one or several foreground slice groups and one left-over background slice groups, or subpictures within a picture are enabled. Slice interleaving and dispersed MB allocation are especially powerful in conjunction with appropriate error concealment, that is, when the samples of a missing slice are surrounded by many samples of

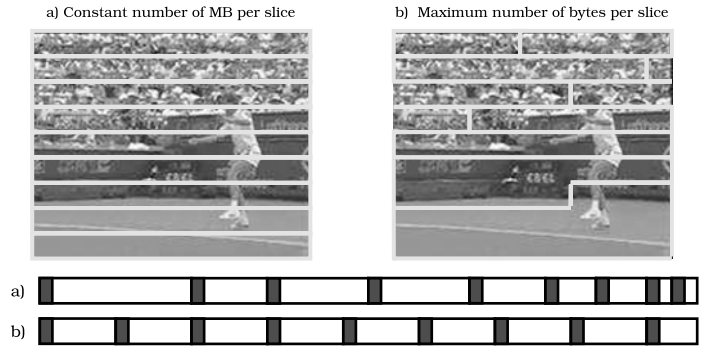


FIGURE 2.11: Different packetization modes: (a) constant number of MBs per slice with variable number of bytes per slices and (b) maximum number of bytes per slice with variable number of MBs per slice.

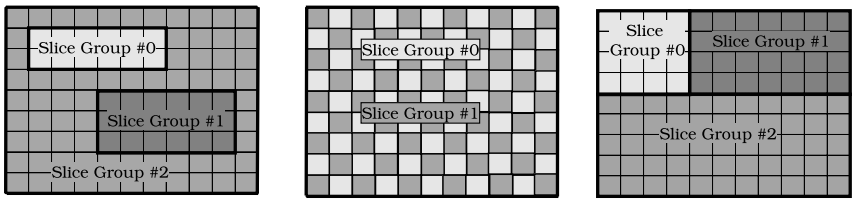


FIGURE 2.12: Specific MB allocation maps: foreground slice groups with one left-over background slice group, checkerboard-like pattern with two slice groups, and subpictures within a picture.

correctly decoded slices. This is discussed in the following section. For dispersed MB allocation typically and most efficiently checkerboard patterns are used, if no specific area of the video is treated with higher priority.

Video data units may also be packetized on a lower transport layer, for example, within RTP [59], by the use of aggregation packets, with which several data units are collected into a single transport packet, or by the use of fragmentation units, that is, a single data unit is distributed over several transport packets.

2.4.3 Error Concealment

With the detection of a lost data unit at the receiver, the decoder conceals the lost image area. Error concealment is a *nonnormative* feature in any video decoder, and a large number of techniques have been proposed that span a wide range of

performance and complexity. The basic idea is that the decoder should generate a representation for the lost area that matches perceptually as close as possible to the lost information without knowing the lost information itself, within a manageable complexity. These techniques are based on *best effort*, with no guarantee of an optimal solution. Since the concealed version of the decoded image will still differ from its corresponding version at the encoding end, error propagation will still occur in the following decoded images until the reference frames are synchronized once again at the encoder and the decoder. This subject will be addressed in detail in Section 2.5.4.

Most popular techniques in this regard are based on a few common assumptions:

- Continuity of image content in spatial domain; natural scene content typically consists of smooth texture.
- Temporal continuity; smooth object motion is more common compared to abrupt scene changes and collocated regions in image tend to have similar motion displacement.

Such techniques exploit the correctly received information of the surrounding area in the spatial and temporal domains to conceal the lost regions. Here we mainly focus on the techniques that conceal each lost MB individually and do not modify the correctly received data.

To simplify the discussion in this section and unless specified otherwise, "data loss" refers to the case that *all* the related information of one or several MBs is lost, for example, MB mode, transformed residual coefficients, and MVs (for the case of inter-coded MBs). This assumption is quite practical as typically a corrupted packet will be detected and discarded before the video decoder.

There exists an exhaustive amount of literature proposing different error concealment techniques. However, only a few simple schemes are commonly used in practical applications. We will put emphasis on error concealment with some practical relevance, but provide reference to other important error concealment methods. In general, error concealment needs to be assessed in terms of performance and complexity.

Spatial Error Concealment

The spatial error concealment technique is based on the assumption of continuity of natural scene content in space. This method generally uses pixel values of surrounding available MBs in the same frame as shown in Figure 2.13. Availability refers to MBs that either have been received correctly or have already been concealed. We consider the case of loss of a 16×16 MB. The most common way of determining the pixel values in a lost MB is by using a weighted sum of the closest boundary pixels of available MBs, with the weights being inversely related to the

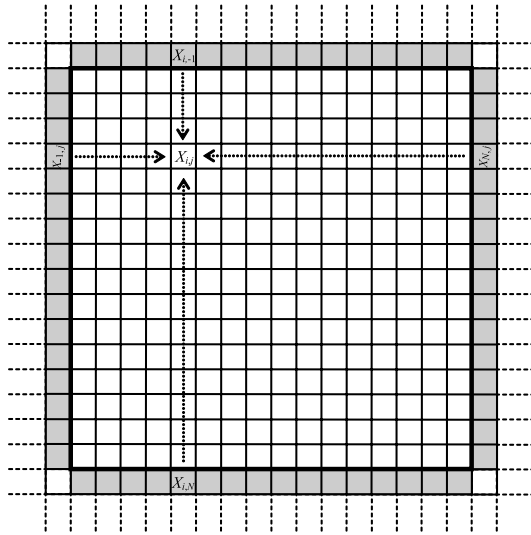


FIGURE 2.13: Pixels used for spatial error concealment (shaded pixels) of a lost MB (thick frame), $M = N = 16$.

distance between the pixel to be concealed and the boundary pixel. For example, at a pixel position i, j in Figure 2.13, an estimate $\hat{X}_{i,j}$ of the lost pixel $X_{i,j}$ is

$$\hat{X}_{i,j} = \alpha\{\beta X_{i,-1} + (1-\beta)X_{i,16}\} + (1-\alpha)\{\gamma X_{-1,j} + (1-\gamma)X_{16,j}\}. \quad (2.1)$$

Here in this equation, α, β, and γ are weighing factors that will determine the relative impact of pixel values of vertical versus horizontal, upper versus lower, and left versus right neighbors, respectively. The top-left pixel of the lost MB is considered as origin. As discussed earlier, the weighing factors are set according to the inverse of the distances from the pixel being estimated. This technique as proposed in [33] is widely used in practice because of its simplicity and low complexity. Since this technique works on the assumption of continuity in spatial domain, discontinuity is avoided in concealed regions of the image. Obviously, this technique will result in blurred reconstruction of the lost region, since natural scene content is not perfectly continuous and lost details will not be recovered. Typically the spatial error concealment technique is never used alone in applications, rather it is combined with other techniques, as discussed in the following sections. It is worthwhile to note that since this technique heavily relies on the availability of horizontal and vertical neighbor pixels, decoders applying this technique can benefit from the application of FMO; for example, by the use of a checkerboard-like pattern.

More sophisticated methods with higher complexity have been proposed in the literature. These methods target to recover some of the lost texture. Some of them are listed in the following.

- In [66], a spatial error concealment technique is proposed that is based on an a priori assumption of continuity of *geometric structure* across the lost region. The available neighboring pixels are used to extract the local geometric structure, which is characterized by a bimodal distribution. Missing pixels are reconstructed by the extracted geometric information.
- Projection onto convex sets in the frequency domain is proposed in [47]. In this method each constraint about the unknown area is formulated as a convex set, and a possible solution is iteratively projected onto each convex set to obtain a refined solution.

Temporal Error Concealment

Temporal error concealment relies on the continuity of a video sequence in time. This technique uses the temporally neighboring areas to conceal lost regions.

In the simplest form of this technique, known as the Previous Frame Concealment (PFC), the spatially corresponding data of the lost MB in the previous frame is copied to the current frame. If the scene has little motion, PFC performs quite well. However, as soon as the region to be concealed is displaced from the corresponding region in the preceding frame, this technique will, in general, result in significant artifacts in the displayed image. However, due to its simplicity, this technique is widely used, especially in decoders with limited processing power.

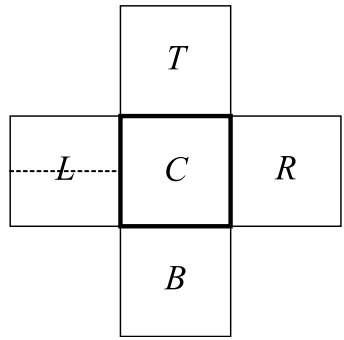


FIGURE 2.14: Neighboring available MBs (T, R, B, and L) used for temporal error concealment of a lost MB C. MB L is encoded in 16×8 inter mode, and the average of its two MVs is used as a candidate.

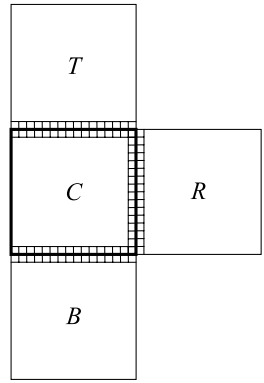


FIGURE 2.15: Boundary pixels of MB C used for the boundary-matching criteria.

A refinement of PFC attempts to reconstruct the image by making an estimate of the lost motion vector. For example, with the assumption of a uniform motion field in the collocated image areas, motion vectors of the neighboring blocks are good candidates to be used as displacement vectors to conceal the lost region. Good candidate MVs for this technique are the MVs of available horizontal and vertical inter-coded neighbor MBs. If a neighboring MB is encoded in an inter mode other than the inter 16×16 mode, one approach is to use the average of the MVs of all the blocks on the boundary of the lost MB. In general, more than one option for the application of displacement vectors exists; for example, using the horizontal neighbor, the vertical neighbor, the zero displacement vector, etc. To select one of the many candidates, a boundary-matching-based technique can, for example, be applied (Figure 2.15). In this case, from the set of all candidate MVs $\mathbf{S}$, the MV $\hat{\upsilon}$ for temporal error concealment is selected according to

$$\begin{aligned}
\varepsilon_T(\upsilon_i) &= \sum_{m=0}^{15} \big(X_{x+m,y}(\upsilon_i) - X_{x+m,y-1}\big)^2, \\
\varepsilon_R(\upsilon_i) &= \sum_{n=0}^{15} \big(X_{x+15,y+n}(\upsilon_i) - X_{x+16,y+n}\big)^2, \\
\varepsilon_B(\upsilon_i) &= \sum_{m=0}^{15} \big(X_{x+m,y+15}(\upsilon_i) - X_{x+m,y+16}\big)^2, \\
\hat{\upsilon} &= \arg\min_{\upsilon_i \epsilon \mathbf{S}} \big(\varepsilon_T(\upsilon_i) + \varepsilon_R(\upsilon_i) + \varepsilon_B(\upsilon_i)\big).
\end{aligned} \tag{2.2}$$

Here, for each motion vector $\upsilon_i \in \mathbf{S}$, errors ε_T, ε_R, and ε_B are calculated for top, right, and bottom edges, respectively. The first term of error functions is the pixel recovered from the reference frame using the selected motion vector υ_i, while the second element is an available boundary pixel of a neighboring MB. The upper-left pixel of the lost MB has a pixel offset x, y. Finally, the vector that results in minimum overall error is selected, since this vector gives a block that possibly fits best in the lost area. Obviously, it is possible that none of the candidate vectors are suitable and in such a case temporal error concealment results in fairly noticeable discontinuity artifacts in the concealed regions.

Several variants and refinements of the temporal error concealment technique have been proposed, usually with some better performance at the expense of sometimes significantly higher complexity. A nonexhaustive list is provided in the following:

- In [4], *overlapped block motion compensation* is proposed. In this case an average of three 16×16 pixel regions is used to conceal the missing MB. One of these regions is the 16×16 pixel data used to conceal the lost MB by the process described earlier, the second and third regions are retrieved from the previous frame by using the motion vectors of horizontal and vertical neighbor MBs, respectively. These three regions are averaged to get the final 16×16 data used for concealment. Averaging in this way can reduce artifacts in the concealed regions.
- In [2], it is proposed to use the median motion vector of the neighboring blocks for temporal concealment. However, the benefits of this technique have been relativized in, for example, [57].
- In [57], Sum of Absolute Differences (SAD) is used instead of Sum of Squared Differences (SSD) for the boundary-matching technique. This results in reduced computational complexity.
- A simpler variant is used in practice [3]: It is proposed to only apply the motion vector of top MB, if available, otherwise zero MV is used (i.e., PFC is used if top MB is not inter coded or is lost as well).
- In [30], a *multihypothesis* error concealment is proposed. This technique makes use of the multiple reference frames available in an H.264/AVC decoder for temporal error concealment. The erroneous block is compensated by a weighted average of correctly received blocks in more than one previous frame. The weighting coefficient used for different blocks can be determined adaptively.
- In [20], the idea presented in [30] is extended. In this proposal, temporal error concealment is used exclusively. However, two variants of temporal error concealment are available: the low-complexity concealment technique governed by (2.2) and the multihypothesis temporal error concealment. The decision as to which technique is used is based on the temporal activity (SAD) in the neighboring regions of the damaged block. For low activity,

the low-complexity technique is used, while multihypothesis temporal error concealment is used for higher activity.

Also, the adaptive combination of spatial concealment with temporal error concealment is of some practical interest and will therefore be discussed in more detail in the following.

Hybrid Concealment

Neither the application of spatial concealment nor temporal concealment alone can provide satisfactory performance: if only spatial concealment is used, concealed regions usually are significantly blurred. Similarly, if only temporal error concealment is applied, significant discontinuities in the concealed regions can occur, especially if the surrounding area cannot provide any or not sufficiently good motion vectors. Hence to achieve better results, the hybrid temporal–spatial technique might be applied. In this technique, MB mode information of reliable and concealed neighbors can be used to decide whether spatial error concealment or temporal error concealment is more suitable. For intra-coded images only spatial concealment is used. For inter-coded images, temporal error concealment is used only if, for example, in the surrounding area more than half of the available neighbor MBs (shown in Figure 2.14) are inter coded. Otherwise, spatial error concealment is used. This ensures that a sufficient number of candidate MVs are available to estimate the lost motion information. We refer to this error concealment as Adaptive temporal and spatial Error Concealment (AEC) in the following.

Other techniques have been proposed to decide between temporal and spatial concealment mode:

- A simple approach in [57] proposes the use of spatial concealment for intra-coded images and temporal error concealment for all inter-coded images invariably.
- In [48], it is suggested that if the residual data in a correctly received neighboring inter-predicted MB is smaller than a threshold, temporal error concealment should be used.

Miscellaneous Techniques

In addition to the signal-domain MB-based approaches, other techniques have been proposed in the literature, for example,

- Model based or object concealment techniques, as proposed in [5,51], do not assume simple a priori assumptions of continuity as given earlier. These techniques are based on the specific texture properties of video objects, and as such are a suitable option for multiobject video codec, that is, MPEG-4.

An object-specific context-based model is built and this model governs the assumptions used for concealment of that object.

- Frequency-domain concealment techniques [16,29] work by reconstructing the lost transform coefficients by using the available coefficients of the neighboring MBs as well as coefficients of the same MB not affected by the loss. These initial proposals are specifically for DCT transform block of 8×8 coefficients. For example, in [16], based on the assumption of continuity of the transform coefficients, lost coefficients are reconstructed as a linear combination of the available transform coefficients. However, noticeable artifacts are introduced by this technique. As a more realistic consideration, in [29] the constraint of continuity holds only at the boundaries of the lost MB in spatial domain.
- In an extension to the spatial and temporal continuity assumptions, it is proposed in [34] that the frames of video content are modeled as a Markov Random Field (MRF). The lost data is suggested to be recovered based on this model. In [35] the authors proposed a less complex but suboptimal alternative to implement this model for error concealment. For example, for temporal error concealment, only the boundary pixels of the lost MB are predicted based on a MAP estimate, instead of predicting the entire MB. These predicted pixels are used to estimate the best predicted motion vector to be used for temporal error concealment. In [39], the MAP estimate is used to refine an initial estimate obtained from temporal error concealment.

Selected results

A few selected results from various important concealment techniques are presented in Figure 2.16. From left to right, a sample concealed frame when using PFC, spatial, temporal, and AEC is shown. PFC simply replaces the missing information by the information at the same location in the temporally preceding frame. Hence, it shows artifacts in the global motion part of the background as well as

FIGURE 2.16: Performance of different error concealment strategies: PFC, spatial concealment only, temporal error concealment only, and AEC.

in the foreground. Spatial error concealment based on weighted pixel averaging smoothes the erroneously decoded image and removes strange block artifacts, but also many details. Temporal error concealment relying on motion vector reconstruction with boundary-matching-based techniques keeps details, but results in strange artifacts in uncovered areas. Finally, AEC—a combination of temporal and spatial error concealment—keeps many details but also avoids strange block artifacts and is therefore very appropriate with feasible complexity. In the remainder of this chapter we will assume exclusively AEC, which reduces to PFC in the case that all MBs of a picture are transmitted in a single packet.

For a further detailed study of error concealment techniques, the reader is referred to [36,52,54,56] and the references therein.

2.4.4 Selected Performance Results for Wireless Test Conditions

To get an insight in error-resilient video coding for 3G mobile communication scenarios, we take a look at a few selected results. The simulated scenario is of a packet-switched conversational application and is specified in detail by 3GPP [8]. This application is characterized by its stringent low-delay and low-complexity requirements, since the processing has to be done in real time on hand-held devices. As a result, the maximum allowed buffering at the encoder is limited to 250 ms and only the first frame is encoded as intra, to limit any delays caused by buffering overheads. A simple random intra MB refresh technique is used, with 5% MBs of every frame coded in intra mode. The most recent frame is used for motion compensation to limit the complexity. With these limitations, we observe the impact of slice size on error resilience of the application. Two channel configurations are compared: one with moderate Radio Link Control (RLC) Packet Data Unit (PDU) loss rate of 0.5% and the other with a higher loss rate of 1.5%. The Radio Access Bearer (RAB) in this test supports transmission of 128 Kbps, with a radio frame size of 320 bytes. Here as an example we use the Quarter Common Intermediate Format (QCIF) sized test sequence *foreman* at 15 frames per second. The encoder is configured to match the maximum throughput of channel while taking into account packetization overheads. The criterion used here as a metric of perceived video quality is PSNR of luma (Y) signal.

Figure 2.17 compares the Y PSNR of the decoded video at a loss rate of 1.5% for two cases: transmitting an entire frame in a slice versus a fixed slice size of 150 bytes. At the given bit rate, a compressed frame has an average size of roughly 1000 bytes. The error-free performance for both cases is also plotted as a reference. Obviously, using a smaller slice size of 150 bytes results in typically lower PSNR in an error-free case because of two reasons: increased packetization overhead and prediction limitations on slice boundaries. However, this configuration outperforms in the case of lossy channel throughout the observed period. A few selected frames are also presented for comparison. The effects of losses already

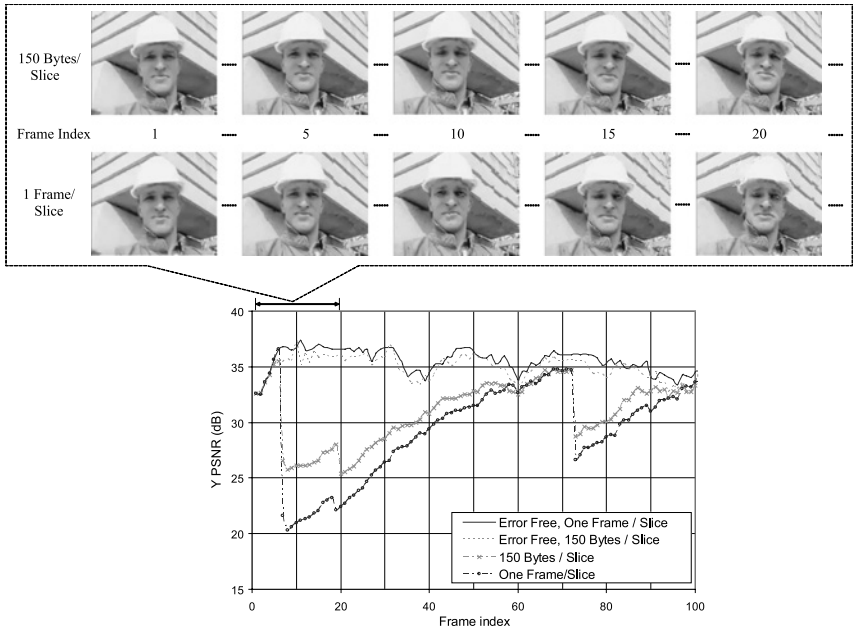


FIGURE 2.17: (Bottom) Plot of Y PSNR with two different slice modes. Results for an error-free case are given as a reference. (Top) A few selected frames for the two slice modes for comparison.

start appearing in the fifth frame. While transmitting one frame per slice results in loss of an entire frame for a lost RLC PDU, the loss affects only a small area of the image for fixed slice size. The spatiotemporal error propagation is much smaller in this case.

Figure 2.18 shows the comparative effects of various slice sizes for different channel loss scenarios. A single point on the curves is obtained by averaging the Y PSNR, denoted as $\overline{\text{PSNR}}$, for several channel realizations to achieve higher statistical significance. The error-free curve shows the effects of reduced compression and hence smaller $\overline{\text{PSNR}}$ for smaller slice sizes. However, at a loss rate of 0.5%, the drawbacks of using larger slice sizes become obvious. The advantage of using slice sizes smaller than 350 bytes does not sufficiently compensate for their overhead. However, increasing slice size beyond this results in a drop of $\overline{\text{PSNR}}$. This is because of a greater portion of a frame affected by a lost RLC

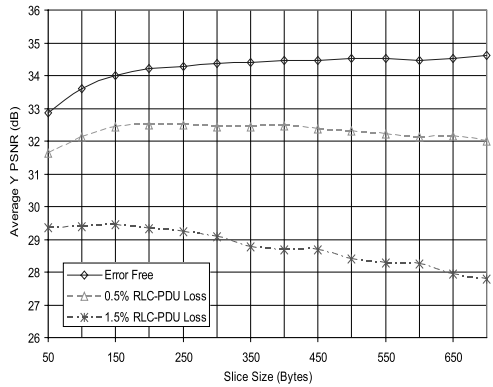


FIGURE 2.18: Plot of $\overline{\text{PSNR}}$ versus slice size with RLC PDU loss rate as a parameter.

PDU. The performance degradation is much more drastic for a loss rate of 1.5%, shown by a significant drop of $\overline{\text{PSNR}}$ for larger slice sizes.

2.5 ERROR MITIGATION

2.5.1 Motivation

As already discussed, error propagation is the major problem when transmitting MCP-coded video over lossy channels. Therefore, if the encoder is aware that the channel will likely be lossy or even knows that the decoder has experienced the loss of certain data units, it should change its encoding behavior, despite sacrificing some compression efficiency. To illustrate this, selected frames for different encoding strategies when transmitting over channels with the same bit rate and error rate constraints are shown in Figure 2.19. The first line, referred to as (a), shows the case where no specific error-resilience tools are applied. The already elaborated problem of error propagation is obvious in later frames. For the sequence in the second line, referred to as (b), the same bit rate and error statistics are applied, but the encoder chooses to select intra-coded MBs in a suitable way. It can be observed that the error propagation is less of an issue but that some residual artifacts are still visible. In addition, the error-free video has lower quality as its compression efficiency is reduced due to the increased amount of intracoding under an identical bit rate constraint. Error propagation can be completely avoided if interactive error control is used, as shown in the third row, labeled with (c). However, in this case also, compression efficiency is sacrificed, especially if necessary feedback of the decoder state is delayed. Details on the appropriate selection of

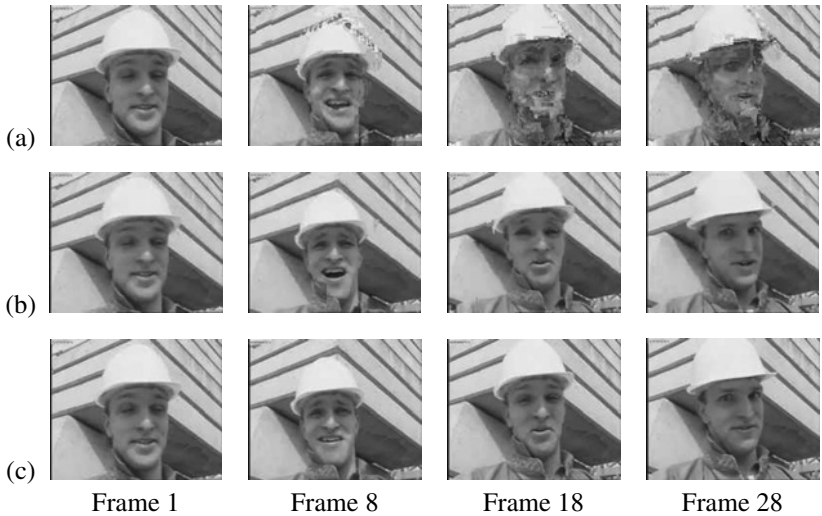


FIGURE 2.19: Selected frames of a decoded video sequence for a packet lossy channel with same bit rate and error constraints: (a) no error robustness, (b) adaptive intra updates, and (c) interactive error control.

MB modes in error-prone environments, especially taking into account the trade-off between quantization distortion and reduced error propagation, are discussed in the following.

2.5.2 Operational Encoder Control

The tools for increased error resilience in *hybrid* video coding, in particular those to limit error propagation, do not significantly differ from the ones used for compression efficiency. Features such as multiframe prediction or intra coding of individual MBs are not primarily error-resilience tools. They are mainly used to increase coding efficiency in error-free environments, although design freedom is left to the video encoder. The encoder implementation is responsible for appropriate selection of one of the many different encoding parameters, the so-called *operational coder control*. Thereby, the encoder must take into account constraints imposed by the application in terms of bit rate, encoding and transmission delay, complexity, and buffer size. When a standard decoder is used, such as an H.264/AVC compliant decoder, the encoding parameters should be selected by the encoder such that good rate–distortion performance is achieved. Since the encoder is limited by the syntax of the standard, this problem is referred to as *syntax-constrained rate–distortion optimization* [28]. In case of H.264/AVC, for

example, the encoder must appropriately select parameters such as motion vectors, MB modes, quantization parameters, reference frames, or spatial and temporal resolution, as shown in Figure 2.20. This also means that bad decisions at the encoder can lead to poor results in coding efficiency, error resilience, or both. For compression efficiency, operational encoder control based on Lagrangian multiplier techniques has been proposed. The distortion d_{b,m_b} usually (at least in the H.264/AVC test model) reflects the SSD between the original MB $\mathbf{s}_b$ and the reconstructed version of the MB $\tilde{\mathbf{s}}_{b,m}$ if coded with option m, that is,

$$d_{b,m} = \sum_i |s_{b,i} - \tilde{s}_{b,m,i}|^2, \tag{2.3}$$

and the rate $r_{b,m}$ is defined by the number of bits necessary to code MB b with option m. Finally, the coding mode is selected for MB b as

$$\forall_b \quad m^*{}_b = \arg\min_{m \in \mathcal{O}} (d_{b,m} + \lambda_{\mathcal{O}} r_{b,m}), \tag{2.4}$$

whereby $\mathcal{O}$ defines the set of selectable options, for example, MB modes. For the Lagrangian parameter $\lambda_{\mathcal{O}}$ it is proposed in [46] and [62] that if the SSD is applied as a distortion measure, then $\lambda_{\mathcal{O}}$ should be directly proportional to the square of the step size Δ of a uniform quantizer applied. The procedure in (2.4) can be applied to select motion vectors, reference frames, and MB modes. However, it is

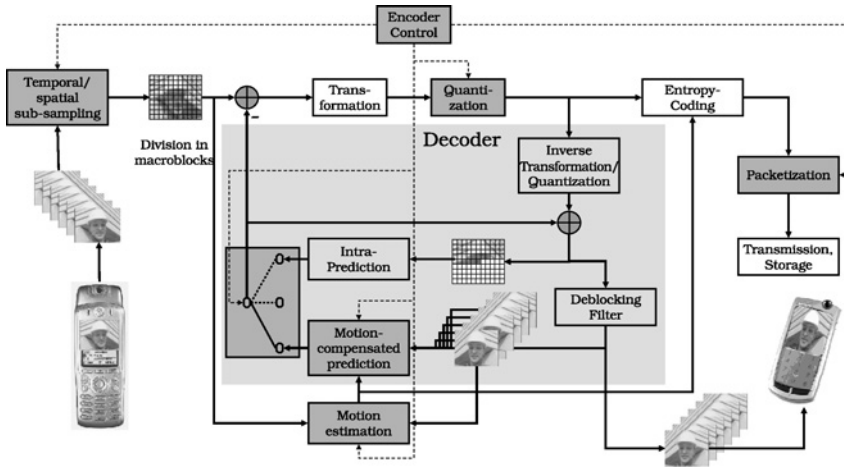


FIGURE 2.20: H.264/AVC video encoder with selectable encoding parameters highlighted.

obviously contradictory if the same decision procedure is applied to obtain good selections for compression efficiency and error resilience. This will be further discussed in the following.

2.5.3 Adaptive Intra Updates

In the presence of errors it has long been recognized that the introduction of more frequent nonpredictively coded image parts is of major importance. In early work on this subject, for example, [15,25,68], it has been proposed to introduce intra-coded MBs, regularly, randomly, or preferably in a certain pseudo-random update pattern. In addition, sequence characteristics and bit rate influence the appropriate percentage of intra updates.

Recognizing this, it has been proposed [67,7,61] to modify the selection of the coding modes according to (2.4) to take into account the influence of the lossy channel. When encoding MB b with a certain coding mode m_b, it is suggested to replace the encoding distortion $d_{b,m}$ by the decoder distortion

$$\tilde{d}_{b,m}(\mathcal{C}_t) \triangleq \left\| \mathbf{s}_{b,t} - \hat{\mathbf{s}}_{b,t}(\mathcal{C}_t, m) \right\|^2, \tag{2.5}$$

which obviously depends on the reconstructed pixel values $\hat{s}_t(\mathcal{C}_t, m)$ and therefore also on the channel behavior $\mathcal{C}_t$ and the selected coding mode m.

In general, the channel behavior is not deterministic and the channel realization $\mathcal{C}$, observed by the decoder, is unknown to the encoder. Thus it is not possible to directly determine the decoder distortion (2.5) at the encoder. However, we can assume that the encoder has at least some knowledge of the statistics of the random channel behavior, denoted as $\hat{\mathcal{C}}_t$. In a Real-Time Transport Protocol (RTP) [37] environment, the Real-Time Control Protocol (RTCP), for example, can use a feedback channel to send receiver reports on the experienced loss and delays statistics, which allow the encoder to incorporate these statistics into the encoding process. Assume that the statistics on the loss process are perfectly known to encoder, that is, $\mathcal{B}(\mathcal{C}_t) = \hat{\mathcal{C}}_t$, and assume that the loss process is stationary. Then, the encoder is able to compute the expected distortion

$$\overline{d}_{b,m} \triangleq \mathbb{E}_{\hat{\mathcal{C}}_t}\left\{\tilde{d}_{b,m}(\hat{\mathcal{C}}_t)\right\} = \mathbb{E}_{\hat{\mathcal{C}}_t}\left\{\left\| \mathbf{s}_{b,t} - \hat{\mathbf{s}}_{b,t}(\hat{\mathcal{C}}_t, m) \right\|^2\right\}. \tag{2.6}$$

A similar procedure can be applied to decisions on reference frames and motion vectors. The selection of motion vectors based on the expected distortion has, for example, been proposed in [65]. The estimation of the squared expected pixel distortion in packet loss environment has been addressed in several contributions. For example, in [7,23], and [61], methods used to estimate the distortion introduced due to transmission errors and the resulting error propagation have been proposed. In all these proposals the quantization noise and the distortion introduced by the

transmission errors (the so-called drift noise) are linearly combined. Since the encoder needs to keep track of an estimated pixel distortion, additional complexity and memory are required in the encoder. The most recognized method, the so-called Recursive Optimal per-Pixel Estimate (ROPE) algorithm [67], however, provides an accurate estimation for baseline H.263 and MPEG-4 simple profile-like algorithms, using simple temporal error concealment, by keeping track of the first and second moment of the decoded pixel value $\tilde{s}(\mathcal{C}_t)$, namely $\mathbb{E}\{\tilde{s}(\mathcal{C}_t)\}$ and $\mathbb{E}\{\tilde{s}^2(\mathcal{C}_t)\}$, respectively.

A powerful yet complex method has been proposed in [44] by applying a Monte Carlo–like method. An estimate of the decoder distortion $\overline{d}_{b,m}$ in (2.6) is obtained as

$$\overline{d}_{b,m}^{(N_{\mathcal{C}})} \triangleq \frac{1}{N_{\mathcal{C}}} \sum_{n=1}^{N_{\mathcal{C}}} \tilde{d}_{b,m}(\mathcal{C}_{n,t}) = \frac{1}{N_{\mathcal{C}}} \sum_{n=1}^{N_{\mathcal{C}}} \left\| \mathbf{s}_{b,t} - \hat{\mathbf{s}}_{b,t}(\mathcal{C}_{n,t}, m) \right\|^p, \tag{2.7}$$

with $\mathcal{C}_{n,t}$, $n = 1, \ldots, N_{\mathcal{C}}$, representing $N_{\mathcal{C}}$ independent realizations of the random channel $\hat{\mathcal{C}}_t$, and estimate of the loss probability at the receiver represented as p. An interpretation of (2.7) leads to a simple solution to estimate the expected pixel distortion $\overline{d}_{b,m}$. For more details we refer to [44]. To obtain an estimate of the loss probability p at the receiver, the feedback channel can be used in practical systems.

2.5.4 Interactive Error Control

The availability of a feedback channel, especially for conversational applications, has led to different standardization and research activities in recent years to include this feedback in the video encoding process. Assume that, in contrast to the previous scenario where only the statistics of the channel process $\hat{\mathcal{C}}$ are known to the encoder, in the case of timely feedback we can even assume that a δ-frame delayed version $\mathcal{C}_{t-\delta}$ of the loss process experienced at the receiver is known at the encoder. This characteristic can be conveyed from the decoder to the encoder by sending acknowledgment for correctly received data units, negative acknowledgment messages for missing slices, or both types of messages.

In less time-critical applications, such as streaming or downloading, the encoder could obviously decide to retransmit lost data units in case it has stored a backup of the data unit at the transmitter. However, in low-delay applications the retransmitted data units, especially in end-to-end connections, would in general arrive too late to be useful at the decoder. In case of online encoding, the observed and possibly delayed receiver channel realization, $\mathcal{C}_{t-\delta}$, can still be useful to the encoder, although the erroneous frame has already been decoded and concealed at the decoder. The basic goal of these approaches is to reduce, limit, or even completely avoid error propagation by integrating the decoder state information into the encoding process.

The exploitation of the observed channel at the encoder has been introduced in [41] and [12] under the acronym *Error Tracking* for standards such as MPEG-2, H.261 or H.263 version 1, but has been limited by the reduced syntax capabilities of these video standards. When receiving the information that a certain data unit—typically including the coded representation of several or all MBs of a certain frame $\mathbf{s}_{t-\delta}$—has not been received correctly at the decoder, the encoder attempts to track the error to obtain an estimate of the decoded frame $\hat{\mathbf{s}}_{t-1}$ serving as reference for the frame to be encoded, $\mathbf{s}_t$. Appropriate actions after having tracked the error are discussed in [12,41,53,56,61]. However, all these concepts have in common that error propagation in frame $\hat{\mathbf{s}}_t$ is only removed if frames $\hat{\mathbf{s}}_{t-\delta+1}, \ldots, \hat{\mathbf{s}}_{t-1}$ have been received at the decoder without any error.

Nevertheless, this promising performance when exploiting decoder state information at the encoder has been recognized by standardization bodies, and the problem of continuing error propagation has been addressed by extending the syntax of existing standards. In MPEG-4 [17, version 2] a tool to stop temporal error propagation has been introduced under the acronym New Prediction (NEWPRED) [10,27,50]. Similarly, in H.263+ Annex N [18, Annex N] RPS for each Group-of-Blocks (GOB) is specified. If combined with slice structured mode as specified in H.263+ Annex K [18, Annex K], as well as Independent Segment Decoding (ISD) as specified in H.263+ Annex R [18, Annex R], the same NEWPRED techniques can be applied within the H.263 codec family.

NEWPRED relies on the availability of timely feedback, online encoding, and the possibility that the encoder can choose other reference frames than the temporally preceding ones. In addition, it allows one to completely eliminate error propagation in frame $\hat{\mathbf{s}}_t$ even if additional errors have occurred for the transmission of frames $\hat{\mathbf{s}}_{t-\delta+1}, \ldots, \hat{\mathbf{s}}_{t-1}$. Different encoder operation modes have been discussed in the literature [10], which can basically be distinguished in a mode where only acknowledged areas are used for reference and another mode, in which the operation is only altered when information is received that the decoder is missing some data units.

In H.263++ Annex U [18, Annex U], NEWPRED was introduced exclusively for the purpose of improving error resilience. In H.264/AVC, the extended syntax allowing selection of reference frames on an MB or even sub-MB basis has a dual impact: enhanced compression efficiency and, at the same time, ease of incorporating methods for limiting error propagation [61]. We will in the following introduce conceptual operation modes when combining decoder state information in the encoding process.

Therefore, we assume that at the encoder each generated data unit $\mathcal{P}_i$ is assigned a decoder state $\mathcal{C}_{\text{enc},i} \in \{\text{ACK}, \text{NAK}, \text{OAK}\}$, whereby $\mathcal{C}_{\text{enc},i} = \text{ACK}$ reflects that data unit $\mathcal{P}_i$ is known to be correctly received at the decoder, $\mathcal{C}_{\text{enc},i} = \text{NAK}$ reflects that data unit $\mathcal{P}_i$ is known to be missing at the decoder, and

$\mathcal{C}_{\text{enc},i} = \text{OAK}$ reflects that for data unit $\mathcal{P}_i$ the acknowledgment message is still outstanding and it is not known whether this data unit will be received correctly.

With feedback messages conveying the observed channel state at the receiver, that is, $\mathcal{B}(\mathcal{C}_t) = \mathcal{C}_t$, and a back channel that delays the back channel messages by δ frames, we assume in the remainder that for the encoding of s_t, the encoder is aware of the following information:

$$\mathcal{C}_{\text{enc},i} = \begin{cases} \text{ACK} & \text{if } \tau_{\text{PTS},i} \leq \tau_{s,t-\delta} \text{ and } \mathcal{C}_i = 1, \\ \text{NAK} & \text{if } \tau_{\text{PTS},i} \leq \tau_{s,t-\delta} \text{ and } \mathcal{C}_i = 0, \\ \text{OAK} & \text{if } \tau_{\text{PTS},i} \geqslant \tau_{s,t-\delta}, \end{cases} \tag{2.8}$$

where $\tau_{\text{PTS},i}$ is the Presentation Time Stamp (PTS) of $\mathcal{P}_i$ and $\tau_{s,t-\delta}$ is the sampling time of $s_{t-\delta}$.

This information about the decoder state $\mathcal{C}_{\text{enc},i}$ can be integrated in a modified rate–distortion optimized operational encoder control similar to what has been discussed in Subsection 2.5.2. In this case the MB mode m_b^* is selected from a modified set of options, $\hat{\mathcal{O}}$, with a modified distortion $\hat{d}_{b,m}$ for each selected option m as

$$\forall_b \quad m_b^* = \arg\min_{m \in \hat{\mathcal{O}}} (\hat{d}_{b,m} + \lambda_{\hat{\mathcal{O}}} r_{b,m}). \tag{2.9}$$

In the following we distinguish four different operation modes, which differ only by the set of coding options available to the encoder in the encoding process, $\hat{\mathcal{O}}$, as well as the applied distortion metric, $\hat{d}_{b,m}$. The encoder's reaction to delayed positive acknowledgment (ACK) and negative acknowledgment (NAK) messages is shown in Figure 2.21, assuming that frame d is lost and the feedback delay is $\delta = 2$ frames for three different feedback modes.

Feedback Mode 1: Acknowledged Reference Area Only

Figure 2.21a shows this operation mode: Only the decoded representation of data units $\mathcal{P}_i$ that have been positively acknowledged at the encoder, that is, $\mathcal{C}_{\text{enc},i} = \text{ACK}$, are allowed to be referenced in the encoding process. In the context of operational encoder control, this is formalized by applying the encoding distortion in (2.9), that is, $\hat{d}_{b,m} = d_{b,m}$, as well as the set of encoding options that is restricted to acknowledged areas only, that is, $\hat{\mathcal{O}} = \mathcal{O}_{\text{ACK},t}$. Note that the restricted option set $\mathcal{O}_{\text{ACK},t}$ depends on the frame to be encoded and is applied to the motion estimation and reference frame selection process. Obviously, if no reference area is available, the option set is restricted to intra modes only, or if no satisfying match is found in the accessible reference area, intra coding is applied. With this mode in use, an error might still be visible in the presentation of a single frame; however, error propagation and reference frame mismatch are completely avoided.

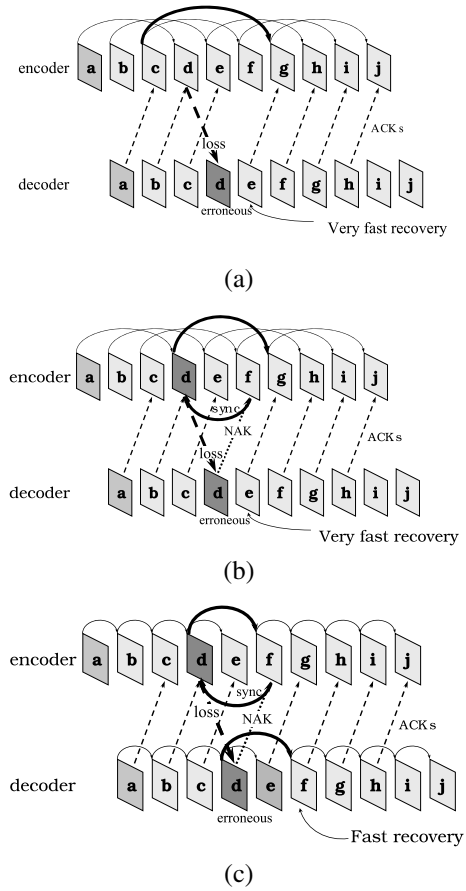


FIGURE 2.21: Operation of different feedback modes. (a) Feedback Mode 1. (b) Feedback Mode 2. (c) Feedback Mode 3.

Figure 2.22a shows the performance in terms of average Peak Signal-to-Noise Ratio (PSNR), denoted by $\overline{\text{PSNR}}$, for feedback mode 1 with different feedback delays δ compared to the channel-adaptive mode selection scheme for *foreman*, error pattern 10 (as given in test conditions specified in [58]), AEC, and $N_{\text{MB/DU}} = 33$. The number of reference frames is $N_{\text{ref}} = 5$, except for $\delta = 8$ with $N_{\text{ref}} = 10$. The results show that for any delay this system with feedback outperforms the best system without any feedback. For small delays, the gains are significant and for the same average PSNR the bit rate is less than 50% compared to the forward-only mode. With increasing delay the gains are reduced, but compared with the

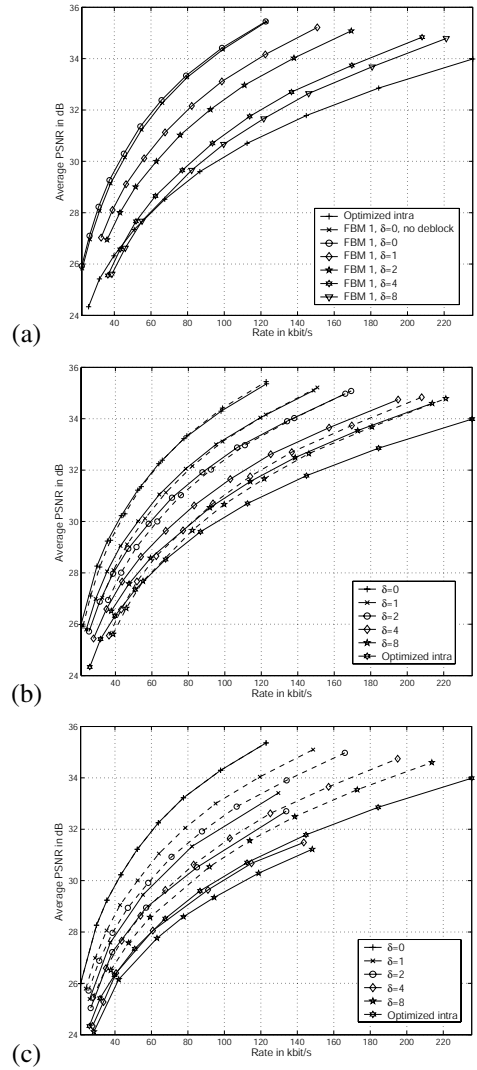


FIGURE 2.22: Average PSNR ($\overline{\text{PSNR}}$) versus bit rate for different feedback modes for sequence *foreman*. (a) Average PSNR ($\overline{\text{PSNR}}$) versus bit rate for Feedback Mode 1. (b) Average PSNR ($\overline{\text{PSNR}}$) versus bit rate for Feedback Mode 2 (solid lines), Feedback mode 1 replotted for comparison (dashed lines). (c) Average PSNR ($\overline{\text{PSNR}}$) versus bit rate for Feedback Mode 3 (solid lines), Feedback mode 2 replotted for comparison (dashed lines).

highly complex mode decision without feedback, this method is still very attractive. Obviously, these high delay results are strongly sequence dependent but for other sequences similar results have been verified.

Feedback Mode 2: Synchronized Reference Frames

Feedback mode 2 as shown in Figure 2.21b differs from mode 1 in that not only positively acknowledged data units but also a concealed version of data units with decoder state $\mathcal{C}_{\text{enc},i} = \text{NAK}$ are allowed to be referenced. This is formalized by applying the encoding distortion in (2.9), that is, $\hat{d}_{b,m} = d_{b,m}$, but the restricted reference area and the option set in this case also include concealed image parts, $\hat{\mathcal{O}} = \mathcal{O}_{\text{NAK},t} \supseteq \mathcal{O}_{\text{ACK},t}$. The critical aspect when operating in this mode results from the fact that for the reference frames to be synchronized the encoder must apply exactly the same error concealment as the decoder.

Figure 2.22b shows the performance in terms of average PSNR, denoted as $\overline{\text{PSNR}}$, for feedback mode 2 with different feedback delays δ compared to the curves in Figure 2.22a for the same parameters. The results for feedback mode 2 show similar results as for feedback mode 1. However, the advantage of feedback mode 2 can be seen in two cases: for low bit rates and for delays $\delta < N_{\text{ref}} - 1$. This is so because referencing concealed areas is preferred over intra coding by the rate–distortion optimization. For higher bit rates this advantage vanishes as the intra mode is preferred anyways over the selection of "bad" reference areas. For delay $\delta = 4$ with $N_{\text{ref}} = 5$, that is, only a single reference frame is available at the encoder, the gains of feedback mode 2 are more obvious, since for feedback mode 1, in case of a lost slice, the encoder basically is forced to use intra coding.

Feedback Mode 3: Regular Prediction with Limited Error Propagation

Feedback modes 1 and 2 are mainly suitable in cases of higher loss rates. If the loss rates are low or negligible, the performance is significantly degraded by the longer prediction chains due to the feedback delay. Therefore, in feedback mode 3 as shown in Figure 2.21c it is proposed to only alter the prediction in the encoder in case of the reception of a NAK. Again, the encoding distortion in (2.9) is applied, that is, $\hat{d}_{b,m} = d_{b,m}$, but the reference area and the option set in this case are altered only in case of receiving a NAK to already acknowledged image parts, that is, $\hat{\mathcal{O}} = \mathcal{O}'_{\text{ACK},t}$, or, as applied in our case to acknowledged and concealed image parts, $\hat{\mathcal{O}} = \mathcal{O}'_{\text{NAK},t}$. Areas that are possibly corrupted by error propagation are also excluded as references. This mode obviously performs well in cases of lower error rates. However, for higher error rates error propagation still occurs quite frequently.

Figure 2.22c shows the performance in terms of average PSNR for feedback mode 3 compared to channel-adaptive mode selection and feedback mode 2, again for the same parameters as in Figure 2.22a. Note that feedback mode 2 and feedback mode 3 are identical for zero feedback delay. However, surprisingly, for increasing delay, feedback mode 3 performs significantly worse than feedback mode 2. The error propagation, though only present for at maximum $\delta - 1$ frames, degrades the overall quality much more significantly; the gain in compression efficiency cannot compensate the distortion due to packet losses. Obviously, the performance depends on the sequence characteristics and especially on the loss rate. For lower rates it is expected (and shown later) that the differences between feedback modes 2 and 3 are less significant, but in general feedback mode 2 is also preferable over feedback mode 3 from the subjective performance.

Feedback Mode 4: Unrestricted Reference Areas with Expected Distortion Update

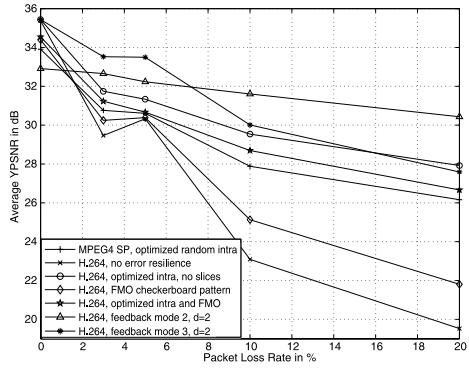
For completeness we present an even more powerful feedback mode, which extends feedback mode 3 to address error propagation with more intra updates. We also discuss its drawbacks and justify why it is hardly used. In [61] and [67] techniques have been proposed that combine the error-resilient mode selection with available decoder state information in the encoder. In this case the set of encoding options is not altered, that is, $\hat{\mathcal{O}} = \mathcal{O}$, but only the computation of the distortion is altered. Only for all data units with outstanding acknowledgment at the encoder, that is, $\mathcal{C}_{\text{enc},i} = \text{OAK}$, is the randomness of the observed channel state considered; for all other data units the observed channel state is no longer random. The expected distortion in this case is computed as

$$\hat{d}_{b,m} = \begin{cases} \mathbb{E}_{\{\hat{\mathcal{C}}_i\}}\{\tilde{d}_{b,m}(\hat{\mathcal{C}}_i)\} & \text{if } \mathcal{C}_{\text{enc},i} = \text{OAK}, \\ \mathbb{E}_{\{\mathcal{C}_i\}}\{\tilde{d}_{b,m}(\mathcal{C}_i)\} & \text{if } \mathcal{C}_{\text{enc},i} \neq \text{OAK}. \end{cases} \tag{2.10}$$

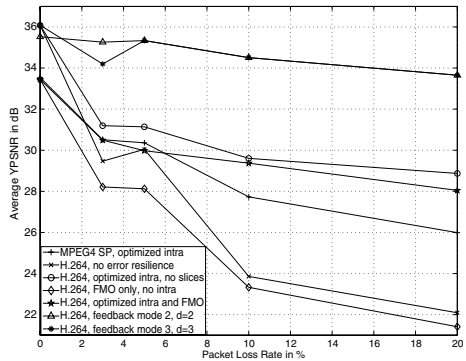
Compared to feedback modes 1 and 2, this method is especially beneficial if the feedback is significantly delayed. Compared to feedback mode 3, it reduces the unsatisfying performance in case of error propagation. Note that for $\delta \to \infty$ this mode turns into the mode selection without any feedback at all, and for $\delta = 0$ this mode is identical to feedback mode 2 and feedback mode 3. However, whenever the encoder gets information on the state of a certain data unit at the decoder, the statistics in the encoder have to be recomputed. Thus, the computational, storage, and implementation complexities are significantly increased [67].

2.5.5 Selected Performance Results for Internet Test Conditions

To verify the conclusions of the previous subsection at least partly for other error rates, bit rates, and test sequences we have evaluated selected error-resilience



(a)



(b)

FIGURE 2.23: Average PSNR ($\overline{\text{PSNR}}$) over packet error rate for *foreman*, QCIF, with frame rate $f_s = 7.5$ fps and *paris*, CIF. MPEG-4 with optimized random intra updates is compared to H.264 in various configurations. (a) *foreman*, 64 kbit/s, (b) *paris*, 384 kbit/s.

tools as presented previously. Test cases as suggested in [58] have been used. That is, we evaluate performance on four IP packet loss traces with 3, 5, 10, and 20% average loss rates, respectively. Note that the 5% trace is especially bursty. Also, the Common Intermediate Format (CIF) test sequence *paris* encoded at frame rate $f_s = 15$ frames per second (fps) is evaluated. Figure 2.23 shows $\overline{\text{PSNR}}$ as a function of packet loss rate[1] for *foreman* at 64 kbit/s and *paris* at 384 kbit/s.

[1]The labels on the abscissa specify the corresponding error pattern rather than random packet loss rates. Note that the 5% error file is burstier than the others, resulting in somewhat unexpected results.

Various error-resilience tools in H.264/AVC are compared to the MPEG-4 simple profile with an optimized ratio of random intra updates. The results are consistent for both sequences. The significance of the difference between different schemes is mainly explained by different sequence characteristics. It is observed that for error-free transmission abandoning any error-resilience tools obviously results in the best performance. The performance gains in terms of compression efficiency of H.264 over the MPEG-4 simple profile is also visible. If feedback mode 2 is used, in our case with feedback delay $\delta = 2$, we have to sacrifice some compression efficiency as the prediction signal is in general worse as it is further in the past. This does not apply for feedback mode 3. However, the performance in an error-free transmission environment is less relevant for our investigations. With increasing loss rates it is obvious that any kind of error-resilience feature in general improves the performance. Thereby, it is again recognized that reducing error propagation is much more important than packetization modes such as FMO, in our case with a checkerboard pattern and two packets per frame. Again, with the average PSNR as the measure of interest, the best performance without any feedback is obtained using channel-adapted rate–distortion-optimized mode selection according to (2.4) with each packet containing an entire source frame. Additional significant performance improvements can be achieved by the introduction of decoder feedback information. Thereby, for lower error rates feedback mode 3 outperforms feedback mode 2, but feedback mode 2 provides very consistent results over a large variety of error rates.

From these results, as well as subjective observations, it can be concluded that avoiding error propagation is basically the most important issue in error-prone video transmission. If no feedback is available, an increased percentage of intra MBs, selected by channel-adapted optimization schemes, performs best. Whenever feedback is available, it is suggested that interactive error control be applied. For short delays or low error rates, it is suggested to modify the prediction only in case of the reception of NACK message. In all other cases, it is suggested to reference only those areas for which the encoder is sure that the decoder has exactly the same reference area.

2.6 SUMMARY AND FURTHER READING

This chapter provides some background when transmitting MCP-coded standard-compliant video over error-prone channels. It is important to understand that video can benefit significantly if the transmitter can be sure that the video will be delivered reliably. Typically, the introduction of error-resilience tools in the video coding layer is very costly in terms of compression efficiency. The overhead is in general much better spent in lower layers of the protocol stack. Nevertheless there exist applications in which errors are inevitable. If the video encoder is not aware

of distortions on the transmission link, this in general leads to dramatic quality degradations due to instantaneous errors as well as spatial–temporal error propagation. Whereas the effect of instantaneous errors can be decreased by the use of specific packetization modes, the usually more severe effect of error propagation can be reduced by the application of more frequent intra information, interactive error control, or a combination of both. Preferably, for good overall performance, the selection of error-resilience tools is integrated in rate–distortion-optimized mode selection whereby the channel characteristics should be taken into account in this optimization. In general, standard-compliant decoders such as H.264/AVC can effectively operate even in harsh transmission environments if the encoder is appropriately designed for the transmission conditions and application constraints and the decoder includes some form of appropriate error concealment.

Additional literature on different subjects for error-resilient video transmission is plentiful; some work has already been discussed. In case of detailed interest in different subjects the reader is first of all encouraged to cover the remaining chapters of this book. Furthermore, magazines as well as journals have published special issues that deal exclusively with error-resilient video transmission, for example, [1,26,49,55,63], which provide a good starting point to dive into deep waters of error-resilient video transmission. Enjoy it!

REFERENCES

[1] J. Apostolopoulos. Reliable video communication over lossy packet networks using multiple state encoding and path diversity. In *Proceedings SPIE Visual Communications and Image Processing (VCIP)*, volume 4310, pages 392–409, San Jose(CA), USA, January 2001.

[2] E. Asbun and E. J. Delp. Real-time error concealment in compressed digital video streams. In *Proceedings Picture Coding Symposium*, Portland, Oregon, April 1999.

[3] G. Bjøntegaard. Definition of an error concealment model TCON. Doc. ITU-T/SG15/LBC-95-186, ITU-T, Boston, USA, June 1995.

[4] M.-J. Chen, L.-G. Chen, and R.-M. Weng. Error concealment of lost motion vectors with overlapped motion compensation. *IEEE Trans. on Circuits Syst. Video Technol.*, 7(3):560–563, June 1997.

[5] T. P.-C. Chen and T. Chen. Second-generation error concealment for video transport over error prone channels. In *Proceedings IEEE International Conference on Image Processing*, Rochester(NY), USA, September 2002.

[6] P. A. Chou and Z. Miao. Rate-distortion optimized streaming of packetized media. *IEEE Transactions on Multimedia*, 8(2):390–404, April 2006.

[7] G. Cote, S. Shirani, and F. Kossentini. Optimal mode selection and synchronization for robust video communications over error-prone networks. *IEEE Journal on Selected Areas in Communications*, 18(6):952–965, June 2000.

[8] Video adhoc group database for video codec evaluation. Technical Report S4-050789, 3GPP SA4 Video Adhoc Group, September 2005.

[9] A. Eleftheriadis, M. R. Civanlar, and O. Shapiro. Multipoint videoconferencing with scalable video. *Journal of Zhejiang University Science*, 7(5):696–706, May 2006.

[10] S. Fukunaga, T. Nakai, and H. Inoue. Error resilient video coding by dynamic replacing of reference pictures. In *Proceedings IEEE Globecom*, London, United Kingdom, November 1996.

[11] M. Ghanbari. Postprocessing of late cells for packet video. *IEEE Trans. on Circuits Syst. Video Technol.*, 6(6):669–678, December 1996.

[12] B. Girod and N. Färber. Feedback-based error control for mobile video transmission. *Proceeding of the IEEE*, 97:1707–1723, October 1999.

[13] J. Goshi, A. E. Mohr, R. E. Ladner, E. A. Riskin, and A. F. Lippman. Unequal loss protection for H.263 compressed video. *IEEE Trans. on Circuits Syst. Video Technol.*, 15(3):412–419, March 2005.

[14] B. G. Haskell, A. Puri, and A. N. Netravali. *Digital Video: An Introduction to MPEG-2*. Chapman & Hall, New York, USA, December 1996.

[15] P. Haskell and D. Messerschmitt. Resynchronization of motion-compensated video affected by ATM cell loss. In *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 3, pages 545–548, 1992.

[16] S. Hemami and T. Meng. Transform coded image reconstruction exploiting interblock correlation. *IEEE Transactions on Image Processing*, 4(7):1023–1027, July 1995.

[17] ISO/IEC JTC 1. *Coding of audio-visual objects, Part 2: Visual*. Apr. 1999; Amendment 1 (version 2), Feb., 2000; Amendment 4 (streaming profile), January 2001.

[18] ITU–T. *Video Coding for Low Bit Rate Communication*, version 1, November 1995; version 2, January 1998; version 3, November 2000.

[19] ITU–T and ISO/IEC JTC 1. *Advanced Video Coding for Generic Audiovisual Services*, 2003.

[20] B. Jung, B. Jeon, M.-D. Kim, B. Suh, and S.-I. Choi. Selective temporal error concealment algorithm for H.264/AVC. In *Proceedings IEEE International Conference on Image Processing*, Singapore, October 2004.

[21] M. Kalman, P. Ramanathan, and B. Girod. Rate–distortion optimized streaming with multiple deadlines. In *Proceedings IEEE International Conference on Image Processing*, Barcelona, Spain, September 2003.

[22] M. Karczewicz and R. Kurceren. The SP and SI frames design for H.264/AVC. *IEEE Trans. on Circuits Syst. Video Technol.*, 13(7), July 2003.

[23] C. W. Kim, D. W. Kang, and I. S. Kwang. High-complexity mode decision for error prone environment. Doc. JVT-C101, Joint Video Team (JVT), Fairfax(VA), USA, May 2002.

[24] A. H. Li, S. Kittitornkun, Y. H. Hu, D. S. Park, and J. D. Villasenor. Data partitioning and reversible variable length codes for robust video communications. In *Proceedings Data Compression Conference (DCC)*, Snowbird(UT), USA, March 2000.

[25] J. Liao and J. Villasenor. Adaptive intra update for video coding over noisy channels. In *Proceedings IEEE International Conference on Image Processing*, volume 3, pages 763–766, October 1996.

[26] A. Luthra, G. J. Sullivan, and T. Wiegand. Special issue on the H.264/AVC video coding standard. *IEEE Trans. on Circuits Syst. Video Technol.*, 13(7), July 2003.

[27] T. Nakai and Y. Tomita. Core experiments on feedback channel operation for H.263+. Doc. LBC96-308, ITU-T SG15, November 1996.

[28] A. Ortega and K. Ramchandran. Rate–distortion methods in image and video compression. *IEEE Signal Processing Mag.*, 15(6):23–50, November 1998.

[29] J. W. Park, J. W. Kim, and S. U. Lee. Dct coefficients recovery-based error concealment technique and its application to the MPEG-2 bit stream error. *IEEE Trans. on Circuits Syst. Video Technol.*, 7(6):845–854, December 1997.

[30] Y. O. Park, C.-S. Kim, and S.-U. Lee. Multi-hypothesis error concealment algorithm for H.26L video. In *Proceedings IEEE International Conference on Image Processing*, pages 465–468, Barcelona, September 2003.

[31] J. Postel. DoD standard transmission control protocol. Request for Comments (standard) 761, Internet Engineering Task Force (IETF), January 1980.

[32] J. Postel. User datagram protocol. Request for Comments (standard) 768, Internet Engineering Task Force (IETF), August 1980.

[33] P. Salama, N. Shroff, E. J. Coyle, and E. J. Delp. Error concealment techniques for encoded video streams. In *Proceedings IEEE International Conference on Image Processing*, volume 1, pages 9–12, Washington DC, USA, October 1995.

[34] P. Salama, N. Shroff, and E. J. Delp. A Bayesian approach to error concealment in encoded video streams. In *Proceedings IEEE International Conference on Image Processing*, pages 49–52, Lausanne, Switzerland, September 1996.

[35] P. Salama, N. Shroff, and E. J. Delp. A fast suboptimal approach to error concealment in encoded video streams. In *Proceedings IEEE International Conference on Image Processing*, pages 101–104, Santa Barbara(CA), USA, October 1997.

[36] P. Salama, N. B. Shroff, and E. J. Delp. Error concealment in encoded video. In *Image Recovery Techniques for Image Compression Applications*. Kluwer Academic Press, Dordrecht, The Netherlands, 1998.

[37] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. Request for Comments (standard) 3550, Internet Engineering Task Force (IETF), July 2003.

[38] C. E. Shannon. A Mathematical Theory of Communications. *Bell Systems Technology Journal*, pages 379–423, 623–656, 1948.

[39] S. Shirani, F. Kossentini, and R. Ward. A concealment method for video communications in an error-prone environment. *IEEE Journal on Selected Areas in Communications*, 18(6):1122–1128, June 2000.

[40] T. Socolofsky and C. Kale. A TCP/IP tutorial. Request for Comments (standard) 1180, Internet Engineering Task Force (IETF), January 1991.

[41] E. Steinbach, N. Färber, and B. Girod. Standard compatible extension of H.263 for robust video transmission in mobile environments. *IEEE Trans. on Circuits Syst. Video Technol.*, 7(6):872–881, December 1997.

[42] T. Stockhammer and M. Bystrom. H.264/AVC data partitioning for mobile video communication. In *Proceedings IEEE International Conference on Image Processing*, pages 545–548, Singapore, October 2004.

[43] T. Stockhammer, M. M. Hannuksela, and T. Wiegand. H.264/AVC in wireless environments. *IEEE Trans. on Circuits Syst. Video Technol.*, 13(7):657–673, July 2003.

[44] T. Stockhammer, T. Wiegand, and D. Kontopodis. Rate-distortion optimization for JVT/H.26L coding in packet loss environment. In *Proceedings International Packet Video Workshop*, Pittsburgh(PA), USA, April 2002.

[45] G. J. Sullivan and T. Wiegand. Video compression – from concepts to the H.264/AVC standard. *Proceeding of the IEEE*, 93(1):18–31, January 2005.

[46] G. J. Sullivan and T. Wiegand. Rate-distortion optimization for video compression. *IEEE Signal Processing Mag.*, 15(6):74–90, November 1998.

[47] H. Sun and W. Kwok. Concealment of damaged block transform coded images using projections onto convex sets. *IEEE Transactions on Image Processing*, 4:470–477, April 1995.

[48] H. Sun and J. Zedepski. Adaptive error concealment algorithm for MPEG compressed video. In *Proceedings SPIE Visual Communications and Image Processing (VCIP)*, pages 814–824, Boston(MA), USA, November 1992.

[49] R. Talluri. Error-resilient video coding in the ISO MPEG-4 standard. *IEEE Communications Magazine*, 36(6):112–119, June 1998.

[50] Y. Tomita, T. Kimura, and T. Ichikawa. Error resilient modified inter-frame coding system for limited reference picture memories. In *Proceedings Picture Coding Symposium*, Berlin, Germany, September 1997.

[51] D. S. Turaga and T. Chen. Model-based error concealment for wireless video. *IEEE Trans. on Circuits Syst. Video Technol.*, 12(6):483–495, June 2002.

[52] V. Varsa, M. M. Hannuksela, and Y.-K. Wang. Non-normative error concealment algorithms. Doc. VCEG-N62, ITU-T SG16/Q6 Video Coding Experts Group (VCEG), Santa Barbara(CA), USA, September 2001.

[53] W. Wada. Selective recovery of video packet losses using error concealment. *IEEE Journal on Selected Areas in Communications*, 7:807–814, June 1989.

[54] B. W. Wah, X. Su, and D. Lin. A survey of error concealment schemes for real-time audio and video transmissions over the internet. In *IEEE International Symposium on Multimedia Software Engineering*, pages 17–24, Taipei, Taiwan, December 2000.

[55] Y. Wang, S. Wenger, J. Wen, and A. G. Katsaggelos. Error resilient video coding techniques. *IEEE Signal Processing Mag.*, 17(4):61–82, July 2000.

[56] Y. Wang and Q. Zhu. Error control and concealment for video communication: A review. *Proceeding of the IEEE*, 86:974–997, May 1998.

[57] Y.-K. Wang, M. M. Hannuksela, V. Varsa, A. Hourunranta, and M. Gabbouj. The error concealment feature in the H.26L test model. In *Proceedings IEEE International Conference on Image Processing*, volume 2, pages 729–732, Rochester(NY), USA, September 2002.

[58] S. Wenger. Common conditions for wireline, low delay IP/UDP/RTP packet loss resilient testing. Doc. VCEG-N79, ITU-T SG16/Q6 Video Coding Experts Group (VCEG), Santa Barbara(CA), USA, September 2001.

[59] S. Wenger, M. M. Hannuksela, T. Stockhammer, M. Westerlund, and D. Singer. RTP payload format for H.264 video. Request for Comments (standard) 3984, Internet Engineering Task Force (IETF), February 2005.

[60] S. Wenger, G. Knorr, J. Ott, and F. Kossentini. Error resilience support in H.263+. *IEEE Trans. on Circuits Syst. Video Technol.*, 8:867–877, November 1998.

[61] T. Wiegand, N. Färber, K. Stuhlmller, and B. Girod. Error-resilient video transmission using long-term memory motion-compensated prediction. *IEEE Journal on Selected Areas in Communications*, 18(6):1050–1062, June 2000.

[62] T. Wiegand and B. Girod. Lagrangian multiplier selection in hybrid video coder control. In *Proceedings IEEE International Conference on Image Processing*, Thessaloniki, Greece, October 2001.

[63] T. Wiegand and B. Girod. *Multi-Frame Motion-Compensated Prediction for Video Transmission*. KL, KLADR, 2001.

[64] T. Wiegand, X. Zhang, and B. Girod. Long-term memory motion-compensated prediction. *IEEE Trans. on Circuits Syst. Video Technol.*, 9(1):70–84, February 1999.

[65] H. Yang and K. Rose. Source–channel prediction in error-resilient video coding. In *Proceedings IEEE ICME*, Baltimore(MD), USA, July 2003.

[66] W. Zeng and B. Liu. Geometric-structure-based error concealment with novel applications in block-based low-bit-rate coding. *IEEE Trans. on Circuits Syst. Video Technol.*, 9(4):648–665, June 1999.

[67] R. Zhang, S. L. Regunthan, and K. Rose. Video coding with optimal inter/intra-mode switching for packet loss resilience. *IEEE Journal on Selected Areas in Communications*, 18(6):966–976, June 2000.

[68] Q. F. Zhu and L. Kerofsky. Joint source coding, transport processing, and error concealment for H.323-based packet video. In *Proceedings SPIE Visual Communications and Image Processing (VCIP)*, volume 3653, pages 52–62, San Jose(CA), USA, January 1999.

[69] W. Zia and F. Shafait. Reduced complexity techniques for long-term memory motion compensated prediction in hybrid video coding. In *Proceedings Picture Coding Symposium*, Beijing, China, April 2006.

3

Error-Resilient Coding and Error Concealment Strategies for Audio Communication

Dinei Florêncio

3.1 INTRODUCTION

In this chapter we review the main techniques for error concealment in packet audio. As explained in Chapters 7–10, forward error correction (FEC) or repeat request solutions are often adequate for streaming media and broadcast. These can virtually eliminate information loss, guaranteeing that every bit is actually received at the decoder side. Nevertheless, these techniques will also require the introduction of additional delay, and the higher the protection level desired, the higher the delay required. Real-time communication (RTC) applications are very delay sensitive and will not be able to fully exploit these techniques to reduce 100% of the losses. For this reason, RTC needs are quite unique. We need error concealment, and we need FEC techniques that can be applied without excessive increase in delay. In this chapter we look at some of the techniques used in error concealment for speech and look at media-aware FEC techniques, with particular interest in RTC.

Compression and error concealment are tightly related. Compression tries to remove as much redundancy from the signal as possible, but the more redundancy is removed, the more important each piece of information is, and therefore the harder it is to conceal lost packets. More specifically, speech is a dynamic but slowly varying signal; the key way of compressing speech is by only transmitting signal *changes* in relation to the previous or expected state. Nevertheless, only transmitting these changes in a differential form means that if you lose some

information (e.g., due to a packet loss), the decoder does not know the current state of the signal any more. It is always expected that the segment corresponding to the missing data will not be properly decoded. But with differential coding, subsequent frames may also be affected. Furthermore, it is easier to replace any missing speech segments if one has received the correct signal in the vicinity of the missing segment. For all these reasons, error concealment may significantly depend on the compression technology used.

We will start this chapter by looking at some of the basic ideas behind packet loss concealment for speech. With that objective, in Section 3.2 we introduce the basic concealment techniques used in nonpredictive speech codecs. The job of concealing losses becomes harder as the codec removes more and more redundancy from the signal. In Section 3.3, we discuss some of the techniques used to reduce the impact of the feedback loop in CELP (Codebook Excited Linear Prediction) and other predictive codecs. In Section 3.4, we present some recent results in loss concealment for transform coders, which are used both in speech and in audio applications. Finally, in Section 3.5 we discuss recent research in media-aware FEC techniques. Particular attention is paid to speech, due to its importance in RTC, but many of the recent advances in loss concealment techniques we will discuss apply also to audio. For example, the same principles used in the overlapped transform concealment techniques can be used for most audio codecs, and the media-aware FEC can be applied to most audio or video coders. We also point out that this chapter is closely related to the ideas presented in Chapters 15 and 16.

3.2 LOSS CONCEALMENT FOR WAVEFORM SPEECH CODECS

When digital systems started replacing analog equipment a few decades ago, processing power was scarce and expensive, and coding techniques still primitive. For those reasons, most early digital systems used a very simple coding scheme: PCM (Pulse Code Modulation). In this digital representation of speech, there isn't really any coding in the compression sense. The signal is simply sampled and quantized. More specifically, the speech signal is typically sampled at 8 KHz, and each sample is encoded with 8-bit precision, using one of two quantization schemes, usually referred to as A-law and μ-law. This gives a total rate of 64 Kbps. The PCM system used in telephony has been standardized by the ITU (International Telecommunication Union) in the standard G.711 [1]. For Voice over Internet Protocol (VoIP) or other packet network applications, the speech samples will be grouped into frames (typically 10 ms in duration) and sent as packets across the network, one frame per packet. Note that a *frame* corresponds to a data unit in the terminology of Chapter 2. Note that, since there is no real coding, there is no dependence across packets: packets can be received and decoded independently. When G.711 was first adopted, the main motivation was

quality: A digital signal was not subject to degradation. At the same time, a 64-Kbps digital channel had a significant cost, and there was a strong push toward increased compression. With the evolution of speech compression technology, and increased processing power, more complex speech codecs were also standardized (e.g., [3–6]), providing better compression. Curiously, today, in many applications bandwidth is not necessarily a significant constraint any more, and we are starting to see basic PCM-coded speech increasing in usage again. Furthermore, many error concealment techniques operate in the time domain, and therefore are best understood as applying to PCM-coded speech. For this reason, in this section we review the basic concept of packet loss as applied to speech and look at some common techniques to conceal loss in PCM coded speech.

We assume speech samples are PCM coded and grouped in 10-ms frames before transmission. Since we assume packets are either received error free or not received at all, this implies that any loss incurred in the transmission process will imply a missing segment of 10 ms (or a multiple thereof). Figure 3.1 shows a segment of a speech signal. The signal is typical of a voiced phoneme. Figure 3.1(a) shows the original signal, whereas 3.1(b) shows a plot where 20 ms (i.e., two packets) is missing. As can be inferred from the picture, a good concealment algorithm would try to replace the missing segment by extending the prior signal with new periods of similar waveforms. This can be done with different levels of complexity, yielding also different levels of artifacts. We will now investigate a

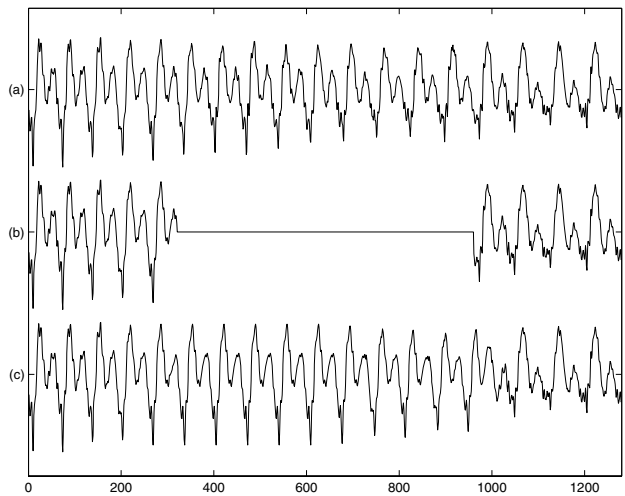


FIGURE 3.1: (a) A typical speech signal. (b) Original signal with two missing frames. (c) Concealed loss using Appendix I of G.711.

simple concealment technique, described in the Appendix I of Recommendation G.711 [2]. The results of applying that algorithm are illustrated in Figure 3.1(c).

3.2.1 A Simple Algorithm for Loss Concealment: G.711 Appendix I

The first modification needed in the G.711 decoder in order to allow for the error concealment is to introduce a 30 sample delay. This delay is used to smooth the transition between the end of the original (received) segment and the start of the synthesized segment. The second modification is that we maintain a circular buffer containing the last 390 samples (48.75 ms). The signal in this buffer is used to select a segment for replacing the lost frame(s).

When a loss is detected, the concealment algorithm starts by estimating the pitch period of the speech. This is done by finding the peak of the normalized cross-correlation between the most recent 20 ms of signal and the signal stored in the buffer. The peak is searched in the interval 40 to 120 samples, corresponding to a pitch of 200 to 66 Hz.

After the pitch period has been estimated, a segment corresponding to 1.25 periods is taken from the buffer and is used to conceal the missing segment. More specifically, the selected segment is overlap-added with the existing signal, with the overlap spanning 0.25 of the pitch period. Note that this overlap will start in the last few samples of the good frame (which is the reason we had to insert the 30 sample delay in the signal). The process is repeated until enough samples to fill the gap are produced. The transition between the synthesized signal and the first good frame is also smoothed by using an overlap-add with the first several samples of the received frame.

Special treatment is given to a number of situations. For example, if two or more consecutive frames are missing, the method uses a segment several pitch periods long as the replication method, instead of repeating several times the same pitch period. Also, after the first 10 ms, the signal is progressively attenuated, such that after 60 ms the synthesized signal is zero. This can be seen in Figure 3.1(c), where the amplitude of the synthesized signal starts to decrease slightly after 160 samples, even though the synthesized signal is still based on the same (preceding) data segment. Also, note that since the period of the missing segment is not identical to the synthesized segment, the transition to the new next frame may present a very atypical pitch period, which can be observed in Figure 3.1(c) around sample 1000.

The reader is directed to the ITU Recommendation [2] for more details of the algorithm. Results of the subjective tests performed with the algorithm, as well as some considerations about bandwidth expansion, can be found in [7]. Alternatively, the reader may refer to Chapter 16, which gives details of a related timescale modification procedure. For our purposes, it suffices to understand that the algorithm works by replicating pitch periods. Other important elements are

the gradual muting when the loss is too long and the overlap-add to smooth transitions. These elements will be present in most other concealment algorithms.

By the nature of the algorithm, it can be easily understood why it works well for single losses in the middle of voiced phonemes. As expected, the level of artifacts is higher for unvoiced phonemes and transitions. More elaborate concealment techniques will address each of these issues more carefully, further reducing the level of artifacts, at the cost of complexity. One possibility is to use an LPC filter and do the concealment in the "residual domain" [8,9]. Note that this is unrelated to the concealment of CELP codecs (which we will investigate in the next section). Here we simply use LPC to improve the extrapolation of the signal; the coefficients are actually computed at the decoder. In CELP codecs, we have to handle the problem of *lost* LPC coefficients.

3.3 LOSS CONCEALMENT FOR CELP SPEECH CODECS

In the previous section we looked at error concealment for PCM coded speech. In PCM coded speech, each speech frame is encoded independently (in fact, each *sample* is encoded independently). For this reason, the loss of one packet does not impair the decoding of subsequent frames. However, since no redundancy is removed from the signal, toll quality speech using G.711 requires 64 Kbps. Many other codecs will remove more redundancy from the signal, and therefore require a lower rate. More recent codecs are actually quite aggressive in removing redundancy. For example, several flavors of CELP coding have been used in speech codecs standardized by the ITU, including G.728 [3], G.729 [4], and G.722.2 [6]. Other organizations have also standardized several other CELP codecs, including the European Telecommunications Standards Institute (ETSI), which standardized several GSM (Global System for Mobile Communications) codecs [10] and the 3GPP (Third Generation Partnership Project) AMR (Adaptive Multi-Rate) codec [11], as well as the US Department of Defense (DoD), which standardized one of the first LPC codecs, the DoD FS-1016 [12], and more recently a 2.4-Kbps mixed excitation linear prediction (MELP) codec, the MIL-STD-3005 [14].

While a full understanding of a CELP codec is outside the scope of this chapter, we will need a basic understanding in order to deal with the concealment techniques used in association with these codecs. We will now present a quick summary of important elements of a CELP codec.

Figure 3.2 shows a block diagram of a typical CELP decoder. The first important element in these codecs is the use of a Linear Prediction (LP) filter, indicated as "LPC Synthesis Filter" in the figure. The second element is the use of a codebook as the input to the filter (thus the name "code excited linear prediction, CELP"). We are mostly concerned with the decoding operation so that we

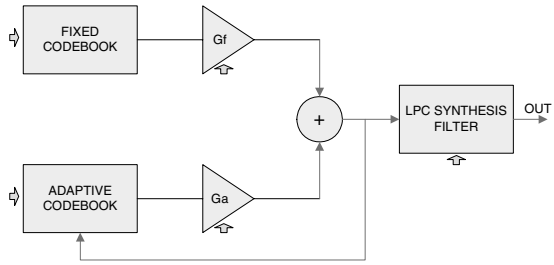


FIGURE 3.2: Block diagram of a basic CELP codec.

can verify what will happen when a frame is lost. In Figure 3.2, the wide arrows indicate the places where data or parameters are received. We see that the decoder will receive information relating to the LP filter (possibly including a long-term predictor, based on pitch) and on what part of the codebook to use as excitation. Specific CELP codecs will vary in how the codebook is populated, if the codebook is adaptive or not, and on how the filter coefficients are encoded and transmitted. Other differences, less relevant to our problem, relate to how the search on the codebook is performed, how filter coefficients are interpolated, and so on. More details about CELP codecs can be obtained from several sources, for example, from [13].

To understand the key elements of loss concealment for CELP codecs, we will now take a look at the loss concealment technique used in G.729. This ITU codec is a typical CELP codec and operates at 8 Kbps. It uses 10-ms frames and two codebooks: a fixed algebraic codebook and an adaptive codebook (based on the recent past excitation signal). The LPC filter is transmitted by first converting from LPC coefficients to Line Spectral Pairs (LSP), which are then differentially encoded by a vector quantization scheme. When a frame is lost, the decoder will take four specific actions to conceal the loss:

- Repeat the synthesis filter parameters. Since the differential information from the lost frame is not available, the same parameters of last received frame are used.
- Attenuate the adaptive and fixed codebook gains. The fixed codebook gain is reduced by 2% at each 5-ms subframe. The adaptive codebook gain is attenuated by 10% at each subframe and is also limited to 0.9. Note that reducing these gains will decrease the output energy, helping to hide artifacts produced by the concealment.
- Generate the replacement excitation. Since no excitation is received regarding the lost frame, a replacement excitation needs to be generated. The way the excitation is generated depends on the periodicity classification of the previous frame. If the previous frame was classified as periodic, the excita-

tion is generated by the adaptive codebook only, and the pitch delay is set to the same as the previous frame. If more then one frame is lost, each lost frame will increment the pitch by one. However, if the previous frame was classified as aperiodic, the excitation is taken only from the fixed codebook. The entry of the codebook to be used as excitation is based on a pseudo-random algorithm.

- Attenuate the memory of the gain predictor. Since the gains are transmitted on a recursive basis, by using a predictor, the exact state of the predictor is lost when a frame is missing. That will imply that even if the next frame is received without errors, the gains will not be correctly decoded. To help alleviate this problem, the value of the gain predictor is updated with an attenuated version of the codebook energy.

Note that the first three actions are related to generating the signal segment corresponding to the lost frame. The fourth item is related to reducing the artifacts produced in future frames, due to the mismatch in the internal state of the decoder. Rosenberg [15] analyzed the behavior of G.729 under losses and concluded that the artifacts produced by the internal state mismatch are actually more significant (subjectively) than the artifacts introduced by synthesizing the lost frame per se. This parallels the findings for video detailed in the previous chapter. He also concluded that the artifacts due to the mismatch last for approximately 70 to 100 ms.

Error concealment algorithms for CELP codecs are generally very codec specific. The error concealment used in G.729 is relatively simple, but it is a good example of how error concealment for CELP codecs work. Because of the importance of mitigating the effects of the internal state mismatch, more elaborate concealment techniques are highly associated with the particular codec they apply to. Furthermore, many modern CELP codecs are already designed with error concealment in mind and provide an associated algorithm that usually performs well. An example of a more elaborate concealment technique is the one used in the Wideband Adaptive Multirate codec (AMR-WB). This codec is standardized as the 3GPP recommendation TS 26.190 and as ITU G.722.2 [6]. The error concealment algorithm is described in standards ITU G.722.2 Annex I and in 3GPP TS 26.191. It follows the same basic principles of the technique described earlier, but it increases the performance at higher loss rates by having several different procedures for each one of six different states. The states are essentially a measure of how reliable the current state of the codec is. The reader is directed to the specification for more details of the concealment algorithm [6].

3.4 LOSS CONCEALMENT FOR LAPPED TRANSFORM CODECS

Linear transforms are widely used in signal compression. They have the primary objective of concentrating the signal energy on a few coefficients, thus preparing

the data for the subsequent quantization and entropy coding. Block transforms (e.g., the Discrete Cosine Transform, DCT) are convenient in that they make each block of data independent, constraining the effect of any error (either by quantization or by loss) to that single block of data. Nevertheless, by not exploiting correlation between adjacent samples in different blocks, they may often produce a structured noise (blocking artifacts), which is readily identifiable in the decoded signal as a buzzing sound. Overlapped transform coders occupy an important niche between block codes and fully predictive coders. They still limit the data to a certain block of samples, but their basis functions do not have discontinuities at block boundaries. Instead, basis functions spread over to (i.e., overlap) neighboring data blocks. This significantly reduces blocking artifacts, while preserving or even improving the compression qualities of the transform. For these reasons, overlapped transforms are used in numerous audio and speech codecs (e.g., MP3, Windows Media Audio [WMA], and ITU-G722.1).

A loss concealment technique based on exploiting the partial information available about certain samples has been recently introduced [16]. The technique can be used with essentially any linear transform where some of the coefficients are missing. Important cases include missing "frames" of overlapped transform (e.g., Modulated Lapped Transform, MLT) coefficients, or wavelet coefficients, or even single or multiple missing transform coefficients within a block of a block transform (e.g., DCT). However, since we are mostly interested in concealment of missing blocks in real-time speech and audio communication over packet networks, we will focus our discussion on the case of overlapped transforms.

When using an overlapped transform based codec, if a frame or block of coefficients is lost, partial information is available about the missing segment. While this information is not of enough quality to be used directly, it provides important clues about the missing segment. In this section we discuss ways in which to exploit this partial information to maximize the quality of the recovered signal. In particular, we apply some of the techniques to single-frame loss concealment on the ITU-G722.1 codec [5].

In order to better understand the scenario, let us take a look at how an overlapped transform is used for coding purposes. Figure 3.3a shows a one-dimensional signal. In this example, the signal is split into overlapping blocks of $2N$ samples, as shown in Figure 3.3b. Then, at each block, N transform coefficients are obtained by multiply/accumulate operations with the N basis functions constituting the transform. Figure 3.4 shows the first few basis functions of a typical transform. On the decoder side, the basis functions are scaled by the transform coefficients and added. Subsequent frames of the signal are then overlapped and added. Figure 3.3c shows the contribution of each overlapping block, before addition. Note that the recovered segments have the same length but are not identical to the original segments: the original signal is recovered only after adding the overlapping parts. Now, suppose the information about one of the blocks was

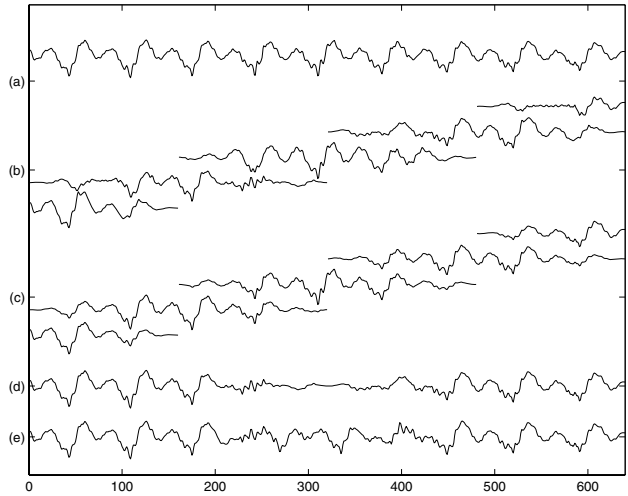


FIGURE 3.3: A sample speech signal. (a) Original signal. (b) Signal split into overlapping segments and windowed. (c) Corresponding segments after decoding. (d) Overlapped/added signal with one missing block. (e) Error concealment using simple block repetition.

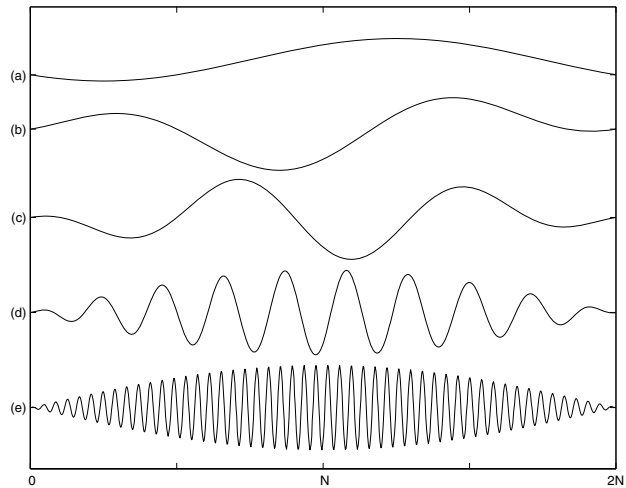


FIGURE 3.4: A few basis functions of the MLT transform. From top to bottom: 1st, 2nd, 3rd, 10th, and 50th basis functions.

lost. A total of $2N$ samples—spawning the lost block—cannot be reconstructed correctly. If we replace the lost coefficients with zeros, we would have the reconstructed signal indicated in Figure 3.3d. Note that in this example, although only N coefficients are missing, a total of $2N$ samples do not reconstruct correctly, due to the overlapping nature of the transform. Nevertheless, overlapped transforms like the MLT are critically sampled. This means that some partial information is available about the $2N$ incomplete samples. More specifically, a total of N linear equations are available regarding these $2N$ samples. We will now examine how this can be used to improve the loss concealment.

3.4.1 Speech Codecs

The ITU standard recommends that lost blocks be replaced with the previous block. While this technique is reasonable for low loss rates, artifacts are still present and become significant at loss rates that are common in the Internet. In particular, replication of coefficients does not take into account the alignment of pitch periods between past and lost frames. (See examples of speech codecs, G.722.1.)

In Section 3.2.1 we presented one of the main principles behind loss concealment for speech: pitch replication. As we will see, the algorithm presented in [16] can be seen as an elaborate pitch replication system. It uses the partial information available to synthesize a signal that has similar spectral characteristics and aligns well with the surrounding blocks.

The MLT transform can be decomposed into a windowing operation, followed by a folding and a DCT. Each block of coefficients can thus be written in matrix form as

$$m = dct(FJx), \tag{3.1}$$

where m is the $N \times 1$ vector of the resulting transform coefficients, F is the $N \times 2N$ fold-over matrix

$$F = \begin{bmatrix}
0 & 0 & & 1 & 1 & & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
\vdots & & & & & \ddots & & \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots \\
0 & 1 & & 0 & 0 & & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
1 & 0 & \cdots & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 0 & \cdots & 0 & -1 \\
0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & 0 & 1 & \cdots & 0 & 0 & \cdots & -1 & 0 \\
\vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & \vdots & & \ddots & & & & \vdots & \vdots \\
0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & & 1 & -1 & & 0 & 0
\end{bmatrix} \tag{3.2}$$

and J is a scaling matrix, that is, an $N \times N$ diagonal matrix with the windowing coefficients

$$J_{ij} = \begin{cases} \sin\big(\pi/2N(i+0.5)\big), & \text{if } i = j, \\ 0, & \text{otherwise.} \end{cases} \tag{3.3}$$

Furthermore, we will often need to refer to the signal before the DCT is applied. Let's call that z. So, we write

$$z = FJx. \tag{3.4}$$

Note that in (3.2) the nonzero elements of the folding matrix form two nonoverlapping subblocks. In other words, we can decompose F in four submatrices, where two of them are zero matrices:

$$F = \begin{bmatrix} F_1 & \mathbf{0} \\ \mathbf{0} & F_2 \end{bmatrix}. \tag{3.5}$$

Similarly, we write

$$J = \begin{bmatrix} J_1 & \mathbf{0} \\ \mathbf{0} & J_2 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}, \quad \text{and} \quad y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}. \tag{3.6}$$

Looking at the block diagonal structure of F and J, we can easily see that only the first half of the samples of x is used in computing the first half of the folded vector FJx (and similarly for the second half). That is, we can write

$$z_1 = F_1 J_1 x_1. \tag{3.7}$$

Therefore, if the next block of coefficients (which would also be using the second half of samples of x) is lost, we can use this partial knowledge about the samples to try to estimate x_2.

More specifically, suppose an isolated block is lost (i.e., both the preceding and the subsequent blocks to the missing block of coefficients are correctly received). The missing (incomplete) set of samples is $2N$ long. By computing the inverse DCT of the received data (but before applying the unfolding matrix), we have access to y. We can therefore write the following equation, applying to the first incomplete N samples:

$$z_2 = F_2 J_2 x_2. \tag{3.8}$$

Note that the x_1 and x_2 in (3.7) and (3.8) refer to different blocks. To avoid confusion, we will now add a time index to our notation. Namely to represent blocks at different time instants we will add a superscript index, indicating the block ordering. For example, x^n will mean the vector x and time instant n.

Assume the block at time n is missing, but both the previous and the subsequent blocks are correctly received. So, since block n is missing, but we have received blocks $n-1$ and $n+1$, we can write

$$\begin{bmatrix} z_2^{n-1} \\ z_1^{n+1} \end{bmatrix} = \begin{bmatrix} F_2 & \mathbf{0} \\ \mathbf{0} & F_1 \end{bmatrix} \begin{bmatrix} J_2 & \mathbf{0} \\ \mathbf{0} & J_1 \end{bmatrix} \begin{bmatrix} x_1^n \\ x_2^n \end{bmatrix}. \tag{3.9}$$

Note that the matrices containing F_1, F_2, J_1, and J_2 are now rotated in relation to the original F and J matrices. For simplicity, let's refer to these modified (block rotated) matrices in the aforementioned equation as G and H. We therefore write

$$\begin{bmatrix} z_2^{n-1} \\ z_2^{n+1} \end{bmatrix} = GHx^n. \tag{3.10}$$

Note that this is an underdetermined system of equations. We know z_2^{n-1} and z_2^{n+1}, and we are trying to estimate the $2N$ samples of x^n. This underdetermined system could be solved for the minimum energy vector x^n using the Moore–Penrose generalized inverse of GH. This would provide the minimum energy signal segment x that satisfies the received (partial) information. Nevertheless, simulations show that this is not a good choice for x, as the nature of the matrix J tends to concentrate the energy in the higher gain samples. A better choice is to find the solution minimizing the energy of the windowed signal Hx. This solution does distribute more evenly the energy across the samples of x. Nevertheless, it still does not use the information about the neighboring frames. Before proceeding to describe the best mode, let us introduce a small change in interpretation. Let us introduce an identity matrix I in (3.10), which becomes

$$\begin{bmatrix} z_2^{n-1} \\ z_2^{n+1} \end{bmatrix} = GHIx^n. \tag{3.11}$$

We now interpret I not as a simple identity matrix, but as a matrix whose columns form a basis for the space of x. In this context, the basis I consists simply of impulses at each sample location. Using the generalized inverse of GH would be minimizing the energy of the basis representation over these impulses. That takes into account the partial information about the missing samples, but it does not take into account all the prior information we have about the missing segment: the properly received signal segments just before (and possibly after) the missing segment. To fully exploit that information, we will reshape the aforementioned equation by introducing two small modifications. The first modification improves the signal continuity across frames by removing the no-excitation response. The

second biases the reconstructed signal toward having the same spectrum and pitch as the neighboring segments.

To account for the signal continuity, we estimate the LPC filter corresponding to the previous block and compute the no-excitation response of the LPC filter into the missing segment, $\check{x}$. We then modify (3.11) to account for $\check{x}$ and write

$$\begin{bmatrix} z_2^{n-1} \\ z_2^{n+1} \end{bmatrix} - GH\check{x} = GHI\hat{x}^n, \tag{3.12}$$

where $\hat{x} = x - \check{x}$.

To account for the spectral continuity, we invoke our interpretation of I as a basis for the vector x (now $\hat{x}$) to claim we should not be minimizing the energy of x. Instead, we should be minimizing the energy of the representation of x under a basis whose functions have a spectrum corresponding to the desired LPC spectrum. To that end, we apply the LPC filter to the identity matrix, to obtain a new basis L, where each column of L corresponds to a time-shifted version of the impulse response of the LPC filter.

Finally, we compute an estimate of the periodicity and pitch period for the segment and apply that to the basis functions as well. Each column of L is now a series of "colored" pulses, each apart by the pitch period, each with the impulse response of the LPC filter, and each with decreasing amplitude, based on the estimated periodicity index. For simplicity, we still call this final basis matrix L. The representation on this new basis is not x any more, so let's call it r. We now have

$$\begin{bmatrix} z_2^{n-1} \\ z_2^{n+1} \end{bmatrix} - GH\check{x} = GHLr^n, \tag{3.13}$$

which is then solved by the pseudo inverse of GHL, that is,

$$r^n = (GHL)^{\dagger} \left(\begin{bmatrix} z_2^{n-1} \\ z_2^{n+1} \end{bmatrix} - GH\check{x} \right), \tag{3.14}$$

where $\dagger$ denotes the pseudo inverse. Note that this is the solution that minimizes the LPC residual of x, as we wanted. The final solution for x is obtained by simply computing

$$x^n = Lr^n + \check{x}. \tag{3.15}$$

Figure 3.5 shows a sample of the results obtained by the concealment algorithm. The first signal is the original, the second is the signal reconstructed using the proposed technique, and the third is the results of concealment by a pitch replication method. In both cases every third packet is lost.

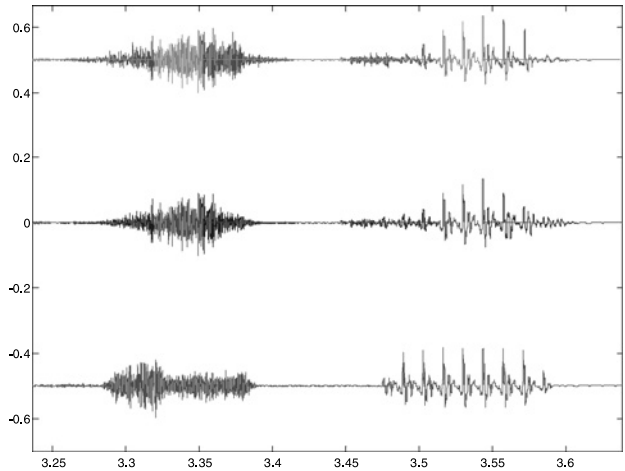


FIGURE 3.5: Sample results. (a) Original signal. (b) Concealed using the partial information method, after losing every third frame. (c) Concealed using the pitch replication method.

In this section, we presented an error concealment technique that exploits the partial information available for the missing segment of a signal encoded by an overlapped transform. The discussion was centered around a speech codec, simply because speech is of foremost importance for real-time communication. Nevertheless, the same principle can be applied to other overlapped transform codecs. In particular, the same ideas apply to error concealment in music, as long as we remove the conditions relating to pitch and introduce a higher order model to account for the harmonic nature of music.

3.5 FORWARD ERROR CORRECTION TECHNIQUES FOR SPEECH

In the previous sections, we discussed several error concealment techniques, targeted at alleviating the consequences of packet losses. Some of these techniques are reasonably effective and will provide quite adequate speech quality, especially at low loss rates. Nevertheless, as the loss rates increase, concealment becomes increasingly hard and is prone to leave a number of artifacts. For this reason, Forward Error Correction (FEC) is often used—either in isolation or as a complementary measure—against packet losses. FEC techniques can range from simple packet replication techniques to more elaborate schemes, including media-dependent FEC. In this section, we discuss media-dependent FEC and present a framework for optimum rate distortion bit allocation. We will also present a case

study based on the AMR-WB codec [6]. More general FEC methods can be found in Chapters 7 and 9.

3.5.1 Delay and FEC

Generally speaking, FEC schemes allow the receiver to correctly decode a message, even if some of the packets are lost. This is done by adding redundant information to the stream. The information can be included in a separate packet, or appended to existing packets. For example, one could send a parity packet after every three data packets, as illustrated in Figure 3.6. In this scheme, if one of the three packets is lost, one can use the parity packet to recover the original information without loss. This increase in robustness is useful, but it also increases the bandwidth requirement by 33% (by sending one extra packet for every three original packets). Furthermore, there is also a delay cost: if the first of the three packets is lost, the receiver has to wait until receiving the parity packet before decoding the lost packet. In this example, this would add an extra two-frame delay. Partially to reduce this added delay, most FEC schemes for real-time communication simply repeat the packet. More information about standard FEC techniques will be discussed in Chapters 7 and 9. But for now, let's simply mention that using an FEC code that spreads over N blocks will essentially add up to N blocks delay. For this reason it is highly desirable for FEC codes to spread the smallest number of blocks possible.

3.5.2 Media-Dependent FEC

As we mentioned, it is desirable that the FEC technique introduces as little extra delay as possible. Ideally, we would like FEC codes that spread only a single block. Unfortunately, under the traditional FEC techniques, the only such "code" available is packet repetition. That happens because traditional FEC try to protect the *bits* of the message. When one is sending media, protecting individual bits is

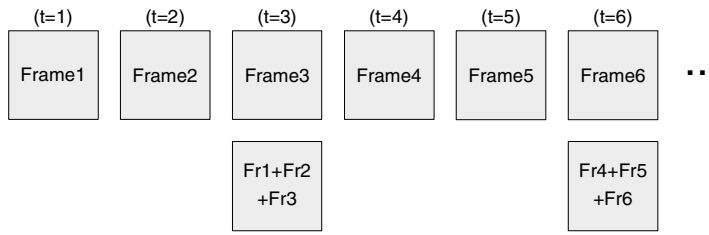


FIGURE 3.6: FEC example with a 4:3 redundancy. Each fourth block is an XOR of the previous three blocks.

not as important anymore, but instead, the idea is to protect the *signal*. In other words, a rate–distortion trade-off can now be applied. Looked at from this point of view, packet repetition is clearly suboptimal. For example, in a 10% loss scenario, the error correction information is only used 10% of the time and yet uses the same rate as the primary packet.

In traditional FEC codes, the sender inserts bit redundancy in the transmitted packets, and the receiver will either perfectly receive the frame or receive nothing. There is no rate–distortion trade-off. In media-dependent FEC methods, in contrast, the transmitter sends multiple descriptions of the same frame so that in case of packet loss, another packet containing the same data, albeit different quality, can be used to recover the loss. Hence, each packet will carry an appropriate representation of the current frame, along with a coarse representation of one or more previous frames. Clearly, there is a trade-off between attributing rate to redundant information instead of to the current frame. By increasing the amount of redundant information, we increase the probability and the quality of loss recovery while sacrificing from the quality of the most recent frame. An example of such media-dependent FEC schemes is the one presented in [17]. Earlier work includes the Robust Audio Tool [18], which limits the repeat packet to be the same as the original one. The problem can be formulated as follows. Given a model for the channel and a total transmission rate R (i.e., fixed packet size), what is the optimum partition of the bit budget between redundant and current frames such that a distortion measure D_T is minimized? We consider each frame as a signal segment and each packet may contain information units regarding one or more frames. The units can contain raw data or a representation of data derived by some compression algorithm (e.g., LPC coefficients, prediction errors). We model each packet as a collection of multiple units corresponding to different segments of the signal, each possibly having a different rate. For each packet, r_1 is the rate of the present segment and r_i is the rate of $(i-1)$th past segment. The number of these units and the rate of each unit can be either fixed by the optimization algorithm prior to transmission or adaptively changed based on the input signal. Figure 3.7 shows an example, with four consecutive packets, with each packet carrying information about the current frame, as well as lower fidelity information about the two previous packets.

Another point of interest is whether each unit is dependent on previous units (i.e., differential coding). We will analyze here the case in which each segment of data is processed independently. This would be the case, for example, of encoding video with all I-frames or encoding speech using G.722.1 (“Siren”) or G.711 (PCM). The case of history-dependent algorithms, where each segment is sent as a unit, is handled in detail in [19].

We now analyze the optimization problem where each frame is encoded independently of neighboring frames. The optimal rate of each packet is chosen to minimize the average distortion given the loss model. We start our discussion with

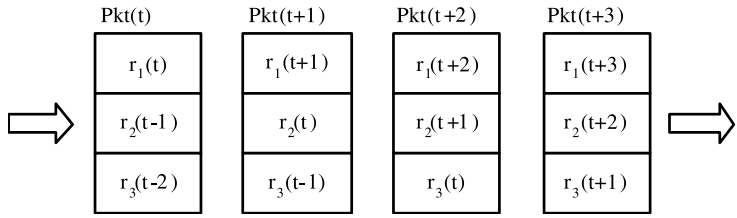


FIGURE 3.7: Media-aware FEC example with a factor of 3 redundancy. The current block carries the frame with full resolution and previous blocks with decreasing degree of accuracy.

the case where there is only a single–rate distortion function to be used, the bit rate allocation is fixed (i.e., independent of the actual signal), the loss model is i.i.d., and delay is ignored. This can then be extended to more complex scenarios.

Assume no inter-frame coding and a fixed rate–distortion function $D(r)$. The distortion $D(r)$ is the average distortion due to using rate r for a generic compression algorithm using only data in the current frame. As an example, suppose there are three units in each packet, as in Figure 3.7, and the packet loss is an i.i.d. Bernoulli process with loss probability p. Since the loss event is i.i.d., without loss of generality, we can restrict our decoder to use the first packet received, even though we may receive multiple units for the same segment. It follows automatically that the optimum solution requires $r_1 \geq r_2 \geq r_3$. Since the probability of a packet being received is $1 - p$, and since if we receive a packet we will use the r_1 contained in that packet for reconstruction of the corresponding frame, the distortion for this frame will be $D(r_1)$ with probability $(1 - p)$. However, there is a probability p that this packet is not received. In that case, we will wait for the next packet, which contains the same frame, but coded at rate r_2. That packet has itself probability $(1 - p)$ of being received. Therefore, the probability that we use the data contained in that packet is going to be $p(1 - p)$, and in that case the distortion is going to be $D(r_2)$. We can proceed similarly with the third packet to conclude that the distortion contributed by that packet is $p^2(1 - p)D(r_3)$. Finally, if none of the three packets containing information about this segment is received, we will use some other loss concealment technique, which we assume will itself induce a distortion K. The same computation will hold for any particular segment (frame) of the signal. Therefore, the expected distortion at any time is given by

$$D_T = (1 - p)D(r_1) + p(1 - p)D(r_2) + p^2(1 - p)D(r_3) + p^3K. \qquad (3.16)$$

The distortion K directly depends on the loss concealment strategy, and we assume it to be comparable to $D(0)$. If we do not include any delay considerations,

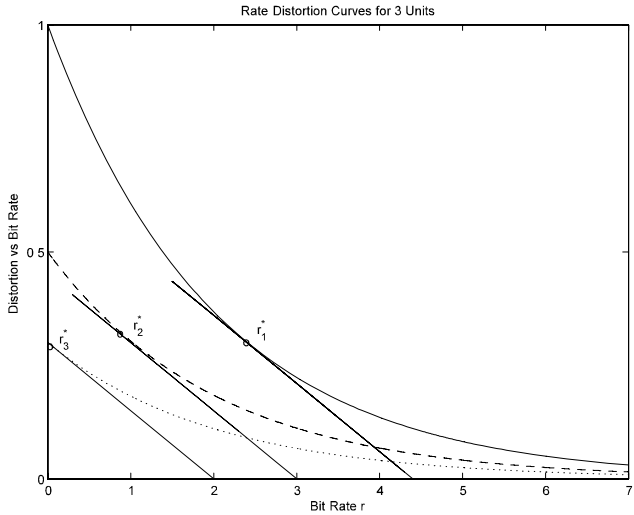


FIGURE 3.8: Optimization procedure. Each rate is used. The optimum solution is the one that implies the same derivative on each of the (scaled) curves.

the optimization problem can be formulated as

$$\min_{r_1,r_2,\ldots,r_N,N} D_T(r_1,\ldots,r_N), \quad \text{s.t.} \quad \sum_{i=1}^{N} r_i < R, \tag{3.17}$$

where N is the total number of units to be used and R is the total rate. Since we assume no inter-frame coding, the R–D curves are the same for the first unit and for subsequent (FEC) units. This can be seen in Figure 3.8, which illustrates the contribution of each of the terms in (3.16). Note in the figure that the curve corresponding to each unit has the same shape, but has been appropriately scaled by the associated probability, as prescribed by (3.16). In this example, the total distortion D_T is the sum of the three different rate distortion curves, where each curve is simply the product of $D(r)$ and the respective probability coefficient coming from the channel model. Hence, given N, the problem (for convex rate–distortion functions) is formulated as an unconstrained optimization using Lagrange multipliers,

$$\min_{r_1,r_2,\ldots,r_N,N} D_T(r_1,\ldots,r_N) + \lambda \sum_{i=1}^{N} r_i, \tag{3.18}$$

where λ is the Lagrange multiplier. The optimal configuration is reached when

$$\frac{\partial D_T}{\partial r_1} = \frac{\partial D_T}{\partial r_2} = \frac{\partial D_T}{\partial r_3}. \tag{3.19}$$

Since we assume the encoding of each unit is independent (no inter-frame coding), the partial derivatives are simplified to

$$\left.\frac{\partial D_T}{\partial r}\right|_{r1} = \left.\frac{\partial D_T}{\partial r}\right|_{r2} = \left.\frac{\partial D_T}{\partial r}\right|_{r3}. \tag{3.20}$$

In other words, the problem is now reduced to finding the optimum rate points $r_1^*, \ldots, r_n^*$ such that the slopes of the scaled–rate distortion curves are the same at each r_i^* and $\sum_{i=1}^{N} r_i^* \leq R$. This is illustrated in Figure 3.8.

Note that whenever N is not given a priori, it must be included as a parameter in the optimization. In principle, the induced delay is N, because to present the frames at a constant rate, the receiver has to wait for the N packets before decoding a frame. (However, we will see in Chapter 16 that adaptive playout can be used to keep the average delay below N.) Since (3.16) does not include any penalty for latency, the optimization in (3.20) will artificially favor a large N. Nevertheless, note that even if there is no penalty for latency, there is always a finite value of N such that the algorithm stops and favors the quality instead of error recovery. If we define ordered curves in the figure as $D_1, \ldots, D_i$ (e.g., for Figure 3.8, $i = 3$), then the number of units that would be included will be upper bounded by

$$N^* = \arg\max_N \sum_{i=1}^{N} \hat{D}_i^{-1}\left(\hat{D}_N(r=0)\right) \leq R, \tag{3.21}$$

where $\hat{D}_i(r)$ is the derivative of the function $D_i(r)$ and $\hat{D}_i^{-1}(r)$is the inverse of $\hat{D}_i(r)$. After getting an upper bound, N can be computed by decreasing N and recomputing the distortion until D_T starts to increase. Since N is generally small, this exhaustive search in N is usually not a problem.

This procedure will determine the optimum rate allocation to each packet and to each error correcting packet. Each subsequent (error correcting) packet will always be at the same or lower rate as the previous packet. If a packet is lost, but a subsequent packet containing the error correcting information is received, the decoder will replace the lost information with that contained in the correction packet. In other words, this can be viewed as a forward error correction technique where the objective is to recover the signal, not the bits. In the same way the original (source) encoding may have introduced signal distortions in order to optimize the channel utilization, the channel coding may also introduce its own distortion,

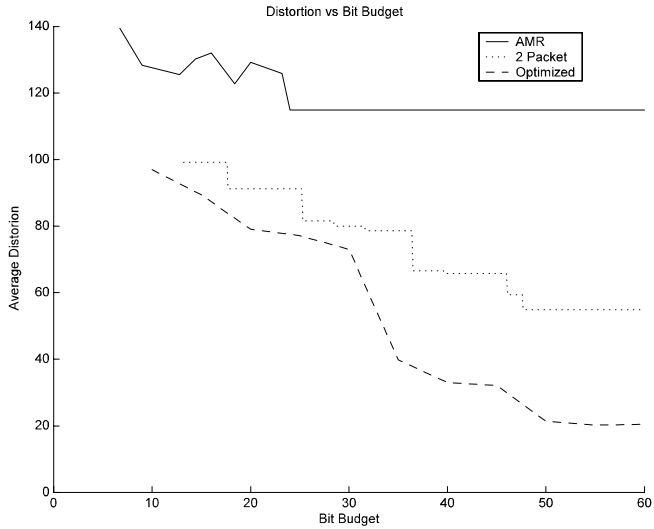


FIGURE 3.9: Subjectively weighted SNR after decoding for several error rates. Top curve is for original (single packet) AMR, dotted line for repeat only, and lower (dashed line) results are for media-aware FEC, which typically selects a lower rate for correction packets.

also to optimize the channel utilization. Furthermore, the aforementioned method guarantees that both the channel and source coding are operating at optimal condition, therefore the name "combined source-channel coding."

The optimization procedure presented in this section assumed an intra-only coder, a fixed rate–distortion function, and an i.i.d. loss model. Each and all of these constraints can be removed under certain assumptions. Details can be found in [19], which also includes an example of applying this technology to a AMR-WB codec. Figure 3.9 shows a plot of final quality as a function of error rate when applying this technology and compares that to a repeat-only FEC strategy.

3.6 OTHER ERROR-RESILIENT CODING TECHNIQUES

In previous sections we looked at two main ways to alleviate the consequences of packet loss: concealment and FEC. In the first technique, we try to synthesize the missing information based on surrounding (i.e., received) blocks. The second technique sends some additional redundant information (FEC), which helps recover the missing information, either in its natural form or with reduced fidelity. A few other techniques fall somewhere between these two, in the sense that in-

stead of adding redundancy, they will leave some redundancy in the signal at coding time. Ideally this is done in a well-planned way, leaving only the redundancy that will be most effective in recovering the lost packets. This is in contrast to some older techniques (e.g., G.711), where redundancy was left in the signals mostly to simplify computation. An example of such an error-resilient technique can be seen in the Siren codec (G.722.1), which intentionally does not use differential coding, to increase robustness to noise. A few other techniques used to improve error resilience include Multiple Description Coding (Chapter 17) and unequal error protection (Chapter 9), which can be used with standard codecs, but are particularly useful when used with scalable codecs (Chapter 6).

3.7 SUMMARY AND FURTHER READING

In this chapter we have looked at Error Concealment Strategies and Error Resilient Coding for Audio Communication. We looked at some of the basic techniques that are used in concealing packet losses, applied to several kinds of codecs, including frame-independent codecs, overlapped transform codecs, and fully predictive codecs. We looked at some of the techniques incorporated into international standards, and looked at a few additional techniques. We saw that many codecs are available and can be used for specific application. The particular choice of a codec will generally involve system design issues, for example, computational complexity, bandwidth availability, backward compatibility, and so on. Furthermore, commercial considerations often play a major role as well. These include existing intellectual property right, licensing terms, availability of source code, and so on. For example, many of the codecs mentioned were designed for a specific application. As a general rule, CELP codecs tend to perform well in terms of rate/distortion for most rates above 2400 bps, as long as encoding clean speech. For example, mostly all codecs used in cellular phone systems are CELP based. However, when coding music or when background noise becomes more prevalent, waveform codecs start to present good performance. Indeed, while the ITU and the GSM have standardized several CELP codecs to use at different rates, in telecommunication systems some of the primary VoIP systems use waveform-based codecs. For example, Microsoft Messenger uses Siren/G711.1 as the default codec. This can be partially attributed to the fact that bandwidth constraints on VoIP are not as severe as in cellular systems and partially to the fact that use of a close-talking microphone is not expected in the desktop environment.

The main objective of this chapter was to look at the different techniques available. A number of subsequent chapters will look at related topics, in particular about aspects related to FEC, scalable audio coding, and adaptive playout. These are techniques that are particularly important for speech communication.

REFERENCES

[1] ITU-T Recommendation G.711, *Pulse code modulation (PCM) of voice frequencies*, November 1988.

[2] ITU-T Recommendation G.711, Appendix I, *A high quality low-complexity algorithm for packet loss concealment with G.711*, September 1999.

[3] ITU-T Recommendation G.728, *Coding of speech at 16 kbit/s using low-delay code excited linear prediction*, September 1992.

[4] ITU-T Recommendation G.728, *Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear prediction (CS-ACELP)*, March 1996.

[5] ITU-T Recommendation G.722.1, *Low-complexity coding at 24 and 32 kbit/s for hands-free operation in systems with low frame loss*, May 2005.

[6] ITU-T Recommendation G.722.2, *Wideband coding of speech at around 16 kbit/s using Adaptive Multi-Rate Wideband (AMR-WB)*, July 2003.

[7] R. V. Cox, D. Malah, and D. Kapilow, "Improving upon toll quality speech for VOIP," *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference*, vol. 1, pp. 405–409, November 2004.

[8] E. Gunduzhan and K. Momtahan, "Linear prediction based packet loss concealment algorithm for PCM coded speech," *IEEE Transactions on Speech and Audio Processing*, vol. 9, num. 8, pp. 778–785, November 2001.

[9] M. Elsabrouty, M. Bouchard, and T. Aboulnasr, "Receiver-based packet loss concealment for pulse code modulation (PCM G.711) coder," *Signal Processing*, vol. 84, pp. 663–667, 2004.

[10] K. Jarvinen *et al.*, "GSM enhanced full rate speech codec," *Proc. of ICASSP*, vol. 2, pp. 771–774, April 1997.

[11] 3GPP Recommendation TS 26.071, *AMR speech Codec; General description*, ver 6.0.0, December 2004.

[12] T. E. Tremain, "The Government Standard Linear Predictive Coding Algorithm: LPC-10," *Speech Technology Magazine*, pp. 40–49, April 1982.

[13] X. Huang, A. Acero, and H. Hon, *Spoken language processing: A guide to theory, algorithms and system development*," Prentice Hall, 2001.

[14] "MELP vocoder algorithm: The new 2400 bps federal standard speech coder," Atlanta Signal Processors, Inc., available (as of August 2006) at http://www.aspi.com/tech/specs/pdfs/melp.pdf.

[15] J. Rosenberg, "Distributed Algorithms and Protocols for Scalable Internet Telephony," *Ph.D. thesis*, Columbia University, 2001.

[16] D. Florencio, *Personal notes about Siren codec packet loss concealment,* Microsoft, 2004.

[17] V. Hardman *et al.*, "Reliable audio for use over the Internet," *Proc. INET*, 1995.

[18] J.-C. Bollot and A. Vega-Garcia, "The case for FEC-based error control for packet audio in the Internet," *ACM Multimedia Systems*, 1996.

[19] S. Kozat and D. Florencio, "Media dependent FEC," Internal Report, Microsoft, 2003.

4

Mechanisms for Adapting Compressed Multimedia to Varying Bandwidth Conditions

Antonio Ortega and Huisheng Wang

4.1 INTRODUCTION

Most currently deployed networks provide no quality of service (QoS) guarantees. Thus it is clearly necessary for bandwidth adaptation mechanisms to be available in order for real-time multimedia delivery applications to be successful. These mechanisms allow applications to adjust gracefully to changes in available channel bandwidth. Without these adaptation tools, changes in available bandwidth will lead to significant quality degradation, leading occasionally to total service interruption. This is an even more pressing need when one considers the increasing heterogeneity of both networks and network access devices.

Consider, for example, a hypothetical application where a traveling user is interested in accessing video captured by a wireless home surveillance camera. This user gains remote access to the video feed using a wireless network device, such as a cell phone. In order to provide a smooth, constant quality playback, such a system would have to be robust to bandwidth fluctuations due to multiple causes; for example, variations in traffic within the home network, load of the cable modem access network, distance of the user to the nearest base station, and load in the wireless access network. This is in addition to traffic fluctuations along the relevant paths within the Internet backbone.

Clearly, one possible way to tackle the problem would be to engineer application and network resources so as to avoid altogether these bandwidth fluctuations, or at least reduce their amplitude. Indeed, in all the most successful deploy-

ments of digital video systems (cable and satellite broadcasting) some form of bandwidth reservation is in place so that each video transmission receives a fixed bandwidth. These systems are in fact (i) closed, so that a single service provider controls all the communication links and thus how many video transmissions occur at any given time, and (ii) reliable, so that, except for rare outages, bandwidth levels available to each video transmission remain constant.

While these applications have been very successful commercially, current interest is driven by multimedia applications that operate over open, heterogeneous and potentially unreliable networks. Witness, for example, the current growth in voice over IP (VoIP) systems. Interest in deploying other such applications (from video conferencing to video on demand) over these networks is considerable, but the technical challenges are very significant.

This chapter is devoted to tools and techniques that allow applications that involve transmission of multimedia streams to cope with significant changes in transmission bandwidth. We call these *bandwidth adaptation* mechanisms. Note that there are other tools available to improve the quality of multimedia delivery over these challenging network environments. In particular, techniques for multimedia resilience and error concealment allow the decoder to either eliminate the effect of losses or excessive delays (by introducing redundancy) or to "mask" their effect on the decoded media (using error concealment). In many cases these techniques are applied to a design based on worst case assumptions, for example, the designer may determine what the maximum packet loss rate for the system is likely to be and then select appropriate error-resilience techniques to ensure sufficient quality even at those loss rates.

Instead, our main focus is on *dynamic* adaptation: various components of the application actively monitor and react to changes in network behavior by making adjustments to the data being sent to the receiver.

Our simple traveling user example illustrates the importance of bandwidth adaptation mechanisms; in their absence variations in available bandwidth may lead to packet losses in the video stream and thus to very abrupt variations in video quality, and possibly to complete interruption of service. For example, if the surveillance camera is set to encode video at a certain rate regardless of network conditions, then usable video quality may not be available unless the minimum bandwidth encountered at all links along the network route exceeds the bandwidth required by the application. This example also indicates that there are many possible mechanisms for adaptation, such as the fact that, the two end applications could communicate with each other, or intermediate nodes of the network could modify the video stream (e.g., via transcoding) to make it conform to the available bandwidth. One of the goals of this chapter will be to describe alternative adaptation mechanisms and illustrate their relative merits. Note that bandwidth adaptation cannot guarantee loss-free transmission, so mechanisms to minimize

the impact of losses and for error concealment may also be needed, as discussed in Chapters 2, 3, 8, 9, and 10.

All practical multimedia networking systems currently in use incorporate some form of bandwidth adaptation. For example, widely deployed commercial systems such as Windows Media Player and RealPlayer make use of adaptation technologies, namely Intelligent Streaming [6] and SureStream [19], respectively. In both systems, multiple redundant representations of the same content are created, with each version optimized for a specific transmission rate. During transmission, the streaming server dynamically adapts to the bandwidth changes by switching between these streams in such a way as to maximize the reconstructed video quality.

These systems, and alternative ones to be discussed in more detail in this chapter, operate by

(i) observing the characteristics of the network environment (e.g., bandwidth availability, packet loss rates) and
(ii) increasing/decreasing the rate of the multimedia stream so as to maximize quality available to the users.

The performance of these systems will depend on many factors, such as the type and accuracy of information that is available about the network state; where this information is acquired; and constraints on how the streaming rate can be adjusted.

Note that the multimedia codec being used is a very important factor in determining what forms of adaptation are feasible. Some media codecs are designed so as to facilitate bandwidth adaptation; for example, this is the case when scalable coding (described in Chapter 5) is used. With a scalable codec a media sequence is encoded into a number of layers, or as an embedded stream with fine granularity, such that the transmitter can easily select a subset of layers or bits to send based on the current channel condition.

This chapter provides a general overview of bandwidth adaptation mechanisms, focusing on **where** the adaptation occurs (e.g., at the sender end, or somewhere in the network), **who** decides how to adapt (e.g., sender- vs. receiver-driven adaptation is possible), **how** the adaptation is supported by various codecs (e.g., scalable vs. nonscalable systems), and the **trade-offs** in performing the adaptation with different options using a number of criteria (e.g., in terms of delay, latency, complexity, quality).

Note that our main goal is to provide an overview of various classes of techniques available for bandwidth adaptation and to summarize their relative merits. A more detailed and quantitative performance comparison falls outside of the scope of this chapter and, given the complexity of these systems, may indeed be difficult to develop. Note also that most of our discussions will be relevant to general streaming media systems, although often our specific examples (in partic-

ular our discussion of coding techniques) will focus on the case of video streaming.

4.1.1 A Simplified System—Definition of Major Components

To facilitate the discussion, in what follows we consider simplified systems with three components. A *sender* provides the media data, which could be encoded from a live media input or could be obtained from a pre-encoded stream. A *client* initiates the request for a media stream and plays it back to the end user. A *proxy* is an intermediate node of the network that facilitates the interaction between client and sender. Note that in some cases a proxy contains all the information requested by the client and thus in effect acts as the sender. Our focus in this chapter will be on scenarios where the sender is the "main" source for the media stream, and where the proxy plays an auxiliary role.

More complex systems can be used in practice [4,11,41,52], for example, multiple proxies could play a role, or the content could be delivered from more than one sender, to multiple users, through multiple network routes, etc. In recent years, peer-to-peer (P2P) networks [45,54,57] have also been studied as alternatives to traditional client–server architectures. However, a discussion of the simpler system is sufficient to understand the various bandwidth adaptation mechanisms.

4.1.2 Chapter Outline

The chapter is organized as follows. In Section 4.2, we first discuss how the bandwidth variations affect the received multimedia quality, especially for delay-constrained transmission. Here we introduce an important concept, namely that of an end-to-end delay constraint for a multimedia communication system. Then we provide a global overview of a bandwidth adaptation system architecture in Section 4.3, discussing the trade-off and criteria to choose a particular adaptation mechanism for a given application. In Section 4.4, we describe different coding techniques that can be used to adjust the coding rate of a multimedia source to match the available bandwidth, and briefly review the optimization techniques. Finally, Section 4.5 provides a summary of the main ideas introduced in this chapter.

4.2 IMPACT OF AVAILABLE BANDWIDTH ON MULTIMEDIA QUALITY

In order to understand bandwidth adaptation mechanisms for multimedia applications, it is necessary to understand first the impact of bandwidth variations on received media quality.

4.2.1 Downloading and Streaming

In contrast to data communication or to simple media downloading, real-time media streaming is often subject to strict delay constraints. The main difference with respect to a download application is that *media playback starts as data is still being received, so that playback could be interrupted if the decoder ran out of data to decode*.

Typical streaming applications operate as follows:

1. **Data request**. A request is sent to the media sender so that data streaming to the receiver starts.
2. **Client buffer loading**. As data starts to reach the client, decoding does not start immediately. Instead the client waits to have "enough" data to start decoding.
3. **Playback**. Once there is sufficient data available at the client, playback starts and at that point only relatively minor adjustments in the playback timing are possible, so that the rate at which media is played back (e.g., the number of video frames per second) needs to remain nearly constant.[1] An example of playback adaptation can be found in [39] and in Chapter 16.

There are multiple strategies for clients to determine that enough data is available to begin decoding. For example, a target total number of bits may have to be buffered before playback starts. Alternatively, a predetermined time for buffering (e.g., a few seconds) could be chosen so that users always experience the same time latency before playback starts. Finally, a more practical approach may be to wait until the number of bits that represents a selected playback time has been received (e.g., the number of bits needed to encode a video segment with a predetermined duration). Details can be found in Chapter 14.

Note that the primary concern in many applications is playback latency, rather than storage at the receiver. Thus, in applications such as streaming to a computer, where there is plenty of memory available, the amount of data loaded before playback may still be kept small to limit the initial latency in the system. Note also that latency is particularly important when the user is expected to frequently switch media sources, as the latency penalty will be incurred every time the user switches.

Regardless of which technique is chosen to preload the buffer, at the time when playback starts, the decoder will have a certain number of bits available for decoding. This available data translates into a playback duration (e.g., if there are

[1] More precisely, the number of frames per second received needs to be consistent with what the receiver expects to play back. Thus, adaptation mechanisms that involve both transmitter and receiver are possible (e.g., so that fewer frames per second are transmitted and played back when bandwidth availability is reduced).

N compressed frames in the decoder buffer and K frames/second are decoded, then the decoder will be able to play from the buffer for N/K seconds). Thus the amount of data available in the decoder buffer at a given time tells us for how long playback could proceed, even if *no data was to be received* from the network.

4.2.2 Available Bandwidth and Media Quality

To understand the need for bandwidth adaptation, consider what would happen if reductions in channel bandwidth were not matched by reductions in the source coding rate. Assume a constant media playback rate, for example, a fixed number of frames per second in a video application. When network bandwidth becomes lower, if the number of bits per frame does not change, then the number of frames per second received is bound to decrease. Since the receiver continues playing frames at the same rate, eventually there will be no frames left for playback in the receiver buffer and thus playback will be interrupted. This will be a *decoder buffer underflow*.

Thus, as network bandwidth fluctuates, bandwidth adaptation is needed to ensure that playback is not interrupted. Roughly speaking, this requires that the number of frames/second provided by the network matches (on average) the number of frames/second consumed for playback at the receiver. The general goal of bandwidth adaptation mechanisms will be to manage the quality of the frames transmitted so that when the available bandwidth is reduced, the rate (and hence the quality) of transmitted frames is also reduced. In essence, the goal is to avoid service interruptions by lowering the media quality in a "graceful" manner.

We next provide a more detailed discussion of the delay constraints that are present in a real-time media communications system.

4.2.3 Delay-Constrained Transmission

Consider, as an example of delay-constrained transmission, a real-time video transmission system where all operations have to be completed within a predetermined time interval. The *end-to-end delay* from a video source to a destination contains the following five components, as illustrated in Figure 4.1:

- *Encoding delay* ΔT_e: the delay required to capture and encode a video frame.
- *Encoder buffer delay* ΔT_{eb}: the time the encoded media data corresponding to a given frame spends in the transmission buffer. Note that this delay could be zero if the channel bandwidth is higher than the bit rate produced by the encoder, that is, data transmission would start as soon as video data is placed in the buffer.

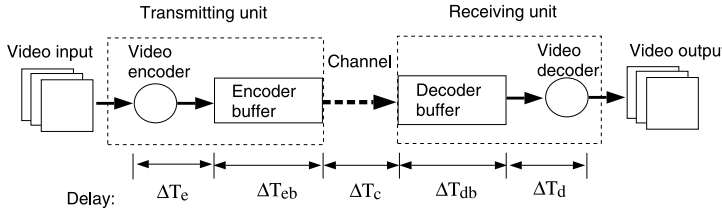


FIGURE 4.1: Delay components of a communication system.

- *Transmission channel delay* ΔT_c: the delay for encoded data being transmitted through the network, caused by transmission, congestion, and possible retransmission over a lossy channel.
- *Decoder buffer delay* ΔT_{db}: the time for encoded data to wait in the decoder buffer for decoding. This delay allows smoothing out the variations in transmission delay and in rates across frames.
- *Decoding delay* ΔT_d: the delay for decoding process and final display.

Both encoding and decoding delays are usually fixed, so that we focus primarily on the remaining delay terms. Note that when considering pre-encoded media, the encoding delay is not considered as the video is already encoded off-line and ready for transmission. In this case we can consider that the encoding buffer is of infinite size and contains the complete encoded sequence.

In summary, the main constraint in the system is the status of the decoder buffer, that is, as long as the decoder buffer contains data, decoding can proceed. Thus bandwidth adaptation mechanisms should be designed with the objective of ensuring that the decoder buffer is not "starved" of data. In some cases, accurate and timely information about the decoder buffer state is available, which can then be used to make bandwidth adjustments (this would be the case, for example, if the client makes bandwidth adaptation decisions). In other cases, exact information may not be available, but the state of the buffer could be estimated using, for example, estimates of available bandwidth.

Different applications may have different delay requirements. For example, for live interactive video, a round-trip delay between 150 and 400 ms is usually required, while an initial play-out delay up to a few seconds is acceptable for noninteractive video streaming. Once selected, end-to-end delay requirements impose a constraint on the encoding rate for each frame.

4.3 OVERVIEW OF BANDWIDTH ADAPTATION ARCHITECTURES

We are now ready to define more precisely what we consider as *bandwidth adaptation mechanisms*: These are techniques that enable the rate of a media stream to

be modified during a playback session (i.e., while a user is connected and receiving content for playback) in order to accommodate changes in the network (e.g., changes in available bandwidth, congestion, and packet losses).

In order to provide a rough classification of bandwidth adaptation architectures, note that defining a specific mechanism requires choosing:

- *Adaptation points*, that is, the locations in the network where the bit stream is adapted to match specific bandwidth requirements. For example, adaptation could take place at the sender, at a proxy, or even at the client application.
- *Decision agents*, that is, the component within the system where decisions about transmission rate changes are made. This decision could be made at the sender, a proxy, or the client, based on whatever information is available at that point in the network.
- *Coding techniques*, that is, the source coding techniques designed to facilitate bandwidth adaptation. Note that not every technique is appropriate for a certain combination of adaptation point and decision agent. These techniques are discussed in Section 4.4.

It is important to note that, in general, bandwidth adaptation decisions need *not* be made at the same point in the network where the adaptation itself takes place. A concrete example of this situation is client-driven techniques where each client evaluates the status and parameters of its own transmission link and requests to the sender changes to the streaming parameters; in this case bandwidth adaptation decisions are made by clients and put in place by the sender.

In general, the choice of adaptation point and decision agent for a particular system depends on what information is available to each component of the system (client, sender, or proxy if there is any), on available computational resources, and on the characteristics of the bit stream.

4.3.1 Trade-Offs

Before discussing specific architectures in detail it is useful to understand how operating at client, server, or proxy leads to different trade-offs.

First, note that adaptation decisions should be based on available information about (i) the state of the network (e.g., bandwidth availability) and (ii) the relative importance of information encoded in the media stream (e.g., how much degradation will result from dropping one of the layers in a scalable representation, or in general the rate–distortion characteristics of different parts of the stream).

Figure 4.2 illustrates that source-related information is likely to be known more accurately at the sender (which can analyze media as it encodes or extracts relevant information from an existing stream) than at the client (which must rely on information provided to it by the server). Similarly, more efficient adaptation

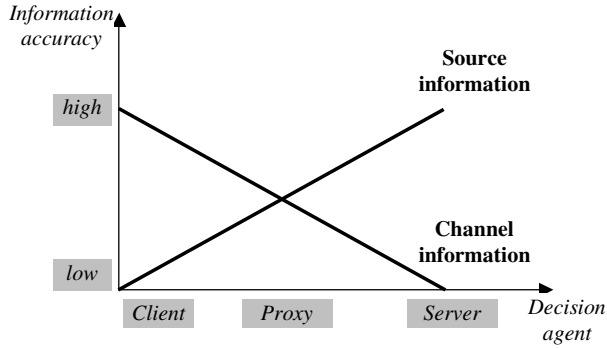


FIGURE 4.2: Trade-off between the accuracy of source information and channel information available at various network locations.

decisions are possible when information about the state of the network is timely and accurate. Ideally, information should be available about the channel behavior observed by the client. Thus the client has access to more accurate information in a more timely way, as it observes packet arrival events. Figure 4.2 also illustrates that since the most accurate information is not available in a single place, some algorithms will entail information exchange between client and sender. Examples include where the client sends packet status feedback to the sender or where the sender provides the client with information about the source, such as an "RD preamble" [40].

Second, two major factors affect the performance of a bandwidth adaptation algorithm for a single client, namely (i) the granularity with which bandwidth can be adapted and (ii) the speed with which changes can be made to react to variations in network behavior.

Figure 4.3 illustrates that when actual adaptation (i.e., change in the rate at which data is sent to the client) is performed at the server, finer granularity can be achieved. Conversely, when adaptation takes place at the server the reaction time may be longer because packets resulting from adaptation will take longer to arrive at the client.

Third, it is often important to consider system-level trade-offs. Not only how a particular client's quality is affected by bandwidth adaptation, but rather how adaptation affects overall network performance. Figure 4.4 illustrates how system scalability and overall network utilization are affected by choices made in the bandwidth adaptation mechanism. If decisions on how to change the bandwidth, and even adaptation itself, are performed close to the client, the system will be easier to scale, since more of the computation cost will be borne by the clients. However, if bandwidth adaptation is performed close to the clients this will be to

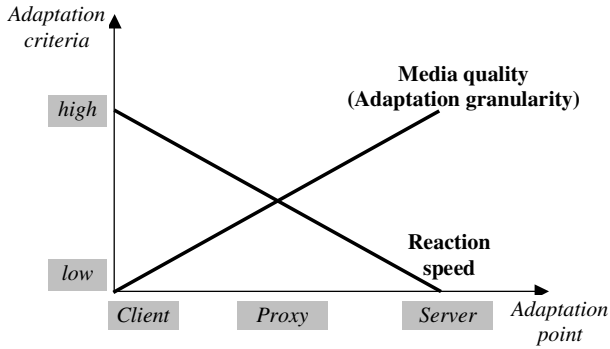


FIGURE 4.3: Comparison on bandwidth adaptation flexibility and reaction time to serve a single client.

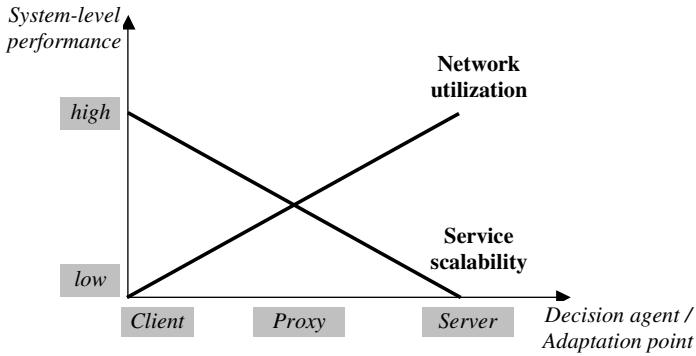


FIGURE 4.4: Comparison on service scalability and overall network utilization when serving multiple clients.

the detriment of overall network utilization, since data rate reductions will only reduce utilization close to the client.

4.3.2 Where Should the Adaptation Points Be?

As introduced earlier, an adaptation point is the system component where the bandwidth of the stream is physically changed. Each possible choice of location for the adaptation points has different advantages in terms of various performance metrics of interest.

4.3.2.1 Sender

The sender has the most flexibility in terms of compression format, since it can adjust the coding parameters in real time (in case live encoding to single users is performed), and can switch between several simultaneously produced streams (simulcast), etc. Moreover, the sender is typically least constrained in terms of storage and processing. Generally, then, adaptation at the sender provides the most flexibility from a source coding perspective. In practice, this means that when the sender performs bandwidth adaptation, this makes it possible to achieve finer grain bandwidth adaptation will be possible, as shown in Figure 4.3, with least penalty in terms of quality at the receiver.

There are several drawbacks in server-driven bandwidth adaptation. The sender is furthest away from the client; thus when congestion occurs in the network there may be a delay before the bandwidth adaptation can take effect (see Figure 4.3). Moreover, depending on where network information is being captured, this information may be unreliable. If bandwidth changes are requested by the client (see Section 4.3.3), and are thus based on more reliable information about the state of the network, letting the adaptation happen at the server means that the delay in reacting can be significant, which can reduce the effectiveness of bandwidth adaptation. If the sender itself is estimating the network state, it will be able to adapt faster, but may not have sufficiently accurate information about the network to be effective.

Adaptation at the server also presents problems in terms of *scalability* in cases where data is being broadcast to multiple clients. First, each server may be limited in the number of clients it can provide content to simultaneously, in particular if compression or bandwidth adaptation is computationally expensive. Second, the server may have to create separate versions of the same content for clients with different Internet access bandwidths, for example, one for 56K modem connections, and another for DSL, etc. This will in turn create a heavy traffic load in the local network around the server, which may also have a negative impact on other content being served.

Physical adaptation is closely related to the coding techniques applied in a particular application. Since the sender can access the source more flexibly, a number of adaptation techniques have been proposed. Such techniques include source rate control (i.e., adjusting coding parameters during the encoding process [29,76]), rate–distortion optimized packet scheduling [16,48], and switching between different bit streams or layers [19,67].

4.3.2.2 Client

Bandwidth adaptation at the client essentially means that the client does not decode all the content it receives. This would be beneficial only in terms of low-

ering the complexity of decoding or avoiding decoding of lower priority data that is likely to be corrupted. This type of adaptation in general requires a coding format that supports complexity scalability. The reconstructed quality is related to the complexity of the decoder used. For example, van der Schaar and de With [70] proposed to reduce the memory costs of an MPEG-2 decoder by re-compressing the I- and P-reference pictures prior to motion-compensated reconstruction. Transform coding and motion estimation algorithms with complexity scalability have also been studied [35,36,55]. In addition to the complexity-scalable modifications of existing decoders, recent research has also attempted to model the complexity based on the compressed source characteristics and the decoding platform capabilities [69]. Clearly, such a system would have no impact on the traffic being carried by the network, and thus would not contribute to reduced congestion.

4.3.2.3 Proxy

Proxies are a good compromise between server and client adaptation. A proxy is responsible for a smaller number of clients than a server, which improves scalability and traffic balancing, and is also closer to the clients so it can respond faster to changes that affect the client. Most often, the source information at the proxy is stored as a pre-encoded stream received from the original media server, and thus transcoding is widely employed for adaptation at this point. For example, Shen *et al.* [59] have proposed a transcoding-enabled proxy caching system to provide different appropriate video quality to different network environments.

4.3.3 Sender-, Client-, and Proxy-Driven Adaptation

Note that there are many situations where the changes in source coding rate are implemented at one point in the network, based on decisions made somewhere else. A particular case of interest is that where the client makes decisions about data to be transmitted and submits these to the server.

4.3.3.1 Client-Driven Decisions

Information about the status of decoded data is best when bandwidth adaptation decisions are made by the client, in particular when these decisions are based on accurate and fine grain information, for example, arrival or not arrival of individual packets. The client-driven approach can also help reduce the processing complexity at the server side, thus allowing the server to support more clients simultaneously.

Examples of this method include the Adaptive Stream Management (ASM) process of the SureStream technology used in RealSystem 8 [19]. Two major

components involved in this process are a compressed media file, which contains multiple independently encoded streams of a given source, and an ASM rule book, which describes various forms of channel adaptation that involve selecting combinations of encoded streams as a function of the channel status (including bandwidth, packet loss, and loss effect on the reconstructed signal). The ASM rule book is sent to the client at the beginning of a session. During transmission, the client monitors the rate and loss statistics of arriving packets, and then instructs the server to subscribe to a rule, or combination of rules, to match the current channel behavior. Another example is that of receiver-driven adaptation in the context of multicast delivery [15,46].

A drawback is that, while the client makes the decisions, these need to be implemented in either a server or a proxy, as in the example given earlier. This is because bandwidth adaptation at the client can only help in reducing the complexity of decoding. Thus there will be some latency before the changes in bandwidth can be implemented. Another potential drawback is that some clients, such as low-power hand-held devices, may not have sufficient computation power to implement a complex decision process.

4.3.3.2 Proxy-Driven Decisions

In this type of system, proxies can estimate the state of the network (or get this information from the client) and then decide on appropriate changes to the bandwidth to be used by the media stream. For example, a proxy can select certain packets to be forwarded to the client, change transcoding parameters, or send instructions to the server so that the server can modify the information it transmits. Chakareski *et al.* [10] have proposed a rate–distortion optimized framework in the scenario of proxy-driven streaming. At any given time, the proxy determines which packets should be requested from the media server and which should be retransmitted directly from the proxy to the client in order to meet rate constraints on the last hop while minimizing the average end-to-end distortion. Approaches that have investigated the role of proxies in terms of both streaming and caching also include [50]. The proxy, usually located at the edge of a backbone network, coordinates the communication between the source server and the client, and can potentially achieve better bandwidth usage than a client- or server-driven system.

4.3.3.3 Server-Driven Decisions

Finally, in this scenario estimates of network state are provided to the server, which decides on data to be sent to each client. Feedback is often required for this approach. The server-based approach has the most information about the source (e.g., about possible rate–distortion operating points) and thus can work with a more flexible and efficient adaptation algorithm in terms of source coding. In

addition, the server can regulate connections with different clients as a whole to improve overall bandwidth utilization. The main disadvantage of this approach is that the server may not have reliable or timely information about the state of the network near the client.

As an example, the work of Hsu *et al.* [29] performs source rate control by assigning quantizers to each of the video blocks under the rate constraints at the encoder, where the available channel rate is estimated by incorporating the channel information provided by the feedback channel and a priori channel model. Related work [30] shows that source rate control algorithms can also be applied for various types of network-related rate constraints. Intelligent transport mechanisms, such as optimal packet scheduling for a scalable multimedia representation [16,48], can also be performed at the server.

4.3.4 Criteria and Constraints

This section provides an overview of different criteria that can be applied to select a bandwidth adaptation mechanism for a given application. We emphasize that this is, by necessity, a qualitative discussion. Many of the techniques that are mentioned in this chapter have only been proposed in a research context and have not been fully tested in a more realistic network environment. Moreover, a quantitative comparison of the various methods is likely to be very complex, as should be clear given the number of criteria to be considered in general.

4.3.4.1 Media Quality

Clearly, the ultimate criterion to evaluate the performance of a bandwidth adaptation mechanism should be the resulting subjective media quality at the receiver in the presence of typical bandwidth variations. Some progress has been made in devising objective metrics that can capture the perceptual quality of media under various compression strategies [34,37,68]. These objective metrics are most advanced for the analysis of audio sources, somewhat less so for video applications. Approaches that can compare meaningfully different methods in the presence of variations in the network behavior (e.g., bandwidth fluctuations, packet losses) are not that readily available.

Service interruptions, such as those that might occur if no bandwidth adaptation is used, are obviously undesirable, and so one could, for example, compare different techniques in terms of their outage probability (the probability that perceptual quality over a given period of time drops below acceptable levels). A comparison would still be challenging: for example, an end user may deem two configurations with different, but nonnegligible, outage probability to be equally unacceptable.

Quality evaluation is also more complicated once a bandwidth adaptation mechanism is put into place because these mechanisms are dynamic in nature.

Thus, they operate only when the bandwidth falls below certain levels and lead to changes in the media quality (e.g., in the context of video, variations in frame rate, frame resolution, frame quality). In this situation, it is unclear whether users will base their quality assessment on the perceived "average" quality, the worst case quality level, the duration of the worst quality, etc.

Many currently deployed practical media streaming systems generally select one of multiple streams, that is, the one whose bandwidth best matches the bandwidth available to the end user; in many cases no adaptation is possible within a stream. Thus system designers only have a limited amount of real-life experience with bandwidth adaptation mechanisms. It also follows from this that the impact of various such mechanisms on perceptual media quality is not as well understood.

In summary, while progress has been made toward understanding subjective quality metrics for various types of media, challenges remain in addressing situations where quality adaptations (not to mention information losses) take place. For this reason, and also to facilitate bandwidth adaptation mechanisms, objective quality metrics, such as peak signal-to-noise ratio (PSNR), are often used. For example, authors have proposed optimizing average PSNR (e.g., [29]) or minimizing the loss in PSNR introduced by bandwidth adaptation, with respect to the PSNR achieved when the media stream transmitted at a given target bit rate (e.g., [16]).

4.3.4.2 End-to-End Delay, Reaction Time, and Latency

As discussed earlier, a longer end-to-end delay facilitates preserving a consistent quality level in the face of bandwidth fluctuations. Roughly speaking, a longer end-to-end delay leads to more multimedia units (e.g., video frames) being stored in the decoder buffer so that the application can absorb short-term variations in bandwidth.

When the end-to-end delay is not long, the reaction time of the adaptation system to changes in bandwidth becomes important. The system has to detect relevant variations in network behavior and then trigger the necessary changes in the media stream so as to best match bandwidth availability. Ideally, this should happen sufficiently fast so that the end user does not suffer from negative consequences of mismatch between network availability and stream requirements.

Note that this leads to interesting design trade-offs in the context of the adaptation architectures discussed earlier. For example, a faster reaction may be possible if the sender makes adaptation decisions, but these may suffer from a somewhat worse knowledge of network status at the client.

Long end-to-end delay is a practical solution only for one-way transmission applications. For two-way communications, a long delay will limit the interactivity. Even in the case of one-way communications, excessive end-to-end delays

lead to higher initial latencies, which would be undesirable if the user switches multimedia streams frequently.

4.3.4.3 Complexity

An interesting challenge in architecting a bandwidth adaptation mechanism is that several components (client, proxy, and sender) can play a role. Thus, taking into account complexity requires identifying first which of these components is least constrained in terms of complexity.

While it may appear at first obvious that the server will be richer in computation resources, this may not be true in general. In particular, for applications such that each sender is responsible for multiple clients, overall computation power at the sender may be significant, but computation power for each client served may have to be limited in order to ensure scalability.

4.3.4.4 Storage

Storage constraints are unlikely to be of much importance, except for mobile applications. It is also worth mentioning the complexity implications of shared storage. While massive storage is often available at a very low cost, there may be significant computation costs involved in managing a large number of streams being produced out of a shared storage device. In this context, bandwidth adaptation tasks (e.g., switching between two pre-encoded versions of a media stream) may add to the complexity of the system.

4.3.4.5 Information Overhead

Consider existing digital video delivery systems (e.g., a digital cable system) and compare them with systems such as those we have discussed. In a digital cable system bandwidth is expected to be reliable and there is minimal interaction between receiver and sender.

Instead, proposed bandwidth adaptation architectures often require auxiliary information to be exchanged between client and sender. Examples of this extra information include estimates of channel state, acknowledgments of reception of information, and rate distortion "preambles."

4.3.5 Examples

Depending on the type of application, network characteristics, and optimization criteria, it is possible that different bandwidth adaptation architectures may be preferable. This section sketches some examples that allow us to discuss how particular choices of architecture can be made. Note that we are not proposing

a concrete methodology for architecture selection. Moreover, there may be several architectures that are suitable for a given scenario. Thus, these examples are meant to illustrate possible approaches in the design process, rather than to claim optimality for any of the different approaches.

In the case of one-to-one interactive two-way communication, such as video conferencing, a relatively short end-to-end delay, usually between 150 and 400 ms, is required. Thus it may be preferable for the decision agent and the adaptation point to be located close to each other so as to avoid excessive delay before adaptation takes place. One possible solution is presented in Table 4.1. When a server receives feedback indicating a channel status change, it estimates the new available channel bandwidth [29] and then makes the corresponding adaptation decision. The adaptation can be as simple as skipping transmission of some of the packets to prevent video freezing or losing connections. More advanced techniques can also be applied, such as, for example, adjusting the video codec parameters to increase or decrease the encoding rate. However, the limited bandwidth and stringent delay requirement in this case may limit the potential performance gains achievable through adaptation.

In the case of one-to-one one-way streaming, a longer initial play-out delay, of up to a few seconds, is likely to be acceptable. A more appropriate solution for this case would then be client-driven streaming, such as the SureStream technology used in RealSystem [19]. During the streaming session, a client monitors the bandwidth and loss characteristics of its connection and makes decisions based on more accurate and fine grain channel information. Then it instructs the server to take certain actions, for example, switching to different streams, or selectively transmitting only the number of layers in a layered codec that the given link can support, such that the end-to-end distortion can be minimized over the current channel condition.

In the case of Internet broadcast or multicast, the traditional single-server-based delivery system faces several major problems, including service scalability and traffic load unbalance, as discussed in Section 4.3.2. To address these problems, today's content delivery networks employ multiple geographically dis-

Table 4.1: Examples of bandwidth adaptation architectures for different video communication applications

Application	Bandwidth estimator	Decision agent	Adaptation point
One-to-one interactive two-way communication	Server	Server	Server
One-to-one one-way streaming	Client	Client	Server
Internet broadcast	Client	Proxy	Proxy

tributed edge servers to either forward the incoming live content or deliver the on-demand content from their local cached storages to their local clients. It is possible to directly extend the client-driven server-adaptation technique to multicast delivery. However, it may be better if proxy servers can take a more active role in the bandwidth adaptation process, as the bandwidth limitations often occur in the access network, such as a DSL connection. This proxy-based architecture, as shown in Table 4.1, can reduce the reaction time, avoid congestion in the Internet, and provide appropriate qualities for clients with different connections.

4.4 CODING TECHNIQUES FOR BANDWIDTH ADAPTATION

Previous sections have discussed general bandwidth adaptation architectures under the assumption that a mechanism would be available to adjust the number of bits transmitted to represent multimedia sources. In this section we provide an overview of coding techniques that can be used in practice to adjust the coding rate of transmitted multimedia sources.

Many criteria can be used to compare different coding techniques. Since their primary goal is to enable representation of the sources at different rate levels, one primary concern is what reproduction quality is achievable at each of those rate levels. Thus, as for all coding techniques, it will be important to know the rate distortion (RD) characteristic of each possible operating point.

In addition, there are other criteria that are specific to bandwidth adaptation scenarios.

First, it will be useful to provide as many rate operating points as possible (i.e., so that fine grain adaptation is possible). Generally speaking, finer grain in the adaptation will come at the cost of increases in achievable distortion for a given rate.

Second, some coding techniques will only allow adaptation to take place at the encoder, while others will enable adaptation anywhere in the network. The latter model will typically also lead to some RD inefficiency.

Finally, adaptation granularity can be evaluated not only in terms of achievable rate points, but also in terms of temporal constraints. In some applications it may be desirable to adjust the rate of individual temporal components (e.g., frames in a video sequence), which again may come at the cost of reduced RD performance.

4.4.1 Rate Control

Rate control techniques are used *during the encoding process*. They rely on adjusting multiple coding parameters to meet a target encoding rate. We focus here on rate control techniques for video, as in both audio and speech coding variable bit rate encoding techniques (which tend to lead to more challenging rate control) are not as popular.

In the case of video, when the same coding parameters (e.g., quantization step size, prediction mode) are used throughout a video session, the number of bits per frame will change depending on the video content so that the output bit rate will vary from frame to frame. Thus, when video content is "easy" to encode (e.g., low motion and low complexity scenes) and a given quantization selection is chosen, the rate will tend to be lower than if the same combination of quantizers was used for a more complex scene. Even though the encoder and decoder buffers can help smooth the (short term) variations in the rate per frame, a rate-control algorithm is usually needed in order to allocate bits among all coding units (e.g., frame, macroblock, or others) to maximize the end quality subject to the rate constraint.

All major video coding standards provide mechanisms for flexible coding parameter selection, with the chosen parameters being communicated to the decoder as overhead. To illustrate the key concepts, here we concentrate on a hybrid video coding structure, which is an essential component of all major standards, and in particular on one based on block-based motion-compensated prediction and Discrete Cosine Transform (DCT) coding. In such a framework, a frame is divided into a number of macroblocks (MB), each containing a luminance block (of size 16×16) and two chrominance blocks (e.g., 8×8 Cb and 8×8 Cr).

A series of coding decisions have to be made in compressing each frame:

1. Type of frame (e.g., I-, P-, or B-frame) to be chosen or whether the frame is to be skipped, that is, not encoded at all.
2. Mode to be used for each MB, for example, Intra, Inter, Skip, etc.
3. If an MB is coded in INTRA mode,
   (a) What quantization step size (QP) should be used to code the DCT coefficients of each block?
   (b) If intra prediction is allowed, for example, in H.264, how to perform intra prediction; that is, how to generate the reference block from the neighboring blocks in the same frame.
4. If an MB is coded in INTER mode,
   (a) What motion compensation should be used, for example, with or without overlapping, reference frame selection, search range, and block size?
   (b) How to code the residual frame, for example, which QP should be chosen?

The options just listed are by no means exhaustive; they are intended to serve as an illustration of the range of coding mode choices available in modern video coders. Note that as the number of possible modes increases so does the complexity of the encoding process and the importance of selecting efficient rate control algorithms. In fact, one can attribute much of the substantial coding gains achieved by recent standards, such as H.264/MPEG-4 part 10 AVC [2], to the ad-

dition of several new coding modes combined with efficient mode decision tools based on RD criteria.

A very common approach to rate control is to modify the QP [29,65]. A large QP can reduce the number of encoded bits at the expense of an increased quantization error, and vice versa. However, changing QP only while keeping the other coding modes constant may not achieve the optimal performance. For example, coding in INTER mode is effective in most cases when changes in video content are due to the motion of objects in the scene. Instead, INTRA mode may be more appropriate in situations when there is a significant difference between coded and reference images, such as uncovered regions (part of the scene is uncovered by a moving object) or lighting changes. However, the optimal selection of INTER/INTRA coding for a given block may in fact be different at different QPs. More general rate-control algorithms should optimize different coding parameters as well, such as frame rate, coding modes for each frame and MB, and motion estimation methods [13,24,76].

Each combination of these coding parameters results in a different trade-off between rate and distortion. Thus efficient parameter settings will be those that are chosen based on rate–distortion optimized techniques. The typical problem formulation seeks to select the coding parameters that minimize the distortion under constraints on the rate (usually the average bit rate over a short interval). Many solutions have been proposed, with some based on heuristic approaches and others following well-known techniques such as Lagrangian optimization or dynamic programming. More details on this topic can be found in [53,65] and references therein.

The computation involved in the optimization approach mainly includes two parts: (1) collection of rate–distortion data, which may require to actually code the source with all different parameter settings, and (2) the optimization algorithm itself. Both parts can be computationally intensive but often the data collection itself represents the bulk of the complexity, which has led to the development of numerous approaches to model the R–D characteristics of multimedia data [20, 27,28,43]. Two main types of modeling approaches have been reviewed in [28]. One class of techniques [27] involves defining models for both the coding system and the source so that R–D functions can be estimated before actually compressing the source. The modeling accuracy depends on the robustness of the R–D model to handle different source characteristics. The second class of techniques requires actually coding the source several times and then processing the observed R–D data to obtain a complete R–D curve. Examples include the estimation algorithms proposed in [20,43]. These approaches are usually more computationally intensive, as well as more accurate, since they estimate the parameters from the actual coding results of the corresponding source.

In summary, the choice of an appropriate rate-control algorithm depends on the multimedia application, especially on whether it is delay constrained. For in-

stance, a complicated approach can be used for off-line coding. However, heuristic approaches may be more practical for online live multimedia communications.

4.4.2 Transcoding

The term "media transcoding" is normally used to describe techniques where a compressed media bit stream format is converted into format. It is often used at either the server or the proxy when the source is only available as a pre-encoded stream so as to match limitations in transmission, storage, processing, or display capabilities of specific network, terminals, or display devices. Transcoding is one of the key technologies for end-to-end compatibility of two or more different networks or systems operating with different characteristics and constraints.

Because the transcoder takes as an input a compressed media stream, the decoded quality of the transcoder output is limited by the input stream, which has certain information loss compared to the original source. However, the transcoder has access to all the coding parameters and statistics, which can be easily extracted from the input stream. This information can be used not only to reduce the transcoding complexity, but also to improve the quality of the transcoded stream using a rate–distortion optimization algorithm.

A typical application of transcoding is to adapt the bit rate of a precompressed video stream to a reduced channel bandwidth. Clearly, we can first reconstruct video back to the pixel domain by decoding the input compressed bit stream and then re-encode the decoded video to meet the target bit rate. The rate control techniques described earlier can then be used at the encoding stage. However, the whole process (decoding and encoding) is very computationally expensive, and more efficient techniques have been developed that reuse information contained in the original input bit stream.

The main drawback of these more efficient transcoding techniques is the drift problem (which will also arise in some of the other coding techniques introduced in this chapter). Drift is created if the reference frame used for motion compensation at the encoder is different from that used at the decoder. This happens, for example, when the transcoder simply requantizes the residual DCT coefficients with a larger QP to reduce the output bit rate. When a decoder receives the transcoded bit stream, it reconstructs the frame at a reduced quality and stores it into the frame buffer. If this frame is used as prediction for future frames, the mismatch error is added to the residual of the predicted frame, leading to a degraded quality for all the following frames until the next I frame. Based on the trade-off between complexity and coding quality, we briefly describe two basic transcoding architectures, namely, open-loop and closed-loop transcoders.

Figure 4.5a shows an open-loop architecture based on a requantization approach [51]. The bit stream is dequantized and requantized to match the bit rate

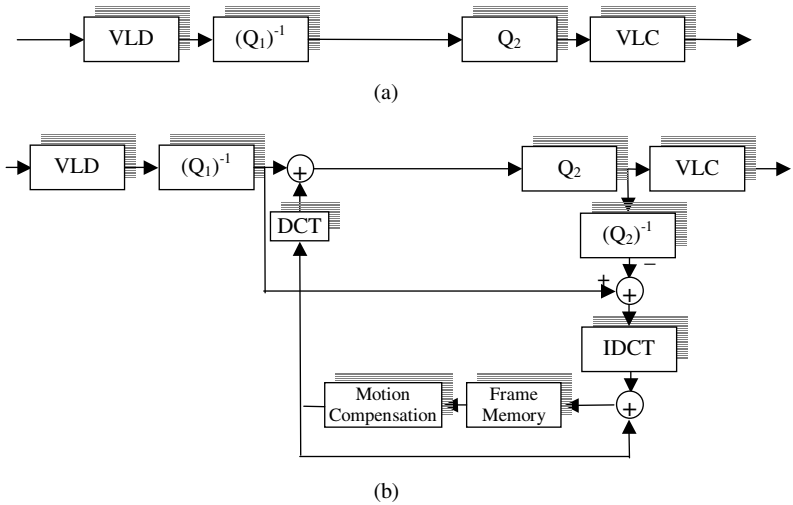


FIGURE 4.5: Transcoding architectures for bit-rate reduction [72]: (a) Open loop. (b) Closed loop.

target. Another open-loop approach is to discard the high-frequency DCT coefficients [22,66] to reduce the rate. All these operations work on the DCT coefficients directly, and thus the computation load is light but this architecture leads to drift.

A closed-loop architecture introduces an extra drift-compensation module, as shown in Figure 4.5b [7], to eliminate the mismatch between the reference frames at the encoder and decoder. The frame memory in the configuration holds a difference signal and is added to the residual component to compensate for the prediction mismatch. The additional DCT/IDCT can be removed by using DCT-domain MC [12,47,62]; several simplified DCT-domain transcoders are described in [8,42]. Compared to the straightforward approach with cascaded decoder and encoder, this approach usually requires less computation to achieve almost equivalent quality with the exception of slight inaccuracy due to nonlinearity introduced by clipping and rounding operations or floating point inaccuracies [79]. Even for the cascaded pixel-domain transcoder, the encoder can be simplified by reusing the motion vectors and other information.

Regardless of the transcoding architecture, a rate-control algorithm is applied to yield the desired bit rate. As discussed in [56], a two-pass rate-control approach typically performs better than a single-pass approach, since information obtained from the results of the first pass (e.g., selected RD operating points of all frames) can be used in the second pass of the algorithm to improve the qual-

ity. A transcoder can be regarded as a special two-pass approach [78], where the first pass creates the input compressed bit stream and the second pass creates the output compressed stream based on the results of the first pass. For example, bit allocation to each frame ideally depends on the frame complexity, which is not easy to estimate for real-time video encoding but can be obtained more accurately from the number of bits each frame spent in the input bit stream. Similarly, optimal requantization for transcoding [26,63,75] requires the knowledge of the original DCT coefficients statistics, which can be estimated from the input compressed bit stream as well.

In addition to being used for bit rate adaptation, video transcoding is also widely employed for spatial resolution and frame rate adaptation. More details on different transcoding techniques are well discussed elsewhere [72,78].

4.4.3 Scalable Coding

The coding methods discussed so far in this chapter aim to optimize the media quality for a fixed bit rate. This poses a problem when multiple users are trying to access the same media source through different network links and with different computing powers. Even in the case of a single user accessing one media source over a link with varying channel capacity, relying on an often complex rate-control algorithm to make rate adjustments in real time may not be practical (e.g., if the changes in rate have to occur in a very short time frame). Scalable coding is thus designed to facilitate bandwidth adaptation over a given bit rate range, as well as to provide error resilience for potential transmission errors.

Scalable coding, or layered coding [1,3,21,38,61], specifies a multilayer format in which a video sequence is coded into a base layer and one or more enhancement layers. The base layer provides a minimum acceptable level of quality, and each additional enhancement layer incrementally improves the quality. Thus, graceful degradation in the face of bandwidth drops or transmission errors can be achieved by decoding only the base layer, while discarding one or more of the enhancement layers. The enhancement layers are dependent on the base layer and cannot be decoded if the base layer is not received. A scalable compressed bit stream typically contains multiple embedded subsets, each of which represents the original video content in a particular amplitude resolution (called SNR scalability), spatial resolution (spatial scalability), temporal resolution (temporal scalability), or frequency resolution (frequency scalability or, in some cases, data partition). Scalable coders can have either coarse granularity or fine granularity. In MPEG-4 fine granularity scalability (FGS) [38], the enhancement-layer bit stream can be truncated at any point, where the reconstructed video quality increases with the number of bits received.

Unfortunately, all current scalable video coding standards suffer to some degree from a combination of lower coding performance and higher coding complexity,

as compared to nonscalable coding. A key issue is how to exploit temporal correlation efficiently in scalable coding. It is well known that motion prediction increases the difficulty of achieving efficient scalable coding because scalability leads to multiple possible reconstructions of each frame [58]. In this situation either (i) the same predictor is used for all layers, which leads to either drift or coding inefficiency, or (ii) a different predictor is obtained for each reconstructed version and used for the corresponding layer of the current frame, which leads to added complexity. MPEG-2 SNR scalability with a single motion-compensated prediction (MCP) loop and MPEG-4 FGS exemplify the first approach. MPEG-2 SNR scalability uses the enhancement-layer information in the MCP loop for both base and enhancement layers, which leads to drift if the enhancement layer is not received. MPEG-4 FGS provides flexibility in bandwidth adaptation and error recovery because the enhancement layers are coded in "intra" mode, which results in low coding efficiency, especially for sequences that exhibit high temporal correlation. Some advanced approaches with multiple MCPs are described elsewhere [5,31,58,71,77]. In summary, the design goal in scalable coding is to minimize the reduction in coding efficiency while realizing the scalability to match the network requirements. More details on scalable video coding can be found in Chapter 5, and details on scalable audio coding can be found in Chapter 6.

An alternative to bandwidth adaptation and reliable communication is Multiple Description Coding (MDC) [25,74]. With this coding scheme, a video sequence is coded into a number of separate bit streams (referred to as descriptions) so that each description alone provides acceptable quality and incremental improvement can be achieved with additional descriptions. Each description is individually packetized and transmitted through separate channels or through one physical channel that is divided into several virtual channels by using appropriate time-interleaving techniques. Each description can be decoded independently to provide an acceptable level of quality. For this to be true, all the descriptions must have some basic information about the source, and therefore they are likely to be correlated. Some hybrid approaches have also been proposed recently to combine the advantages of layered coding and MDC [18,73].

Scalable coding techniques allow media servers to adapt to varying network conditions in real time. To do this, an intelligent transport mechanism is required to select the right packets (layers or descriptions) to send at a given transmission time to maximize the playback quality at the decoder. Some recent work has been focused on rate–distortion optimized scheduling algorithms for scalable video streaming [16,48]. In this case, each packet is not equally important due to different distortion contributions, playback deadlines, and packet dependencies caused by temporal prediction and layering. Runtime feedback information is employed to make the transport decisions based on the current network condition and decoder receiving status. See also Section 4.4.5.

4.4.4 Bit Stream Switching

Although scalable coding can potentially provide flexible bandwidth adaptation over unpredictable best-effort networks, current coding techniques still suffer from relatively low coding efficiency, especially when the bit rate range is large. As a result, bit stream switching techniques are widely used in many commercial video streaming systems [6,19] to create multiple versions of the same content at different bit rates and dynamically switch among them to accommodate the bandwidth variations. In this section, we introduce three major switching techniques, namely multiple bit rate coding, SP/SI pictures, and stream morphing.

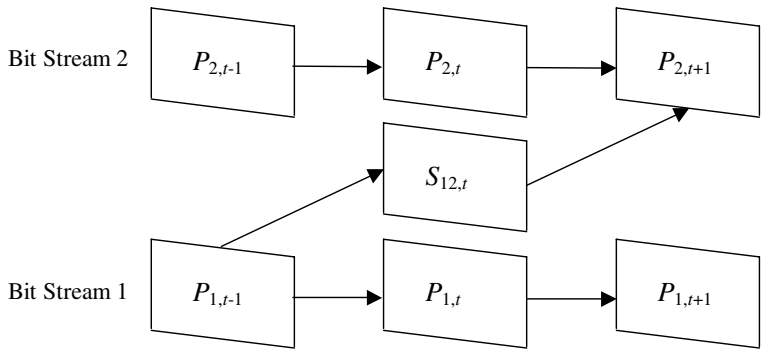
4.4.4.1 Multiple Bit Rate (Simulcast) Coding

In this approach each media source is simply compressed into multiple independent nonscalable bit streams at different bit rates and qualities. During the transmission, the server switches to a particular bit stream whose transmission yields the minimum reconstructed distortion based on the estimation of actual channel bandwidth and loss characteristics. Ideally, once a change in network bandwidth is detected, the server will immediately switch to a more appropriate stream to reflect the change promptly. However, because of motion prediction, switching between bit streams at arbitrary locations, such as a P-frame, may introduce severe drift effects since the reference frames are different at the encoder and decoder.

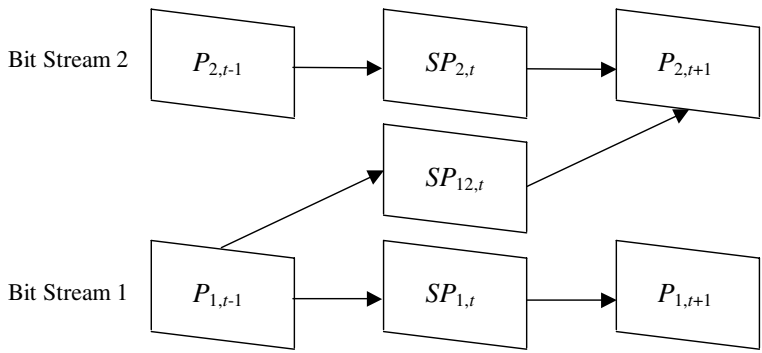
The simplest way to achieve a drift-free switching is to insert I-frames periodically in each stream and let the switching from stream to stream occur only at those I-frames. Obviously, because adaptation requests only take effect when an I-frame is reached, this increases the latency of bandwidth adaptation. To provide more flexible adaptation, the frequency of I-frames has to be increased at a cost of significantly increased bit rates to achieve the same quality. Thus, allowing more effective stream switching comes at the cost of a decrease in video quality for a given target bit rate. In addition, the flexibility of bandwidth adaptation also depends on the number of different bit streams available, each coded at a different bit rate. The more bit streams are available, the more accurate and finer level bandwidth adjustments can be supported. The inefficiency of coding I-frames results in a much larger storage requirement on the media server when the number of supported bit streams is large. The trade-off between coding efficiency and switching flexibility thus becomes a main consideration on the design of a drift-free switching approach.

More efficient approaches for drift-free switching aim at removing the overhead associated with I-frames, which exists even for normal transmission without switching between bit streams. In order to facilitate switching at inter frames (i.e., P-/B-frames), an extra bit stream is created at each predefined switching point at

an increased rate cost when switching happens, while keeping the coding efficiency for normal transmission at the same or close to the one without supporting the switching functionality. One way is to encode the difference of reference frames at the switching points and transmit this as an additional bit stream, which can be used for drift compensation at the decoder. The mismatch can be removed if lossless compression is applied. Another way is to introduce a specially encoded P-frame, called an S-frame [23], to achieve switching at the location of inter frames. As illustrated in Figure 4.6a, to initiate switching from bit stream 1 to bit stream 2 at time t, an S-frame (frame $S_{12,t}$) is encoded as a P-frame with the previously reconstructed frame at time $t-1$ in bit stream 1 (frame $P_{1,t-1}$) as the reference frame and the reconstructed frame at time t in bit stream 2 (frame



(a) S-frame



(b) SP-frame

FIGURE 4.6: Switching from bit stream 1 to bit stream 2 through specially encoded frames: (a) S-frame and (b) SP-frame.

$P_{2,t}$) as the target frame. This approach cannot completely eliminate the drift. However, by reducing the QP of the S-frame, the drift amount can be controlled and made relatively small. Another disadvantage of this approach is that the rate required for S-frames can be very large due to the small QP that is required. The SP-/SI-frames to be introduced in the next section provide an improved drift-free switching approach to the S-frames. In addition to switching between nonscalable bit streams, bit stream switching can also be performed for several independently coded scalable streams [67].

4.4.4.2 SP/SI Pictures

The extended profile of H.264/MPEG-4 part 10 AVC [2] introduces two new frame types referred to as SP-frames and SI-frames [33]. SP- and SI-frames facilitate switching between multiple independently coded bit streams and also provide "VCR-like" functionalities, such as random access, fast forward, fast backward, and so on.

Within each encoded bit stream, SP-frames are created at the switching points in two different types, namely primary SP-frame and secondary SP-frame (see Figure 4.6b). The primary SP-frame (frames $SP_{1,t}$ and $SP_{2,t}$ in Figure 4.6b) is created by motion-compensated prediction from the previously reconstructed frames in the same bit stream, while the corresponding secondary SP-frame (frame $SP_{12,t}$ as an example) is generated, with identical reconstructed values as the primary SP-frame (frame $SP_{2,t}$), by using the previously reconstructed values from another bit stream. A primary SP-frame is encoded with almost the same coding efficiency as the corresponding P-frame. The difference between SP- and P-/S-frames lies in that, due to the special encoding of the secondary SP-frame, the pair of SP-frames can be identically reconstructed even if they are predicted using different frames. Compared to I-frames, SP-frames can achieve same switching functionality with significantly fewer bits by exploiting motion-compensated predictive coding. An alternative to a secondary SP-frame is an SI-frame, using only intra prediction to produce identical reconstructed values as the corresponding primary SP-frame. It is mainly used when motion prediction is not efficient, such as switching between bit streams representing completely different video sequences, or for random access in which decoding of the current frame does not depend on any previous frames.

4.4.4.3 Stream Morphing

Stream morphing [44] has been introduced as an interesting alternative to scalable video coding and is related to techniques that have been proposed for efficient scalable DPCM coding [58,60,64]. Scalable coding schemes operate in the signal domain to separate an input into different layers. For example, in a closed

loop system, the video sequence obtained from reconstructing the base layer is subtracted from the original video sequence, which is in turn compressed. Alternatively, an open loop system (e.g., one based on wavelet transforms) would directly separate the input sequence into "components" (e.g., subbands), compress these separately, and form the layers by grouping various of these components.

Stream morphing is based on the following observation. Consider a video sequence encoded with a nonscalable codec (say MPEG-2) at two different target rates. Clearly there will be some redundancy between the two bit streams since they represent the same sequence, albeit at different rates. For example, most blocks will have the same motion vectors at both rates, large DCT coefficients in the residual signal will tend to be in the same locations, etc. A stream morphing technique would use the low rate stream as the base layer. Then the enhancement layer will contain a bit stream with a special syntax that allows the decoder to reconstruct the high rate *bit stream* from the low rate *bit stream*. For example, this enhancement layer could include differential information with respect to the motion vectors included in the base layer. Note that this is a transformation between bit streams. Thus one of the principal differences between stream morphing and standard scalability tools is that *decoding the base layer is not needed to reproduce the signal at the highest quality*. Instead, the base layer bit stream is "morphed" into the high-resolution bit stream, on which a standard decoder is used (e.g., the MPEG-2 decoder in our example). Note also that the quality levels at the decoder are exactly determined by the two (or more) originally encoded versions.

4.4.5 Overview of Optimization Techniques for Bandwidth Adaptation

Recall that bandwidth adaptation requires (i) observing the state of the network, (ii) estimating or observing the state of the decoder, and then (iii) based on bandwidth availability and decoder state, deciding what information should be sent next to the decoder. In this section we discuss briefly this decision process. Our focus here is in highlighting the challenges involved and how these have to be addressed by proposed techniques.

Ideally the goal in deciding what information is sent to the decoder should be to maximize the expected quality at the decoder. Note that we consider expected quality because there is uncertainty about the actual quality available at the decoder; changes in bandwidth, packet losses, and so forth will affect the resulting quality.

To facilitate the discussion, in what follows we assume that information available for transmission has already been packetized. The role of the decision mechanisms under consideration is essentially to prioritize the transmission so that most "important" information is sent first.

Optimization of expected quality at the decoder is complex because of multiple factors:

- The expected distortion is hard to estimate.
- The candidate packets may depend on each other.
- At any given time there are many candidate packets.

Estimating the expected distortion at the transmitter requires first determining both the current "state" of the transmission channel and its expected behavior in the near future. Various types of channel models are considered in Chapters 7 and 11. The type of channel models available, for example, with memory [29] or without it [16,48], depends on the systems being considered. Observations may include packet receipt feedback, received power measurements, etc. While the accuracy of the models may be questionable, it is also likely that even an inaccurate model will provide enough information to improve on a system that makes no assumptions about the transmission channel.

In addition, estimations of expected distortion are based on the reconstruction quality achievable when different sets of packets are received. In cases where pre-encoded data is being transmitted it is possible, in theory, to quantify exactly achievable distortion in each scenario. In practice, however, techniques that require less computation and provide estimates of expected distortion may be preferable. For example, some methods may attach some importance to each packet, where the importance is based on some simplifications about the decoding process (e.g., frames that depend on frames received in error are not decoded, no error concealment is applied); see, for example, [16,48]. Then optimization techniques would seek to maximize the expected "importance" of packets received.

Most widely used video coding techniques make use of prediction across frames. This complicates distortion estimation, since a packet loss may affect multiple future frames. A very powerful technique used to capture the dependencies is that formalized by Chou and Miao [16], which leads to the creation of a directed acyclic graph to represent all the packets being transmitted. With this type of technique it is possible to attach more importance to packets from which multiple other packets depend. As we had indicated earlier for the channel model, even a rough model of these dependencies (which may not provide exact distortion values) is likely to provide better results than techniques that completely ignore the existence of these dependencies.

Optimization complexity should definitely be of concern. As has been demonstrated by various authors (see [9–11,14,16,17,32,48,49,73,80,81]) efficient techniques can be developed once knowledge of the structure of the media stream (including dependencies) and an estimate of the channel state are available. This can be done by estimating the expected distortions if several different candidate packets (not necessarily all available ones) were transmitted. This distortion can be estimated for one decision (the next packet to be transmitted) or more than one.

After this evaluation, the packet leading to a lower expected distortion is chosen, and this decision process is repeated for the next packet.

4.5 SUMMARY AND FURTHER READING

The heterogeneous and time-variant nature of today's networks imposes a number of challenges for real-time video communication. In this chapter, we have discussed alternative techniques for bandwidth adaptation and their relative merits. The main points made in this chapter are summarized as follows.

- We classify bandwidth adaptation architectures based on three basic design decisions, namely selection of adaptation points, decision agents, and source coding techniques. Bandwidth adaptation is made based on available source and channel information. The source-related information is known more accurately at the sender, while channel information is more accurate at the client. A proxy, located in the middle of the network, can achieve a good compromise between server and client adaptation.
- When the sender acts as the adaptation point, the highest degree of flexibility is possible in terms of source coding, which facilitates achieving finer granularity rate adaptation, reducing the quality penalty at the receiver. However, this may lead to a longer reaction time if network state information is provided by the receiver. Adaptation decisions may be inefficient if, instead, the sender itself has to estimate the state of the network without waiting for receiver feedback. Adaptation at the sender makes scaling to a large number of receivers more difficult, as it increases the computation load at the sender. Adaptation at the client can reduce decoding complexity, but will have no impact on the network traffic.
- If the sender is the decision agent, it will have access to more accurate source information, but may not have reliable or timely information about the network state near the receiver. This approach helps improve overall bandwidth utilization when multiple receivers are served by the sender. In contrast, if the client acts as the decision agent, there is potential for better adaptation decisions given the higher accuracy network and packet arrival information. However, when decisions made by the receiver have to be put in place by the sender, the latency involved can lead to lower adaptation efficiency.
- Rate control techniques are used during the encoding process to adjust coding parameters to meet a target encoding rate. Transcoding techniques, often used at either the server or the proxy, take a compressed media stream as an input and convert it to another compressed stream. Scalable coding provides flexible bandwidth adaptation over a given bit rate range rather than at a fixed bit rate. Different from the aforementioned techniques, bit

stream switching techniques encode the same media content into multiple versions at different bit rates and dynamically switch among them to accommodate the bandwidth variations. In this chapter we have discussed several switching techniques: multiple bit rate coding, SP/SI pictures, and stream morphing. The trade-off between coding efficiency (to reduce overhead) and switching flexibility is a main consideration on the design of various switching techniques.

Further details on many of the bandwidth adaptation techniques described in this chapter can be found in other literature, as well as in other chapters in this book. For example, Ortega and Ramchandran [53] and Sullivan and Wiegand [65] discuss rate–distortion optimization for image and video compression; Vetro *et al.* [72] and Xin *et al.* [78] provide overviews of transcoding; and Goyal [25] and Wang *et al.* [74] review state-of-the-art multiple description coding. For more details on rate–distortion-optimized streaming, the article by Chou and Miao [16] can serve as a starting point. Although this chapter focused on the fundamentals of bandwidth adaptation on a simple client–server system, there is considerable interest in more complex systems with multiple paths used for media transport, such as content delivery networks and P2P networks. The interested reader is referred to the work of Apostolopoulos *et al.* [4], Padmanabhan *et al.* [54], and Rejaie and Ortega [57].

REFERENCES

[1] ISO/IEC 13818-2. *Generic coding of moving pictures and associated audio, part-2 video*. November 1994.

[2] ISO/IEC 14496-10 and ITU-T Rec. H.264. *Advanced video coding*. 2003.

[3] ISO/IEC 14496-2/FPDAM4. *Coding of audio-visual objects, part-2 visual, amendment 4: streaming video profile*. July 2000.

[4] J. Apostolopoulos, T. Wong, W.-T. Tan, and S. Wee. On multiple description streaming with content delivery networks. In *Proc. Conf. Computer Communications (INFOCOM)*, June 2002.

[5] J. F. Arnold, M. R. Frater, and Y. Wang. Efficient drift-free signal-to-noise ratio scalability. *IEEE Trans. Circuits and Systems for Video Technology*, 10(1):70–82, February 2000.

[6] B. Birney. Intelligent streaming. http://www.microsoft.com/windows/windowsmedia/howto/articles/intstreaming.aspx, May 2003.

[7] P. Assunção and M. Ghanbari. Post-processing of MPEG2 coded video for transmission at lower bit rates. In *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing*, volume 3, pages 1998–2001, May 1996.

[8] P. Assunção and M. Ghanbari. A frequency-domain video transcoder for dynamic bit-rate reduction of MPEG-2 bit streams. *IEEE Trans. Circuits and Systems for Video Technology*, 8(8):953–967, December 1998.

[9] J. Chakareski, J. Apostolopoulos, S. Wee, W. Tan, and B. Girod. Rate-distortion hint tracks for adaptive video streaming. *IEEE Trans. Circuits and Systems for Video Technology*, 15(10):1257–1269, October 2005.

[10] J. Chakareski, P. A. Chou, and B. Girod. Rate-distortion optimized streaming from the edge of the network. In *Proc. Workshop on Multimedia Signal Processing*, December 2002.

[11] J. Chakareski and B. Girod. Rate-distortion optimized packet scheduling and routing for media streaming with path diversity. In *Proc. Data Compression Conference*, March 2003.

[12] S.-F. Chang and D. G. Messerschmitt. Manipulation and compositing of MC-DCT compressed video. *IEEE J. Selected Areas in Communications*, 13(1):1–11, January 1995.

[13] M. C. Chen and A. N. Willson. Rate-distortion optimal motion estimation algorithms for motion-compensated transform video coding. *IEEE Trans. Circuits and Systems for Video Technology*, 8(2):147–158, April 1998.

[14] G. Cheung and W. Tan. Directed acyclic graph based source modeling for data unit selection of streaming media over QoS networks. In *Proc. Int'l Conf. Multimedia and Exhibition*, August 2002.

[15] P. A. Chou, A. E. Mohr, A. Wang, and S. Mehrotra. Error control for receiver-driven layered multicast of audio and video. *IEEE Trans. Multimedia*, 3(1):108–122, March 2001.

[16] P. A. Chou and Z. Miao. Rate-distortion optimized streaming of packetized media. *IEEE Trans. Multimedia*, 8(2):390–404, April 2006.

[17] P. A. Chou and A. Sehgal. Rate-distortion optimized receiver-driven streaming over best-effort networks. In *Proc. Int'l Packet Video Workshop*, volume 1, April 2002.

[18] P. A. Chou, H. J. Wang, and V. N. Padmanabhan. Layered multiple description coding. In *Proc. Int'l Packet Video Workshop*, volume 1, April 2003.

[19] G. J. Conklin, G. S. Greenbaum, K. O. Lillevold, A. F. Lippman, and Y. A. Reznik. Video coding for streaming media delivery on the internet. *IEEE Trans. Circuits and Systems for Video Technology*, 11(3):269–281, March 2001.

[20] W. Ding and B. Liu. Rate control of MPEG video coding and recording by rate-quantization modeling. *IEEE Trans. Circuits and Systems for Video Technology*, 6(1):12–20, February 1996.

[21] M. Domanski, A. Luczak, and S. Mackowiak. Spatio-temporal scalability for MPEG video coding. *IEEE Trans. Circuits and Systems for Video Technology*, 10(7):1088–1093, October 2000.

[22] A. Eleftheriadis and D. Anastassiou. Constrained and general dynamic rate shaping of compressed digital video. In *Proc. Int'l Conf. Image Processing*, volume 3, pages 396–399, October 1995.

[23] N. Farber and B. Girod. Robust H.263 compatible video transmission for mobile access to video servers. In *Proc. Int'l Conf. Image Processing*, volume 2, pages 73–76, October 1997.

[24] B. Girod. Rate-constrained motion estimation. In *Proc. Visual Communications and Image Processing*, pages 1026–1034, September 1994.

[25] V. K. Goyal. Multiple description coding: Compression meets the network. *IEEE Signal Processing Magazine*, 18(5):74–93, September 2001.

[26] Z. Guo, O. C. Au, and K. B. Letaief. Parameter estimation for image/video transcoding. In *Proc. Int'l Symp. Circuits and Systems*, volume 2, pages 269–272, May 2000.

[27] H.-M. Hang and J.-J. Chen. Source model for transform video coder and its application. *IEEE Trans. Circuits and Systems for Video Technology*, 7(2):287–311, April 1997.

[28] Z. He and S. K. Mitra. From rate–distortion analysis to resource-distortion analysis. *IEEE Circuits and Systems Magazine*, 5(3):6–18, 2005.

[29] C.-Y. Hsu, A. Ortega, and M. Khansari. Rate control for robust video transmission over burst-error wireless channels. *IEEE J. Selected Areas in Communications*, 17(5):1–18, May 1999.

[30] C.-Y. Hsu, A. Ortega, and A. Reibman. Joint selection of source and channel rate for VBR video transmission under ATM policing constraints. *IEEE J. Selected Areas in Communications*, 15(6):1016–1028, August 1997.

[31] H.-C. Huang, C.-N. Wang, and T. Chiang. A robust fine granularity scalability using trellis-based predictive leak. *IEEE Trans. Circuits and Systems for Video Technology*, 12(6):372–385, June 2002.

[32] M. Kalman and B. Girod. Rate-distortion optimized streaming of video with multiple independent encodings. In *Proc. Int'l Conf. Image Processing*, volume 1, October 2004.

[33] M. Karczewicz and R. Kurceren. The SP- and SI-frames design for H.264/AVC. *IEEE Trans. Circuits and Systems for Video Technology*, 13(7):637–644, July 2003.

[34] S. A. Karunasekera and N. G. Kingsbury. A distortion measure for image artifacts based on human visual sensitivity. In *Proc. Int'l Conf. Image Processing*, April 1994.

[35] K. Lengwehasatit and A. Ortega. Probabilistic partial distance fast matching for motion estimation. *IEEE Trans. Circuits and Systems for Video Technology*, 11(2):139–152, February 2001.

[36] K. Lengwehasatit and A. Ortega. Scalable variable complexity approximate forward DCT. *IEEE Trans. Circuits and Systems for Video Technology*, 14(11):1236–1248, November 2004.

[37] A. Leontaris and A. R. Reibman. Comparison of blocking and blurring metrics for video compression. In *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing*, March 2005.

[38] W. Li. Overview of fine granularity scalalability in MPEG-4 video standard. *IEEE Trans. Circuits and Systems for Video Technology*, 11(3):301–317, March 2001.

[39] Y. J. Liang, N. Farber, and B. Girod. Adaptive playout scheduling and loss concealment for voice communication over IP networks. *IEEE Transactions on Multimedia*, 5(4), December 2003.

[40] Yi J. Liang and B. Girod. Prescient R-D optimized packet dependency management for low-latency video streaming. In *Proc. Int'l Conf. Image Processing*, September 2003.

[41] Yi J. Liang, E. Steinbach, and B. Girod. Multi-stream voice transmission over the internet using path diversity. In *Proc. ACM Multimedia*, September 2001.

[42] C.-W. Lin and Y.-R. Lee. Fast algorithms for DCT-domain video transcoding. In *Proc. Int'l Conf. Image Processing*, volume 1, pages 421–424, October 2001.

[43] L.-J. Lin and A. Ortega. Bit-rate control using piecewise approximated rate–distortion characteristics. *IEEE Trans. Circuits and Systems for Video Technology*, 8(4):446–459, August 1998.

[44] J. Macnicol, J. Arnold, and M. Frater. Scalable video coding by stream morphing. *IEEE Trans. Circuits and Systems for Video Technology*, 15(2):306–319, February 2005.

[45] N. Magharei and R. Rejaie. Adaptive receiver-driven streaming from multiple senders. *Proceedings of ACM Multimedia Systems Journal, Springer-Verlag*, 11(6):1–18, April 2006.

[46] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *ACM SIGCOMM*, August 1996.

[47] N. Merhav. Multiplication-free approximate algorithms for compressed-domain linear operations on images. *IEEE Trans. Image Processing*, 8(2):247–254, February 1999.

[48] Z. Miao and A. Ortega. Expected run-time distortion based scheduling for delivery of scalable media. In *Proc. Int'l Packet Video Workshop*, volume 1, April 2002.

[49] Z. Miao and A. Ortega. Fast adaptive media scheduling based on expected run-time distortion. In *Proc. Asilomar Conf. Signals, Systems, and Computers*, volume 1, November 2002.

[50] Z. Miao and A. Ortega. Scalable proxy caching of video under storage constraints. *IEEE J. Selected Areas in Communications*, 20(7):1315–1327, September 2002.

[51] Y. Nakajima, H. Hori, and T. Kanoh. Rate conversion of MPEG coded video by requantization process. In *Proc. Int'l Conf. Image Processing*, volume 3, pages 408–411, October 1995.

[52] T. Nguyen and A. Zakhor. Multiple sender distributed video streaming. *IEEE Trans. Multimedia*, 6(2):315–326, April 2004.

[53] A. Ortega and K. Ramchandran. Rate-distortion methods for image and video compression. *IEEE Signal Processing Magazine*, 15(6):23–50, November 1998.

[54] V. N. Padmanabhan, H. J. Wang, and P. A. Chou. Resilient peer-to-peer streaming. In *Proc. Int'l Conf. Network Protocols*, November 2003.

[55] W. Pan and A. Ortega. Complexity-scalable transform coding using variable complexity algorithms. In *Proc. Data Compression Conference*, pages 263–272, March 2000.

[56] A. R. Reibman and M. T. Sun, editors. *Compressed Video over Networks*. "Variable bit rate video coding." Marcel Dekker, New York (NY), 2001.

[57] R. Rejaie and A. Ortega. Pals: Peer-to-peer adaptive layered streaming. In *Proc. Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, June 2003.

[58] K. Rose and S. Regunathan. Toward optimality in scalable predictive coding. *IEEE Transactions on Image Processing*, 10:965–976, July 2001.

[59] B. Shen, S.-J. Lee, and S. Basu. Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks. *IEEE Trans. Multimedia*, 6(2):375–386, April 2004.

[60] R. Singh and A. Ortega. Erasure recovery in predictive coding environments using multiple description coding. In *IEEE Workshop on Multimedia Signal Processing*, 1999.

[61] I. Sodagar, H.-J. Lee, P. Hatrack, and Y.-Q. Zhang. Scalable wavelet coding for synthetic/natural hybrid images. *IEEE Trans. Circuits and Systems for Video Technology*, 9(2):244–254, March 1999.

[62] J. Song and B.-L. Yeo. A fast algorithm for DCT-domain inverse motion compensation based on shared information in a macroblock. *IEEE Trans. Circuits and Systems for Video Technology*, 10(5):767–775, August 2000.

[63] H. Sorial, W. E. Lynch, and A. Vincent. Selective requantization for transcoding of MPEG compressed video. In *Proc. Int'l Conf. Multimedia and Exhibition*, volume 1, pages 217–220, August 2000.

[64] N. Srinivasamurthy, A. Ortega, and S. Narayanan. Efficient scalable encoding for distributed speech recognition. *Speech Communication*, 48:888–902, 2006.

[65] G. J. Sullivan and T. Wiegand. Rate-distortion optimization for video compression. *IEEE Signal Processing Magazine*, 15(6):74–90, November 1998.

[66] H. Sun, W. Kwok, and J. W. Zdepski. Architectures for MPEG compressed bitstream scaling. *IEEE Trans. Circuits and Systems for Video Technology*, 6(2):191–199, April 1996.

[67] X. Sun, F. Wu, S. Li, W. Gao, and Y.-Q. Zhang. Seamless switching of scalable video bitstreams for efficient streaming. *IEEE Trans. Multimedia*, 6(2):291–303, April 2004.

[68] K. T. Tan and M. Ghanbari. A multi-metric objective picture-quality measurement model for MPEG video. *IEEE Trans. Circuits and Systems for Video Technology*, 10(7):1208–1213, October 2000.

[69] M. van der Schaar and Y. Andreopoulos. Rate-distortion-complexity modeling for network and receiver aware adaptation. *IEEE Trans. Multimedia*, 7(3):471–479, June 2005.

[70] M. van der Schaar and P. H. N. de With. Near-lossless complexity-scalable embedded compression algorithm for cost reduction in DTV receivers. *IEEE Trans. on Consumer Electronics*, 46(4):923–933, November 2000.

[71] M. van der Schaar and H. Radha. Adaptive motion-compensation fine-granular-scalability (AMC-FGS) for wireless video. *IEEE Trans. Circuits and Systems for Video Technology*, 12(6):360–371, June 2002.

[72] A. Vetro, C. Christopoulos, and H. Sun. Video transcoding achitectures and techniques: an overview. *IEEE Signal Processing Magazine*, 20(2):18–29, March 2003.

[73] H. Wang and A. Ortega. Robust video communication by combining scalability and multiple description coding techniques. In *Proc. Symp. Electronic Imaging*, volume 1, January 2003.

[74] Y. Wang, A. R. Reibman, and S. Lin. Multiple description coding for video delivery. *Proceedings of the IEEE*, 93(1):57–70, January 2005.

[75] O. Werner. Requantization for transcoding of MPEG-2 intraframes. *IEEE Trans. Image Processing*, 8(2):179–191, February 1999.

[76] T. Wiegand, M. Lightstone, D. Mukherjee, T. G. Campbell, and S. K. Mitra. Rate-distortion optimized mode selection for very low bit rate video coding and the emerging H.263 standard. *IEEE Trans. Circuits and Systems for Video Technology*, 6(2):182–190, April 1996.

[77] F. Wu, S. Li, and Y.-Q. Zhang. A framework for efficient progressive fine granularity scalable video coding. *IEEE Trans. Circuits and Systems for Video Technology*, 11(3):332–344, March 2001.

[78] J. Xin, C.-W. Lin, and M.-T. Sun. Digital video transcoding. *Proceedings of the IEEE*, 93(1):84–97, January 2005.

[79] J. Youn, J. Xin, and M.-T. Sun. Fast video transcoding architectures for networked multimedia applications. In *Proc. Int'l Symp. Circuits and Systems*, volume 4, pages 25–28, May 2000.

[80] R. Zhang, S. Regunathan, and K. Rose. Optimized video streaming over lossy networks with real-time estimation of end-to-end distortion. In *Proc. Int'l Conf. Multimedia and Exhibition*, volume 1, August 2002.

[81] R. Zhang, S. L. Regunathan, and K. Rose. Video coding with optimal inter/intra-mode switching for packet loss resilience. *IEEE J. Selected Areas in Communications*, 18(6):966–976, June 2000.

5

Scalable Video Coding for Adaptive Streaming Applications

Béatrice Pesquet-Popescu, Shipeng Li, and Mihaela van der Schaar

5.1 INTRODUCTION

The transmission of multimedia content over IP networks such as the Internet and wireless networks has been growing steadily over the past few years. Moreover, multimedia streaming and the set of applications that rely on streaming are expected to continue growing. Meanwhile, the current quality of streaming multimedia, in general, and video, in particular, can still be greatly improved. To achieve a higher level of quality and further proliferation of IP video, many technical challenges have to be addressed in the two areas of video coding and networking (streaming). A framework that addresses both the video coding and the networking challenges associated with IP-based video streaming is *scalability*. From a video coding point of view, scalability plays a crucial role in delivering the best possible video quality over unpredictable, "best-effort" networks. Bandwidth variation is one of the primary characteristics of "best-effort" networks, and current IP networks are a prime example of such networks. Video scalability enables an application to adapt the streamed video quality to changing network conditions (and specifically to bandwidth variation) and device complexities [40]. From a networking point of view, scalability is needed to enable a large number of users to view any desired video stream, at anytime, and from anywhere.

Scalable techniques try to avoid simulcast solutions, in which several encoders run in parallel. Simulcast solutions usually require knowledge of the network and decoder capabilities in advance in order to select the best encodings. To avoid network overload, the number of bit streams that can be simultaneously multiplexed

is limited. Even though point-to-multipoint connections are enabled by the simulcast solution, there is a clear loss in efficiency.

Consequently, any scalable video coding solution has to enable a very simple and flexible streaming framework, and hence, it must meet the following requirements.

1. The solution must enable a streaming server to perform only minimal real-time processing and rate control while outputting a very large number of simultaneous unicast (on-demand) streams.
2. The scalable video coding approach over IP networks has to be highly adaptable to unpredictable bandwidth variations due to heterogeneous access technologies of the receivers (e.g., analog modem, cable mode, xDSL, wireless mobile, and wireless LANs) or due to dynamic changes in network conditions (e.g., congestion events).
3. The video coding solution must enable low-complexity decoding and low-memory requirements to provide common receivers (e.g., set top boxes and digital televisions), in addition to powerful computers, the opportunity to stream and decode any desired Internet video content.
4. The streaming framework and related scalable video coding approach should be able to support both multicast and unicast applications. This, in general, eliminates the need for coding content in different formats to serve different types of applications.
5. The scalable bit stream must be resilient to packet loss events, which are quite common over IP networks.

These requirements are the primary drivers behind the design of the existing and emerging scalable video coding schemes.

5.2 SCALABILITY MODES IN CURRENT VIDEO CODING STANDARDS

5.2.1 Spatial, Temporal, and SNR Coding Structures

There are three basic types of scalability in scalable video coding: spatial, temporal, and quality (or SNR) scalabilities. In a spatial scalable scheme, full decoding leads to high spatial resolution, while partial decoding leads to reduced spatial resolutions (reduction of the format). In a temporal scalable scheme, partial decoding provides lower decoded frame rates (temporal resolutions). In an SNR scalable scheme, temporal and spatial resolutions are kept the same, but the video quality (SNR) varies depending on how much of the bit stream is decoded.

Current standards, such as H.263, H.264, MPEG-2, and MPEG-4 (both part 2 and part 10), are based on a predictive video coding scheme (see Figure 5.1).

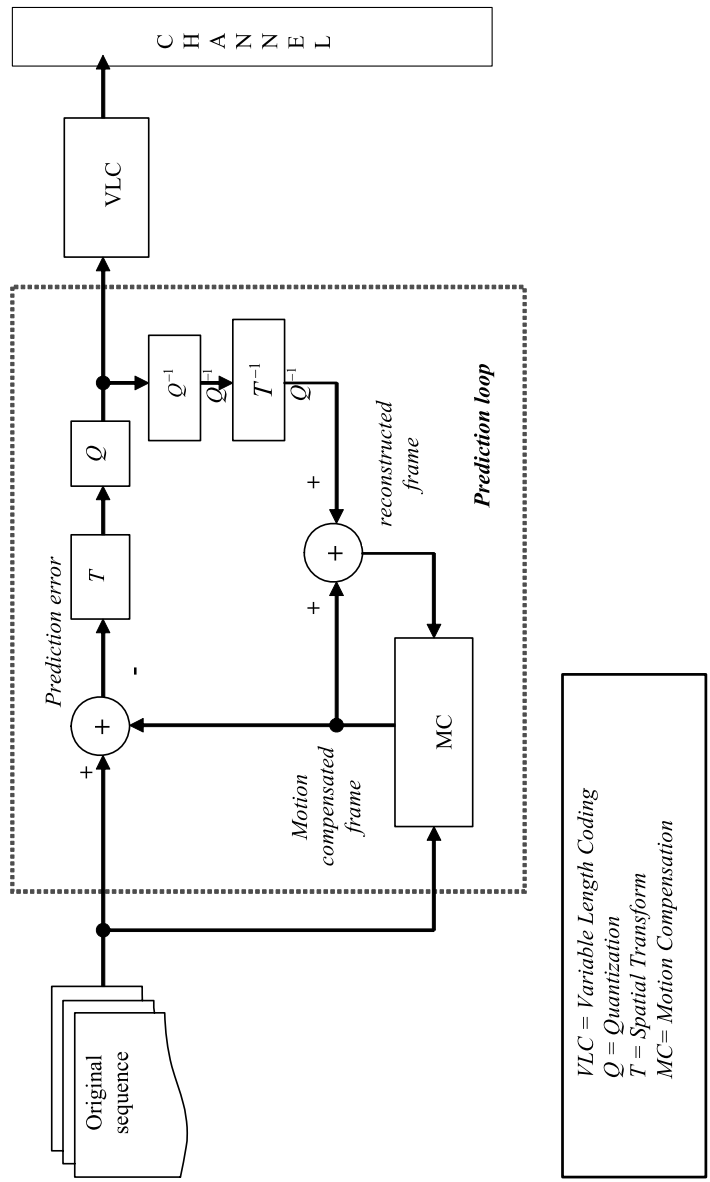


FIGURE 5.1: Predictive (hybrid) video coding scheme.

Although they were not initially designed to address these issues, current standards tried to upgrade their video coding schemes in order to include scalability functionalities. However, this integration generally came at the expense of coding efficiency (performance).

In a standard environment, scalability is achieved through *a layered structure*, where the encoded video information is divided into two or more separated bit streams corresponding to the different layers (see Figure 5.2).

- The base layer (BL) is generally highly and efficiently compressed by a nonscalable standard solution.
- The enhancement layer(s) (EL) encode(s) the residual signal to produce the expected scalability (it delivers, when combined with the base layer decoding, a progressive quality improvement in case of SNR scalability, a higher spatial resolution for spatial scalability, and a higher frame rate for temporal scalability).

To achieve *spatial scalability* in the hybrid scheme presented in Figure 5.3, the input video sequence is first spatially decimated to yield the lowest resolution layer, which is encoded by a standard encoder. A similar coding scheme is employed for the enhancement layer. To transmit a higher resolution version of the current frame, two predictions are formed: one is obtained by spatially interpolating the decoded lower resolution image of the current frame (spatial prediction) and the other by temporally compensating the higher resolution image of the predicted frame with motion information (temporal prediction). The two predictions are then adaptively combined for a better prediction and the residue after prediction is coded and transmitted. In Figure 5.3, a scheme with two resolution levels is depicted, but the same principle can be used to produce several spatial resolution enhancement levels. This solution corresponds to a Laplacian pyramid and is noncritically sampled, or redundant (the number of output samples is higher than the number of input samples).

The drawback of this approach is that the different encoding loops with their own motion estimation steps are used in parallel, at the encoder side, and several motion compensation loops are necessary at the decoder side, thus increasing the computational complexity both at the encoder and at the decoder. A possible advantage of this scheme is the flexibility in choosing the downsampling/upsampling filters, in particular for reducing aliasing at lower resolutions.

Related to the spatial scalability, there is the issue of *motion vector scalability*. Indeed, the different resolution levels will need motion vector fields with different resolutions and, possibly, accuracies. For the aforementioned Laplacian pyramid coding, the simplest approach is to estimate and encode the motion vectors, starting from the lowest resolution and going to the highest. From one layer to the other, the motion vector size needs to be doubled. Additionally, a refinement of the motion vector can be performed at higher resolutions. At this point, the pre-

FIGURE 5.2: Global structure of a layered scalable video-coding scheme.

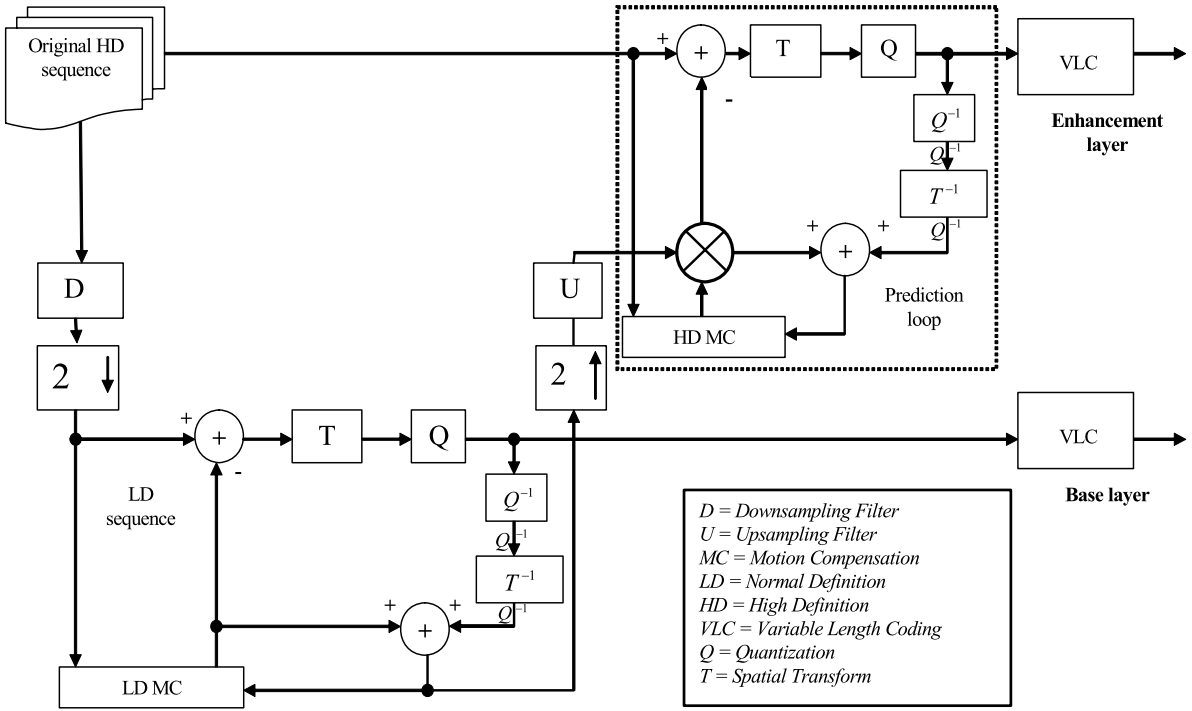


FIGURE 5.3: Layered spatial scalability.

cision and the accuracy of the motion can also be increased at higher levels. By precision we understand here the size of the block considered for motion estimation and compensation. When doubling the resolution, the dimensions of the block also double, and the motion representation loses in precision. Therefore, it may be convenient to split the block in smaller subblocks (two rectangular or four square ones) and look for refinement vectors in the subblocks. The decision to split or keep the lower resolution precision may be taken based on a rate–distortion criterion. Once the lowest resolution motion vector field is encoded, the next levels can be either encoded independently, with a possible loss in efficiency, or only the refinement vector(s) can be encoded in the refinement layer. The interested reader is referred to [22] for a more detailed discussion on motion vector scalability and its impact on the prediction complexity.

Temporal scalability involves partitioning of the group of pictures (GOP) into layers having the same spatial resolution. A simple way to achieve temporal scalability is to put some of the B frames from an IBBP ... stream into one or several enhancement layers. This solution comes at no cost in terms of coding efficiency. In a more general setting, the base layer may contain I, P, and B frames at the low frame rate, while the enhancement layers can only use frames from the immediately lower temporal layer and previous frames from the same enhancement layer for temporal prediction. Generally, temporal prediction from future frames in the same enhancement layer is prohibited in order to avoid reordering in the enhancement layers. An example with one enhancement layer is presented in Figure 5.4.

The layered solution can be seen as an upgrade of standard solutions in order to provide scalability. The main shortcoming of these schemes comes from the fact that the information redundancy between the different layers cannot be fully exploited. This functionality is thus achieved at the expense of implementation complexity and coding efficiency.

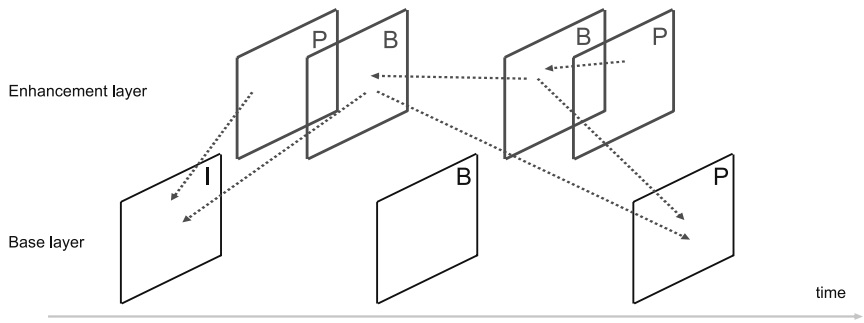


FIGURE 5.4: General framework for layered temporal scalability.

A general problem with introducing scalability in a predictive video coding scheme is the so-called *drift effect*. It occurs when the reference frame used for motion compensation in the encoding loop is not available or not completely available at the decoder side. Therefore both the encoder and the decoder have to maintain their synchronization on the same bit rate in the case of SNR scalability, resolution level for spatial scalability, and frame rate in the case of temporal scalability.

For *SNR scalability*, a layered encoder exploits correlations across subflows to achieve better overall compression: the input sequence is compressed into a number of discrete layers arranged in a hierarchy that provides progressive refinement. A strategy often used in the scalable extensions of current standards (i.e., in MPEG-2 and H263) is to encode the base layer using a large quantization step, whereas the enhancement layers have a refinement goal and use finer quantizers to encode the base layer coding error. This solution is illustrated in Figure 5.5 and is discussed in more detail later.

5.2.2 Successive Approximation Quantization and Bit Planes

To realize the SNR scalability concept discussed earlier, an important category of embedded scalar quantizers is the family of embedded dead zone scalar quantizers. For this family, each transform coefficient x is quantized to an integer

$$i_b = Q_b(x) = \begin{cases} \text{sign}(x) \cdot \left\lfloor \dfrac{|x|}{2^b \Delta} + \dfrac{\xi}{2^b} \right\rfloor, & \text{if } \dfrac{|x|}{2^b \Delta} + \dfrac{\xi}{2^b} > 0, \\ 0, & \text{otherwise,} \end{cases}$$

where $\lfloor a \rfloor$ denotes the integer part of a; $\xi < 1$ determines the width of the dead zone; $\Delta > 0$ is the basic quantization step size (basic partition cell size) of the quantizer family; and $b \in \mathbb{Z}_+$ indicates the quantizer level (granularity), with higher values of b indicating coarser quantizers. In general, b is upper bounded by a value $B_{\max}$, selected to cover the dynamic range of the input signal. The reconstructed value is given by the inverse operation,

$$y_i^p = Q_b^{-1}(i_b) = \begin{cases} 0, & i_b = 0, \\ \text{sign}(i_b) \cdot \left(|i_b| - \dfrac{\xi}{2^b} + \delta \right) 2^b \Delta, & i_b \neq 0, \end{cases}$$

where $0 \leq \delta < 1$ specifies the placement of the reconstructed value y_i^b within the corresponding uncertainty interval (partition cell), defined as $C_{i_b}^b$, and i is the partition cell index, which is bounded by a predefined value for each quantizer level (i.e., $0 \leq i \leq M_b - 1$, for each b). Based on the aforementioned formulation, it is rather straightforward to show that the quantizer Q_0 has embedded within it

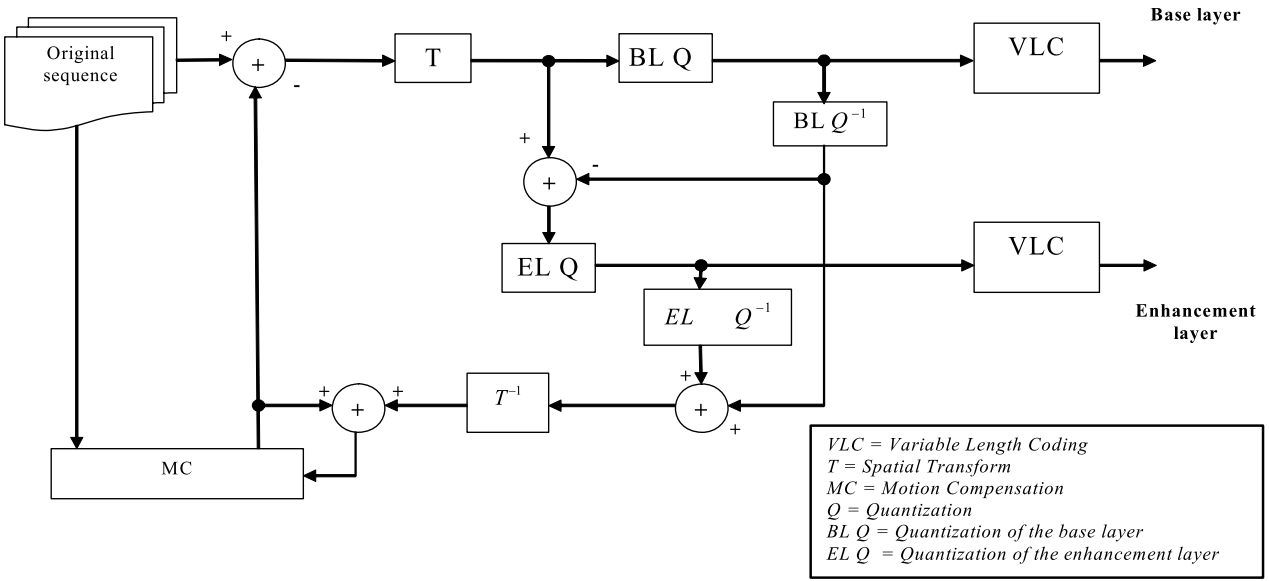


FIGURE 5.5: Layered SNR scalability.

all the uniform dead zone quantizers with step sizes $2^b\Delta$, $b \in \mathbb{Z}_+$. Moreover, it can be shown that, under the appropriate settings, the quantizer index obtained by dropping the b least-significant bits (LSBs) of i_0 is the same as that which would be obtained if the quantization was performed using a step size of $2^b\Delta$, $b \in \mathbb{Z}_+$ rather than Δ. This means that if the b LSBs of i_0 are not available, one can still dequantize at a lower level of quality using the inverse quantization formula.

The most common option for embedded scalar quantization is successive approximation quantization (SAQ). SAQ is a particular instance of the generalized family of embedded dead zone scalar quantizers defined earlier. For SAQ, $M_{B_{\max}} = M_{B_{\max}-1} = \cdots = M_0 = 2$ and $\xi = 0$, which determines a dead zone width twice as wide as the other partition cells, and $\delta = 1/2$, which implies that the output levels y_i^b are in the middle of the corresponding uncertainty intervals $C_{i_p}^b$. SAQ can be implemented via thresholding by applying a monotonically decreasing set of thresholds of the form

$$T_{b-1} = \frac{T_b}{2},$$

with $B_{\max} \geq b \geq 1$. The starting threshold $T_{B_{\max}}$ is of the form $T_{B_{\max}} = \alpha x_{\max}$, where $x_{\max}$ is the highest coefficient magnitude in the input transform decomposition, and α is a constant that is taken as $\alpha \geq 1/2$.

Let us consider the case of using a spatial transform for the compression of the frames. By using SAQ, the significance of the transform coefficients with respect to any given threshold T_b is indicated in a corresponding binary map, denoted by W^b, called the significance map. Denote by $w(\mathbf{k})$ the transform coefficient with coordinates $\mathbf{k} = (\kappa_1, \kappa_2)$ in the two-dimensional transform domain of a given input. The significance operator $s^b(\cdot)$ maps any value $x(\mathbf{k})$ in the transform domain to a corresponding binary value $w^b(\mathbf{k})$ in W^b, according to the rule

$$w^b(\mathbf{k}) = s^b(x(\mathbf{k})) = \begin{cases} 0, & \text{if } |x(\mathbf{k})| < T_b, \\ 1, & \text{if } |x(\mathbf{k})| \geq T_b. \end{cases}$$

In general, embedded coding of the input coefficients translates into coding the significance maps W^b, for every b with $B_{\max} \geq b \geq 0$.

In most state-of-the-art embedded coders, for every b this is effectively performed based on several encoding passes, which can be summarized in the following:

Nonsignificance pass: encodes $s^b(x(\mathbf{k}))$ in the list of nonsignificant coefficients (LNC). If significant, the coefficient coordinates $\mathbf{k}$ are transferred into the refinement list (RL).

Block Significance pass: For a block of coefficients with coordinates $\mathbf{k}_{\text{block}}$, this pass encodes $s^b(x(\mathbf{k}_{\text{block}}))$ and $\text{sign}(x(\mathbf{k}_{\text{block}}))$ if they have descendant blocks

(under a quad tree decomposition structure) that were not significant compared to the previous bit plane.

Coefficient Significance pass: If the coordinates of the coefficients of a significant block are not in the LNC, this pass encodes the significance of coefficients in blocks containing at least one significant coefficient. Also, the coordinates of new significant coefficients are placed into the RL. This pass also moves the coordinates of nonsignificant coefficients found in the block into the LNC for the next bit plane level(s).

Refinement pass: For each coefficient in the RL (except those newly put into the RL during the last block pass), encode the next refinement of the significance map.

5.2.3 Other Types of Scalability

In addition to the aforementioned scalabilities, other types of scalability have been proposed.

- Complexity scalability: the encoding/decoding algorithm has less complexity (CPU/memory requirements or memory access) with decreasing temporal/spatial resolution or decreasing quality [40].
- Content (or object) scalability: a hierarchy of relevant objects is defined in the video scene and a progressive bit stream is created following this importance order. Such methods of content selection may be related to arbitrary-shaped objects or even to rectangular blocks in block-based coders. The main problem of such techniques is how to automatically select and track visually important regions in video.
- Frequency scalability: this technique, popular in the context of transform coding, consists of allocating coefficients to different layers according to their frequency. Data partitioning techniques may be used to implement this functionality. The interested reader is referred to Chapter 2 of this book for more information on data partitioning.

Among existing standards, the first ones (MPEG-1 and H.261) did not provide any kind of scalability. H.263+ and H.264 provide temporal scalability through B-frames skipping.

5.3 MPEG-4 FINE GRAIN SCALABLE (FGS) CODING AND ITS NONSTANDARDIZED VARIANTS

5.3.1 SNR FGS Structure in MPEG-4

The previously discussed conventional scalable coding schemes are not able to efficiently address the problem of easy, adaptive, and efficient adaptation to time-

varying network conditions or device characteristics. The reason for this is that they provide only coarse granularity rate adaptation and their coding efficiency often decreases due to the overhead associated with an increased number of layers.

To address this problem, FGS coding has been standardized in the MPEG-4 standard, as it is able to provide fine-grain scalability to easily adapt to various time-varying network and device resource (e.g., power) constraints [6,44]. Moreover, FGS can enable a streaming server to perform *minimal real-time* processing and rate control when outputting a very large number of simultaneous unicast (on-demand) streams, as the resulting bit stream can be easily truncated to fulfill various (network) rate requirements. Also, FGS is easily adaptable to unpredictable bandwidth variations due to heterogeneous access technologies (Internet, wireless cellular or wireless LANs) or to dynamic changes in network conditions (e.g., congestion events). Moreover, FGS enables low-complexity decoding and low-memory requirements that provide common receivers (e.g., set top boxes and digital televisions), in addition to powerful computers, the opportunity to stream and decode any desired streamed video content. Hence, receiver-driven streaming solutions can only select the portions of the FGS bit stream that fulfill these constraints [40,45].

In MPEG-4 FGS, a video sequence is represented by two layers of bit streams with identical spatial resolution, which are referred to as the base layer bit stream and the fine granular enhancement layer bit stream, as illustrated in Figure 5.6.

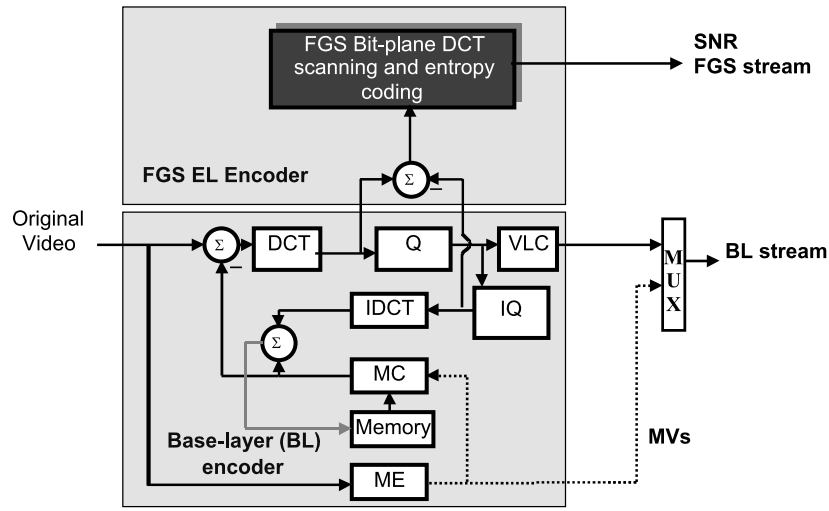


FIGURE 5.6: MPEG-4 FGS encoder.

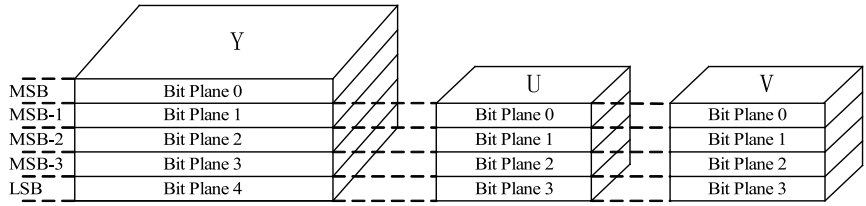


FIGURE 5.7: The structure of bit planes of Y, U, and V components.

The base layer bit stream is coded with nonscalable coding techniques, whereas the enhancement layer bit stream is generated by coding the difference between the original DCT coefficients and the reconstructed base layer coefficients using a bit-plane coding technique [1,6,7]. The residual signal is represented with bit planes in the DCT domain, where the number of bit planes is not fixed, but is based on the number of bit planes needed to represent the residual magnitude in binary format. Before a DCT residual picture is coded at the enhancement layer, the maximum number of bit planes of each color component (Y, U, and V) is first found. In general, three color components may have different numbers of bit planes. Figure 5.7 gives an example of 5 bit planes in Y component and 4 bit planes in U and V components. These three values are coded in the picture header of the enhancement layer stream and transmitted to the decoder.

All components have aligned themselves with the least significant bit (LSB) plane. The FGS encoder and decoder process bit planes from the most significant bit (MSB) plane to the LSB plane. Because of the possible different maximum numbers of bit planes on Y, U, and V components, the first MSB planes may contain only one or two components. In the example given by Figure 5.7, there is only Y component existing in the MSB plane. In this case, bits for the coded block pattern (CBP) of each macroblock can be reduced significantly. Every macroblock in a bit plane is coded with row scan order.

Since the enhancement layer bit stream can be truncated arbitrarily in any frame (see Figure 5.8), MPEG-4 FGS provides the capability of easily adapting to channel bandwidth variations.

5.3.2 MPEG-4 Hybrid Temporal–SNR Scalability with an All-FGS Structure

As mentioned earlier, temporal scalability is an important tool for enhancing the motion smoothness of compressed video. Typically, a base layer stream coded with a frame rate f_{BL} is enhanced by another layer consisting of video frames that do not coincide (temporally) with the base layer frames. Therefore, if the enhancement layer has a frame rate of f_{EL}, then the total frame of both base and enhancement layer streams is $f_{BL} + f_{EL}$.

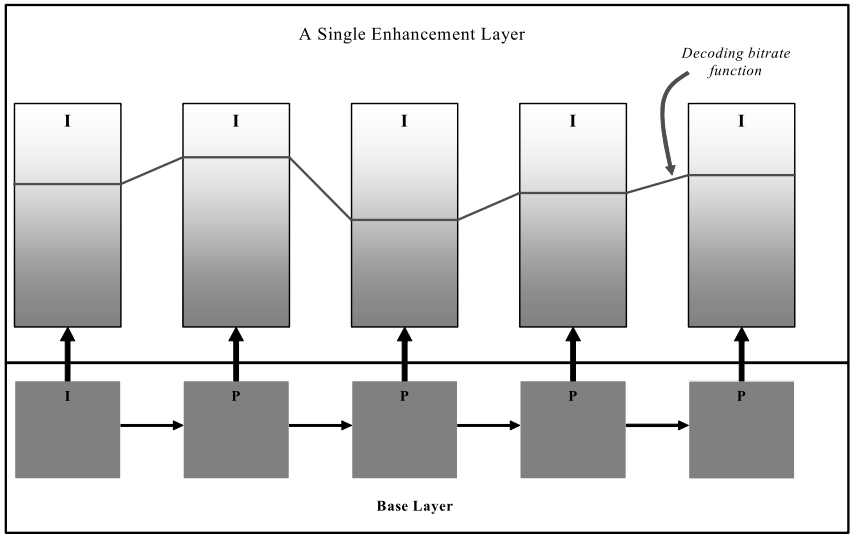


FIGURE 5.8: An MPEG-4 FGS two-layer bit stream.

In the SNR FGS scalability structure described in the previous section, the frame rate of the transmitted video is *locked* to the frame rate of the base layer regardless of the available bandwidth and corresponding transmission bit rate. Since one of the design objectives of FGS is to cover a relatively wide range of bandwidth variation over IP networks (e.g., 100 kbps to 1 Mbps), it is quite desirable that the SNR enhancement tool of FGS be complemented with a temporal scalability tool. It is also desirable to develop a framework that provides the flexibility of choosing between temporal scalability (better motion smoothness) and SNR scalability (higher quality) at transmission time. This, for example, can be used in response to user preferences and/or real-time bandwidth variations at transmission time [44]. For typical streaming applications, both of these elements are not known at the time of encoding the content.

Consequently, the MPEG-4 framework for supporting hybrid temporal–SNR scalabilities building on the SNR FGS structure is described in detail in [44]. This framework provides a new level of abstraction between encoding and transmission processes by supporting *both* SNR and temporal scalabilities through a *single* enhancement layer. Figure 5.9 shows the hybrid scalability structure. In addition to the standard SNR FGS frames, this hybrid structure includes motion-compensated residual frames at the enhancement layer. These motion-compensated frames are referred to as FGS Temporal (FGST) pictures [44].

As shown in Figure 5.9, each FGST picture is predicted from base layer frames that do not coincide temporally with that FGST picture, and therefore, this leads

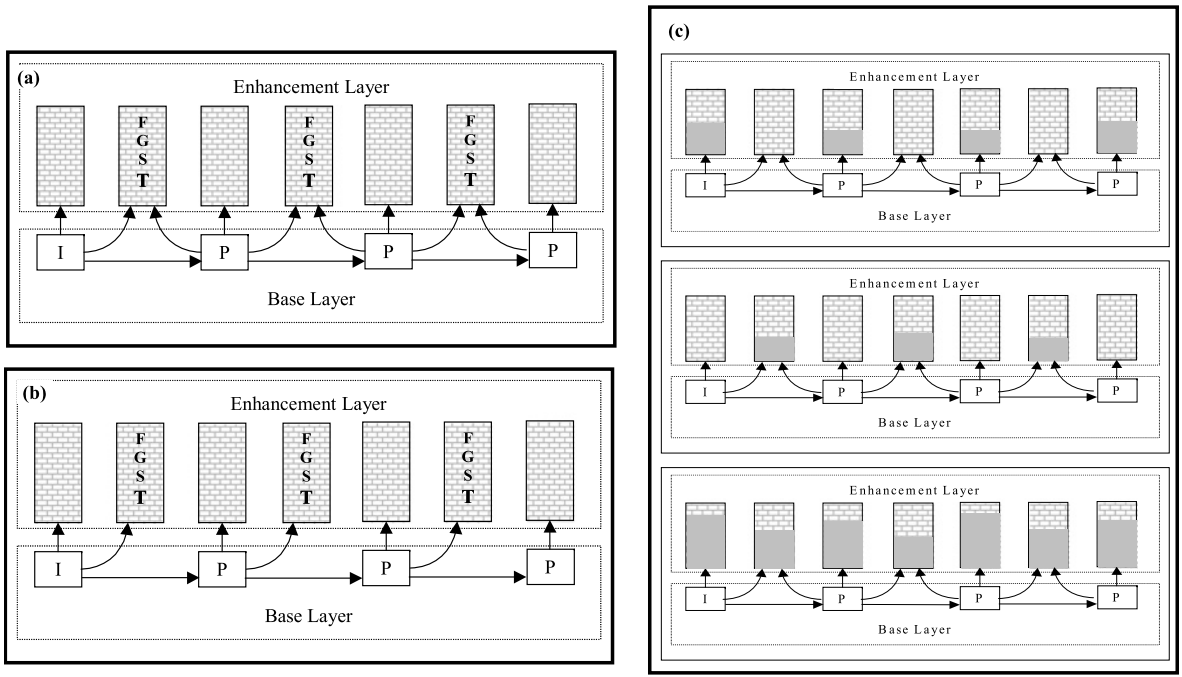


FIGURE 5.9: FGS hybrid temporal–SNR scalability structure with (a) bidirectional and (b) forward prediction FGST pictures and (c) examples of SNR-only (top), temporal-only (middle), or both temporal and SNR (bottom) scalability.

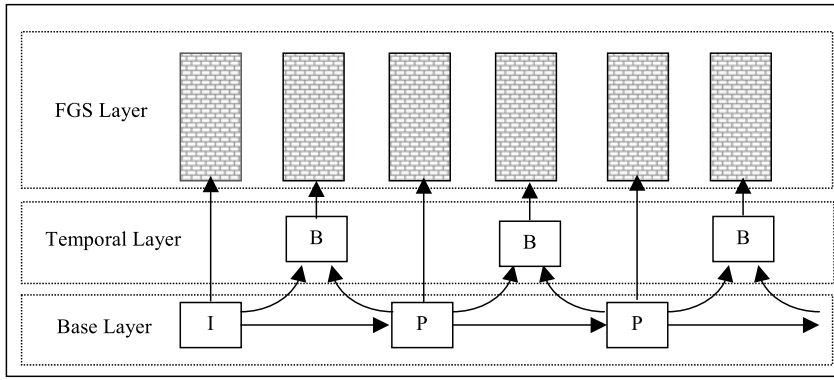


FIGURE 5.10: Multilayer FGS–temporal scalability structure.

to the desired temporal scalability feature. Moreover, the FGST residual signal is coded using the same fine granular video coding method employed for compressing the standard SNR FGS frames.

Each FGST picture includes two types of information: (a) motion vectors (MVs) that are computed in reference to temporally adjacent base layer frames and (b) coded data representing the bit planes DCT signal of the motion-compensated FGST residual. The MVs can be computed using standard macroblock-based matching motion-estimation methods. Therefore, the motion-estimation and compensation functional blocks of the base layer can be used by the enhancement layer codec.

The FGST picture data is coded and transmitted using a data-partitioning strategy to provide added error resilience. Under this strategy, after the FGST frame header, all motion vectors are clustered and transmitted before the residual signal bit planes. The MV data can be transmitted in designated packets with greater protection. More details on hybrid SNR–temporal FGS can be obtained from [44]. Finally, these scalabilities can be further combined in a multilayer manner, and an example of this is shown in Figure 5.10.

5.3.3 Nonstandard FGS Variants

To improve the coding efficiency of FGS, various temporal prediction structures have been proposed. For example, in [8], an additional motion compensation loop is introduced into the enhancement layer using the reconstructed high-quality reference. Furthermore, an improved method is proposed in [9] where an estimation-theoretic framework is presented to obtain the prediction optimally considering both the reconstructed high-quality reference and the base layer information.

This optimization translates into consistent performance gains in compression efficiency at the enhancement layer. Nonetheless, the main disadvantage of such schemes is the high complexity due to the multiple motion estimation loops for the enhancement layer coding.

However, the FGS scheme can also benefit from temporal dependency at the FGS enhancement layer based on one prediction loop. Motion-Compensated FGS (MC-FGS) was first proposed to address this problem in [10]. A high-quality reference, generated from the enhancement layer, can be utilized in the motion compensation loop to get better prediction. However, in case there is a close-loop structure at the enhancement layer, it could induce drift errors when the enhancement layer cannot be guaranteed at the decoder side due to network bandwidth fluctuations. Several methods used to reduce the drift in the MC-FGS structure are discussed in [10].

To introduce temporal prediction into the FGS enhancement layer coding without severe drift errors, several alternative techniques have been proposed. Progressive FGS (PFGS) proposed in [12,13] explores a separate motion compensation loop for the FGS enhancement layer to improve the compression performance and provides means to eliminate the drift error as well. There are two key points in the PFGS coding. One is to use as many predictions from the enhancement reference layers as possible (for coding efficiency) instead of always using the base layer as in MPEG-4 FGS. The other point is to keep a prediction path from the base layer to the highest quality layer across several frames, for error recovery and channel adaptation. Such a prediction path enables lost or erroneous higher quality enhancement layers to be automatically reconstructed from lower layers gradually over a few frames. Thus, PFGS trades off coding efficiency for drift error reduction.

In [14], a robust FGS (RFGS) technique was presented by incorporating the ideas of leaky [10,15] and partial predictions to deal with the drift error. In RFGS, the high-quality reference used in the enhancement layer compensation loop is constructed by combining the reconstructed base layer image and part of the enhancement layer. A frame-based fading mechanism is introduced to cope with the mismatch error. Specifically, at each frame, a uniformly leaky factor between 0 and 1 is applied to the enhancement layer before adding to the base layer image to alleviate the error propagation. Moreover, an adaptive leaky prediction based on the RFGS is proposed in [16] where the leaky factor is determined for each bit plane of enhancement layer according to its significance and location to further improve the coding performance.

Furthermore, several techniques are proposed to achieve more flexible tradeoff between drift errors and coding efficiency at the macroblock level rather than at the frame level. The macroblock-based PFGS (MPFGS) is one such scheme [17,18]. In MPFGS, three INTER modes, HPHR, LPLR, and HPLR, are proposed for the enhancement layer macroblock encoding (see Figure 5.11). In fact,

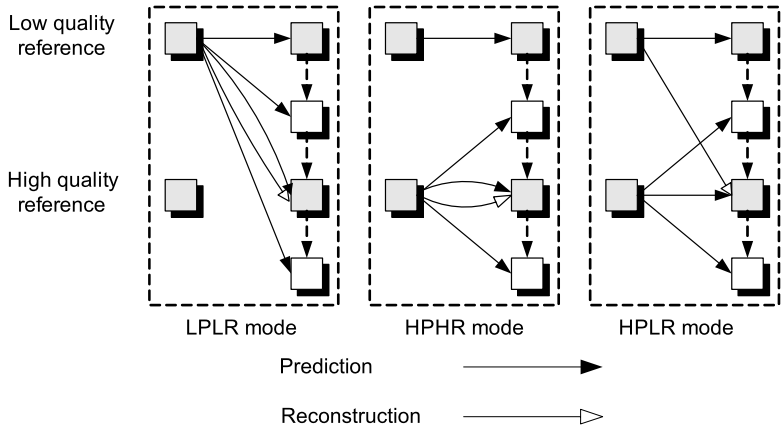


FIGURE 5.11: INTER modes for the enhancement macroblocks in MPFGS.

the HPHR mode is used to get high coding efficiency by using higher quality reference, while the HPLR mode is imposed to attenuate the drifting error by introducing the mismatch error into the encoding process. Assuming that the base layer is always available at the decoder, LPLR and HPLR modes help reset the drift errors potentially caused by the HPHR mode. A decision-making mechanism is presented in MPFGS to choose the optimal prediction mode for each enhancement layer macroblock by considering the error propagation effects and taking advantage of the HPLR mode to achieve a flexible trade-off between high coding efficiency and low drifting error. Another macroblock-based approach is presented in [19,20], called enhanced mode-adaptive FGS (EMFGS). Three predictors, the reconstructed base layer macroblock, the reconstructed enhancement layer macroblock, and the average of the previous two, are proposed in EMFGS. A uniformly fading factor, 0.5, is used to form the third predictor. Also, a mode-selection algorithm is provided to decide the encoding mode of the enhancement layer macroblock.

Another network-aware solution was presented in [45] to alleviate the FGS coding inefficiencies based on the available network conditions—video transcaling (TS), which can be viewed as a generalization of (nonscalable) transcoding. With TS, a scalable video stream that covers a given bandwidth range is mapped into one or more scalable video streams covering different bandwidth ranges. The TS framework exploits the fact that the level of heterogeneity changes at different points of the video distribution tree over wireless and mobile Internet networks. This provides the opportunity to improve the video quality by performing the appropriate TS process. An Internet/wireless network gateway represents a good

candidate for performing TS, thus improving the performance of FGS-based compression schemes.

5.4 MOTION-COMPENSATED WAVELET VIDEO CODECS

As wavelets inherently provide a hierarchical representation of the analyzed content and also have proved very attractive for spatial and quality scalability in still image coding, an intense effort has been deployed since the late-1980s to extend these decompositions in the temporal direction. This can be done by considering the video sequence as a volume of pixels and applying the temporal transform on samples along the temporal dimension. Temporal subbands are then spatially transformed also using a wavelet transform.

5.4.1 Motion-Compensated Temporal Filtering (MCTF)

The idea of temporal extensions of subband decompositions appeared in the late 1980s, with the works of Karlsson and Vetterli [1] and Kronander [2]. In these works the classical temporal closed-loop prediction scheme was replaced by a temporal subband decomposition, which didn't take into account any motion compensation. Later it was shown that motion prediction was also important in these schemes [3] in order to reduce the detail energy subbands, thus leading to much better compression performance and visual quality, and ideally the temporal transform should be applied along the motion trajectories.

The simplest temporal wavelet transform is the Haar transform, performing sums and differences of pairs of frames to obtain respectively approximation and detail subbands. It is illustrated in Figure 5.12 on a group of frames (GOF) of eight frames, which allows performing a maximum of three levels of dyadic decomposition. A review of various MCTF structures for scalable video coding can be found in [42].

Due to the two-tap low-pass and high-pass filters and downsampling by a factor of 2, no boundary problems appear when decomposing a GOF of size 2^L into a number of up to L resolution levels.

Moreover, if motion estimation and compensation is performed between pairs of successive frames, without overlapping, the number of operations and the number of motion vector fields are the same as for coding the same number of frames in a predictive scheme (and equal to $2^L - 1$). However, as pairs of pixels have to be processed in successive frames in order to obtain the coefficients of the approximation and detail frames, the motion invertibility becomes a very important problem.

For example, in a block-based motion-compensated prediction, which is the most usual technique for temporal prediction, the same area in the reference frame

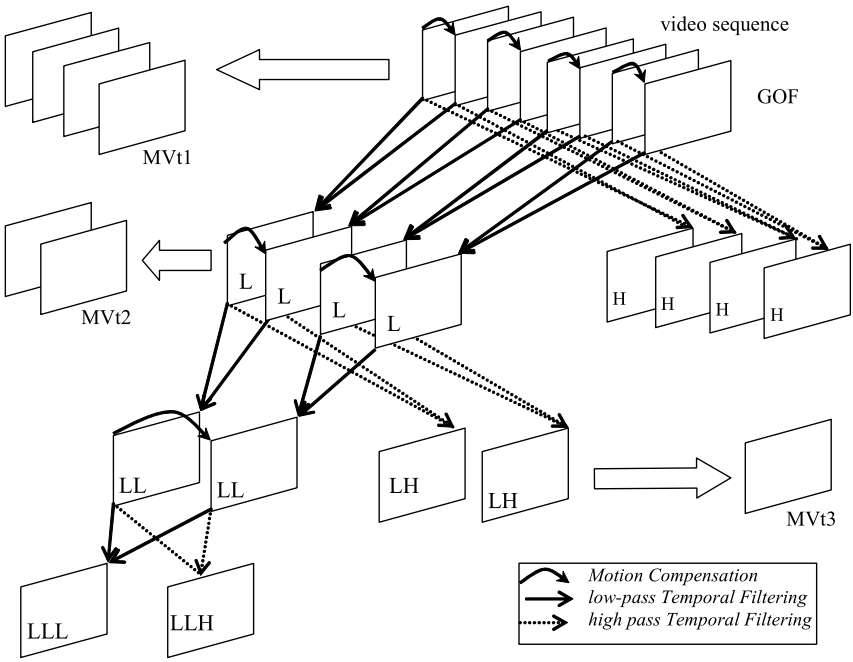


FIGURE 5.12: Temporal Haar wavelet decomposition of a GOF.

can be used to predict several areas in the current frame, while parts of the reference frame are not used at all for prediction. This gives rise to multiple connected and unconnected pixels (see Figure 5.13) [3,4]. In order to avoid such problems, other motion models, such as *meshes*, can be employed [30].

Moreover, in order to take advantage of in-place calculations and guaranteed reversibility of the scheme even for nonlinear operations (such as the operations involving motion compensation), a lifting implementation of the wavelet filter bank was proposed [5,21]. This way, after splitting the input samples in odd and even indexed ones, theoretically any biorthogonal filter bank with finite impulse responses can be represented with a finite number of predict-update (see Figure 5.14) steps, possibly followed by multiplication with a constant.

In the case of temporal decomposition of the video, motion estimation is first performed between input frames, and the motion vector fields (denoted by v in Figure 5.15) are used for motion-compensated operations in both the predict and the update steps.

An important remark is that the predict operator can use *all* the even indexed input frames (denoted by x_{2t_t}) to perform the motion-compensated prediction of the odd indexed frames (denoted by $\{x_{2t+1}\}$), while the update operator can use

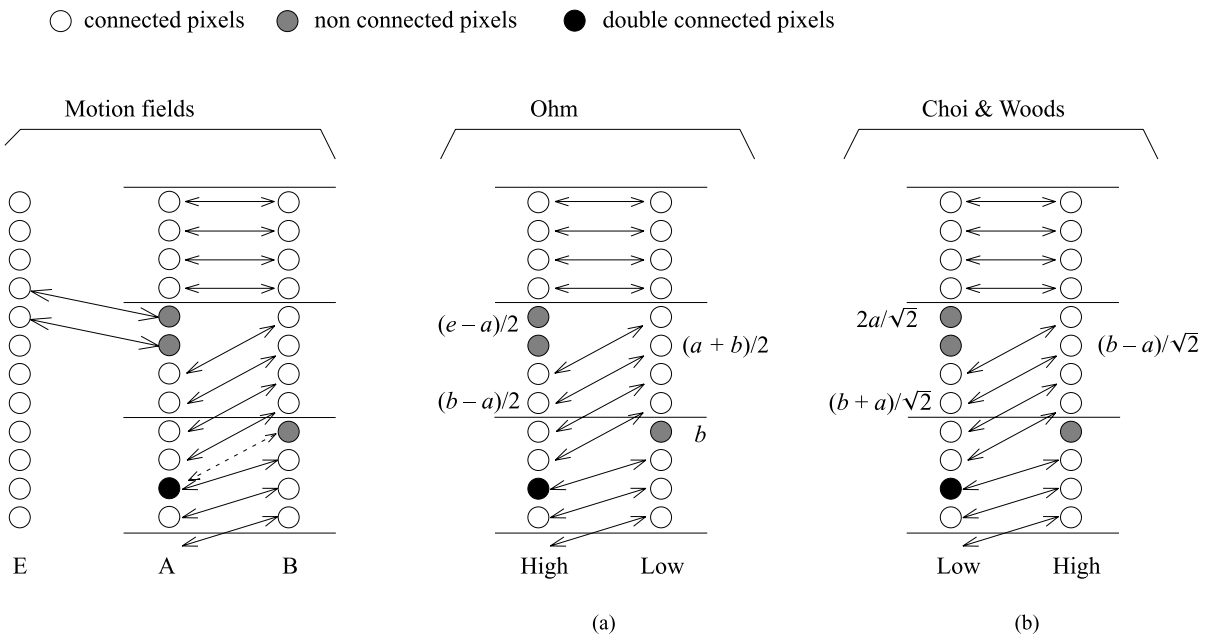


FIGURE 5.13: Temporal MC Haar filtering: connected, unconnected, and multiple connected pixels.

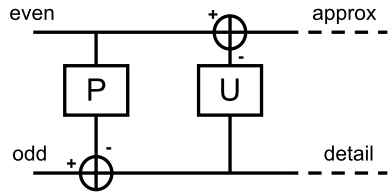


FIGURE 5.14: Basic steps of a lifting scheme.

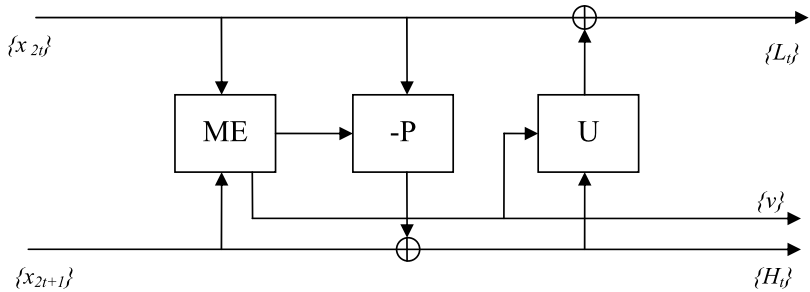


FIGURE 5.15: Spatiotemporal motion-compensated lifting scheme.

all the detail frames ($\{H_{t_t}\}$) thus computed to obtain the approximation subband frames ($\{L_{t_t}\}$).The predict and update operators then also involve the motion vectors used to match corresponding positions. Therefore, in the $t + 2D$ framework they actually become spatiotemporal operators:

$$H_t(\mathbf{n}) = x_{2t+1}(\mathbf{n}) - \mathrm{P}\Big(\big\{x_{2(t-k)}, v_{2t+1}^{2(t-k)}\big\}_{k\in T_k^p}\Big), \quad \forall \mathbf{n} \in S,$$
$$L_t(\mathbf{p}) = x_{2t}(\mathbf{p}) + U\Big(\big\{H_{t-k}, v_{2t}^{2(t-k)+1}\big\}_{k\in T_k^u}\Big), \quad \forall \mathbf{p} \in S,$$

where v_i^j is the motion vector field used to predict the current frame i from the reference frame j, T_k^p (respectively T_k^u) is the support of the temporal predict (respectively update) operator, and the spatial position is denoted by $\mathbf{n}$ or $\mathbf{p}$.

In the simplest case of the Haar multiresolution analysis, the previous relations become

$$H_t(\mathbf{n}) = x_{2t+1}(\mathbf{n}) - x_{2t}\big(\mathbf{n} - v_{2t+1}^{2t}\big), \quad \forall \mathbf{n} \in S,$$
$$L_t(\mathbf{p}) = x_{2t}(\mathbf{p}) + H_t\big(\mathbf{p} + v_{2t}^{2t+1}\big), \quad \forall \mathbf{p} \in S.$$

However, from the temporal prediction viewpoint, it is better to make use of longer filters. The biorthogonal 5/3 filter bank has been most studied. In this

case, both forward and backward motion vectors need to be used for a bidirectional prediction. The analysis equations have the form

$$H_t(\mathbf{n}) = x_{2t+1}(\mathbf{n}) - \frac{1}{2}\left[x_{2t}\left(\mathbf{n} - v_{2t+1}^{2t}\right) + x_{2t+2}\left(\mathbf{n} - v_{2t+1}^{2t+2}\right)\right], \quad \forall \mathbf{n} \in S,$$

$$L_t(\mathbf{p}) = x_{2t}(\mathbf{p}) + \frac{1}{4}\left[H_{t-1}\left(\mathbf{p} + v_{2t}^{2t+1}\right) + H_t\left(\mathbf{p} + v_{2t+1}^{2t+2}\right)\right], \quad \forall \mathbf{p} \in S.$$

Due to the fact that a bidirectional prediction is used in this structure, the number of motion vector fields is double compared with the Haar decomposition, and therefore the coding of this information may represent an important part of the bit stream at low bit rates. Efficient algorithms are needed to further exploit redundancies between motion vector fields at the same temporal decomposition level or at different levels [23].

To effectively deal with the problem of motion-compensated temporal wavelet filtering associated with fractional precision motion vectors, many-to-one mapping for the covered areas and nonreferred pixels for the uncovered areas [31], proposes a new and general lifting structure (see Figure 5.16) that unifies all the

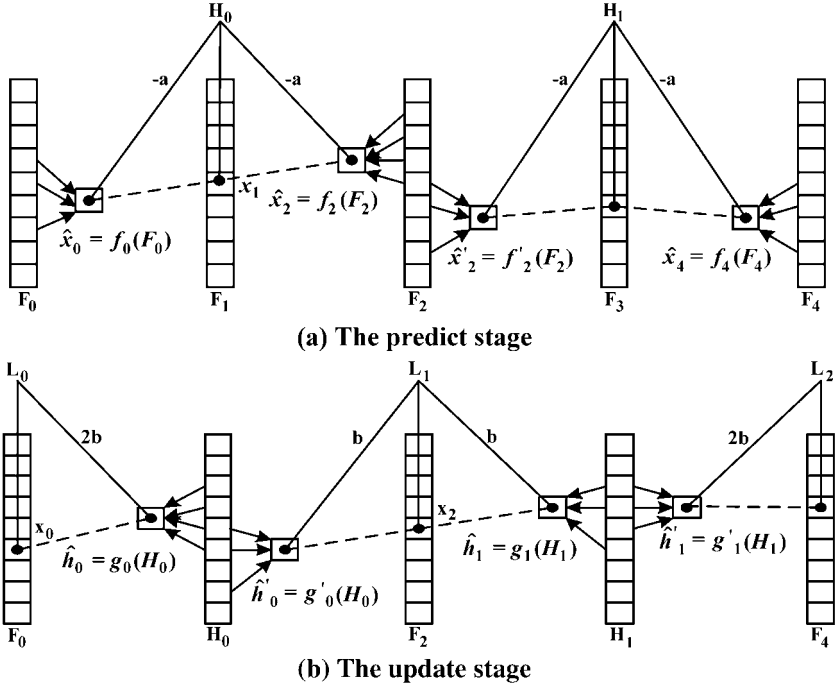


FIGURE 5.16: The Barbell lifting scheme.

previous works to solve this problem and enables any traditional motion compensation techniques in block-based motion prediction coding to be easily adopted in the MCTF framework. The core of this work is a so-called *Barbell lifting* scheme, in which instead of a single pixel value, a function of a set of pixel values is used as the input to the lifting step. The Barbell lifting scheme essentially moves any existing effective motion prediction schemes in traditional video coding to the MCTF frames.

A new update scheme, energy distributed update (EDU), is proposed in [32] to avoid a second set of motion vectors or complex and inaccurate inversion of the motion information used in the traditional update step. The idea is to update where predict is made by distributing high-pass signals to the low-pass frame. Meanwhile, it provides further coding efficiency gain.

5.4.2 Three-Dimensional (3D) Wavelet Coefficients Coding

After 3D (temporal and spatial) wavelet analysis, a video sequence will be decomposed into a certain number of 3D subbands. For example, in Figure 5.17, a three-level wavelet (motion compensated) decomposition is performed in the temporal direction, followed by a three-level 2D spatial dyadic decomposition within each of the resulting temporal bands.

The next step in 3D wavelet video coding is to encode the transformed 3D wavelet coefficients in each subband efficiently. Since the subband structure in 3D wavelet decomposition for video sequence is very similar to the subband structure

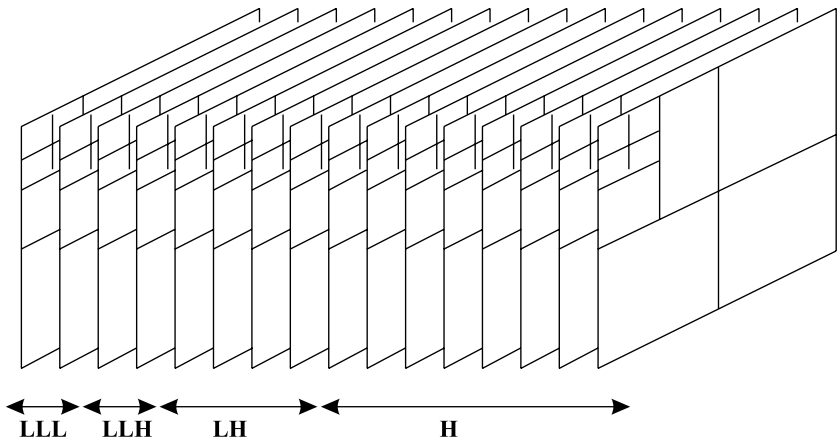


FIGURE 5.17: Separable 3D wavelet transform. Three-level dyadic temporal (motion compensated) wavelet decomposition, followed by three-level 2D spatial dyadic decomposition.

in 2D wavelet decomposition for image, it is natural to extend many existing 2D wavelet-based image coding techniques, such as SPIHT [33], EBCOT [34], and EZBC [35], to the 3D case. As a matter of fact, almost all the existing 3D wavelet coefficients coding schemes use one form of these 3D extensions, such as 3D SPIHT [38], 3D ESCOT [36,37], and 3D EZBC [38,39].

Generally speaking, after 3D (motion compensated) wavelet decomposition, there is not only spatial similarity inside each frame across the different scale, but also temporal similarity between two frames at the same temporal scale. Furthermore, temporal linkages of coefficients between frames typically show more correlation along the motion trajectory. An efficient 3D wavelet coefficient coding scheme should exploit these properties as much as possible. Several algorithms for texture coding in 3D wavelet schemes have been developed.

5.4.2.1 3D SPIHT

3D SPIHT is an extension of the concept of SPIHT still image coding to 3D video coding. As we know, the SPIHT algorithm takes advantages of the nature of energy clustering of subband/wavelet coefficients in frequency and space and exploits the similarity between subbands. It utilizes three basic concepts: (1) searching for sets in spatial-orientation trees in a wavelet transform; (2) partitioning the wavelet transform coefficients in these trees into sets defined by the level of the highest significant bit in a bit plane representation of their magnitudes; and (3) coding and transmitting bits associated with the highest remaining bit planes first.

There is no constraint to dimensionality in the SPIHT algorithm itself, as pixels are sorted regardless of dimensionality. The 3D SPIHT scheme can be easily extended from 2D SPIHT, with the following three similar characteristics: (1) partial ordering by magnitude of the 3D wavelet transformed video with a 3D set partitioning algorithm; (2) ordered bit plane transmission of refinement bits; and (3) exploitation of self-similarity across spatiotemporal orientation trees.

For the 3D wavelet coefficients, a new 3D spatiotemporal orientation tree and its parent–offspring relationships are defined. For pure dyadic wavelet decomposition with an alternate separable wavelet transform in each dimension, a straightforward extension from the 2D case is to form a node in 3D SPIHT as a block with eight adjacent pixels, two in each dimension, hence forming a node of $2 \times 2 \times 2$ pixels. The root nodes (at the highest level of the pyramid) have one pixel with no descendants and the other seven pointing to eight offspring in a $2 \times 2 \times 2$ cube at corresponding locations at the same level. For nonroot and nonleaf nodes, a pixel has eight offspring in a $2 \times 2 \times 2$ cube one level below in the pyramid. For nondyadic decomposition similar to the 2D wavelet packet decomposition case, the $2 \times 2 \times 2$ offspring nodes are split into pixels in these smaller subbands at the corresponding orientation in the nodes at the original level. For the common

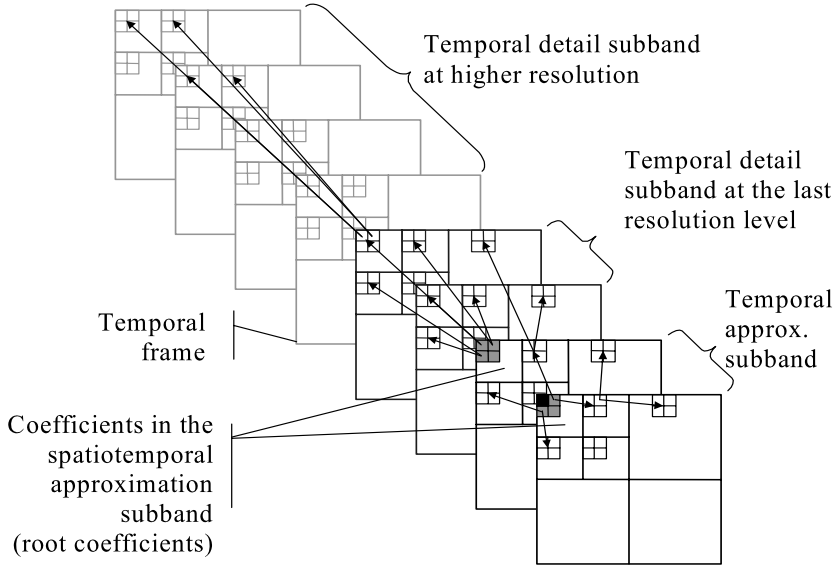


FIGURE 5.18: Parent–offspring relationship in a spatiotemporal decomposition.

$t + 2$D type of wavelet decomposition the parent–offspring relationship is shown in Figure 5.18. With such defined 3D spatiotemporal trees, the coefficients can be compressed into a bit stream by feeding the 3D data structure to the 3D SPIHT coding kernel. The 3D SPIHT kernel will sort the data according to the magnitude along the spatiotemporal orientation trees (sorting pass) and refine the bit plane by adding necessary bits (refinement pass).

5.4.2.2 3D ESCOT

The 3D SPIHT coding scheme provides natural scalability in rate (quality). However, it is difficult to provide temporal or spatial scalabilities due to the inherent spatiotemporal tree structure. Even with extra effort, it can only provide partial temporal or spatial scalabilities by modifying the decoder or encoder [35]. However, 3D ESCOT [36,37] can provide full rate, temporal and spatial scalabilities by constraining the encoding of wavelet coefficients independently within each subband. Meanwhile, the R–D optimized bit stream truncation process after encoding guarantees a bit stream with the best video quality given a bit rate constraint.

The 3D ESCOT scheme is in principle very similar to EBCOT [34] in the JPEG-2000 standard, which offers high compression efficiency and other functionalities

(e.g., error resilience and random access) for image coding. In extending the 2D EBCOT algorithm to 3D ESCOT, a different coding structure is used to form a new set of 3D contexts for arithmetic coding that makes the algorithm very suitable for scalable video compression. Specifically, each of the subbands is coded independently in the extended coding structure. The advantage of doing so is that each subband can be decoded independently to achieve flexible spatial/temporal scalability. The user can mix an arbitrary number of spatiotemporal subbands in any order to obtain the desired spatial or temporal resolution.

Unlike the EBCOT encoder [34] in JPEG2000, the ESCOT encoder takes a subband as a whole entity. There are two reasons for this. (1) Normally a video frame has lower resolution than a still image. Not splitting a subband further into many small 3D blocks brings better coding efficiency of the context-based adaptive arithmetic coder. (2) Taking a subband as a whole entity is also convenient for incorporating the possible motion model in the coding process, since within the same 3D subband, the motion vector may point from any coefficients on a temporal plane to any other coefficients on other temporal planes.

As in the 2D EBCOT case, the contexts for 3D ESCOT are also formed with immediate neighbors in the same subband. The difference is that the immediate neighbors are now in three directions instead of two: horizontal, vertical, and temporal (Figure 5.19). In addition, the temporal neighbors not only may be spatially collocated in different frames, but also may be neighbors pointed to by motion vectors across frames with a certain motion model [36,37].

The encoding of the 3D wavelet coefficients in the 3D ESCOT scheme is done bit plane by bit plane. For each bit plane, the coding procedure consists of three distinct passes: Significance Propagation, Magnitude Refinement, and Normalization, which are applied in turn. Each pass processes a "fractional bit plane." In each pass, the scanning order is along the horizontal direction first, the vertical direction second, and finally the temporal direction. In the Significance Propagation pass, samples that are not yet significant but have a "preferred neighborhood" are

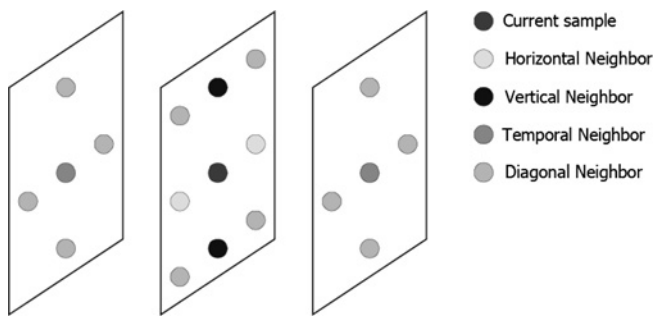


FIGURE 5.19: Immediate neighbors of a sample in 3D ESCOT coding.

processed. A sample has a "preferred neighborhood" if and only if the sample has at least a significant immediate diagonal neighbor for a HHH (high frequency in three directions) subband or a significant immediate horizontal, vertical, or temporal neighbor for the other types of subbands. In the Magnitude Refinement pass, the samples that have been significant in the previous bit planes are encoded. In the Normalization pass, those samples that have not yet been coded in the previous two passes are coded.

In the previous stage, each subband is coded separately up to a specific precision and each forms an independent bit stream. The objective of optimal bit stream truncation is to construct a final bit stream that satisfies the bit rate constraint and minimizes the overall distortion. As in the EBCOT algorithm [34], the end of each pass at each "fractional" bit plane is a candidate truncation point with a precalculated R–D value pair for that subband. A straightforward way to achieve R–D optimized truncation is to find the convex hull of the R–D pairs at the end of each fractional bit plane and truncate only at the candidate truncation points that are on the convex hull.

To achieve quality scalability, a multilayer bit stream may be formed, where each layer represents a quality level. Depending on the available bandwidth and the computational capability, the decoder can choose to decode up to the layer it can handle. The fractional bit plane coding ensures that the bit stream is finely embedded. Since each subband is independently coded, the bit stream of each subband is separable. The encoder can choose to construct a bit stream favoring spatial scalability or temporal scalability. Also, the decoder can easily extract only a few subbands and decode only these subbands. Therefore, the implementation of resolution scalability and temporal scalability is natural and easy.

5.4.2.3 3D EZBC

3D EZBC is an extension of the EZBC image coder [35] to allow coding of three-dimensional wavelet coefficients. The concept of EZBC is inspired by the success of two popular embedded image coding techniques: zero-tree/-block coding, such as SPIHT [33], and context modeling of the subband/wavelet coefficients, such as EBCOT [34]. As discussed, zero-tree/-block coding takes advantage of the natural energy clustering of subband/wavelet coefficients in frequency and in space and exploits the similarity between subbands. Moreover, instead of all pixels, only a small number of elements in the lists [33] need to be processed in individual bit plane coding passes. Thus, processing speed for this class of coders is very fast. However, in the context model based coders [34], individual samples of the wavelet coefficients are coded bit plane by bit plane using context-based arithmetic coding to effectively exploit the strong correlation of subband/wavelet coefficients within and across subbands. Nevertheless, unlike zero-tree/-block coders, these algorithms need to scan all subband/wavelet coefficients at least once to

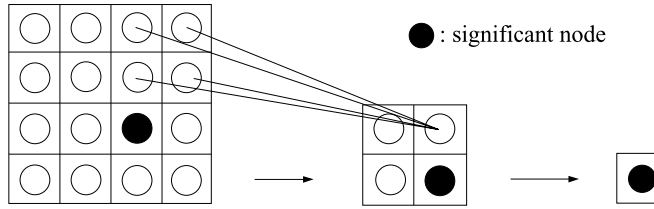


FIGURE 5.20: Quad tree decomposition in 3D EZBC.

finish coding of a full bit plane, with an implied higher computation cost. The EZBC algorithm combines the advantages of these two coding techniques, that is, low computation complexity and effective exploitation of correlation of subband coefficients, using both ZeroBlocks of subband/wavelet coefficients and context modeling.

Similar to EZBC for image coding, 3D EZBC is based on quad tree representation of the individual subbands and frames. The bottom quad tree level, or pixel level, consists of the magnitude of each subband coefficient. Each quad tree node of the next higher level is then set to the maximum value of its four corresponding nodes at the current level; see Figure 5.20. In the end, the top quad tree node corresponds to the maximum magnitude of all the coefficients from the same subband. As in EZBC, 3D EZBC uses this quad tree-based zero-block coding approach for hierarchical set-partition of subband coefficients to exploit the strong statistical dependency in the quad tree representation of the decomposed subbands. Furthermore, to code the significance of the quad tree nodes, context-based arithmetic coding is used. The context includes eight first-order neighboring nodes of the same quad tree level and the node of the parent subband at the next lower quad tree level. Experiments have shown that including a node in the parent subband in the interband context model is very helpful in predicting the current node, especially at higher levels of a quad tree.

Like SPIHT and other hierarchical bit plane coders, lists are used for tracking the set-partitioning information. However, the lists in 3D-EZBC are separately maintained for nodes from different subbands and quad tree levels. Therefore, separate context models are allowed to be built up for the nodes from different subbands and quad tree levels. In this way, statistical characteristics of quad tree nodes from different orientations, subsampling factors, and amplitude distributions are not mixed up. This ensures a resolution scalable bit stream while maintaining the desirable low complexity feature of this class of coders.

5.4.3 Variants and Extensions: UMCTF, 3-bands

The concept of "unconstrained MCTF" (UMCTF) [10,43] allows very useful extensions of the MCTF. By selecting the temporal filter coefficients appropriately,

multiple reference frames and bidirectional prediction can be introduced, such as in H.264, in the MC-wavelet framework. No update step is used, however, which makes this scheme comparable with an open-loop multiresolution predictive structure. We can adaptively change the number of reference frames, the relative importance attached to each reference frame, the extent of bidirectional filtering, and so on. Therefore, with this filter choice, the efficient compensation strategies of conventional predictive coding can be obtained by UMCTF, while preserving the advantages of conventional MCTF.

Other extensions of the temporal transform are aimed at providing nondyadic scalability factors. This can be achieved by M-band filter banks, for which a general framework was proposed in [25]. In particular, a three-band filter bank in lifting form was proposed in [24] and is illustrated in Figure 5.21. For simplicity, Figure 5.21 shows only predict and update blocks; however, as in the dyadic case, they involve motion estimation/compensation.

Following the notation in Figure 5.21, the analysis equations, which lead to one approximation and two detail subbands, are

$$\begin{cases} H_t^+(\mathbf{n}) = x_{3t+1}(\mathbf{n}) - P^+\big(\{x_{3t}\}_{t\in N}\big), \\ H_t^-(\mathbf{n}) = x_{3t-1}(\mathbf{n}) - P^-\big(\{x_{3t}\}_{t\in N}\big), \\ L_t(\mathbf{p}) = x_{3t}(\mathbf{p}) + U^+\big(\{H_t^+\}_{t\in N}\big) + U^-\big(\{H_t^-\}_{t\in N}\big). \end{cases}$$

Note that in this scheme all the frames indexed by multiples of three are used by the two prediction operators. For example, by choosing frames x_{3t} and x_{3t+3} for the prediction of frame x_{3t+1} and likewise choosing frames x_{3t-3} and x_{3t} for predicting frame x_{3t-1}, a structure similar to the classical IBBP ... structure can be obtained.

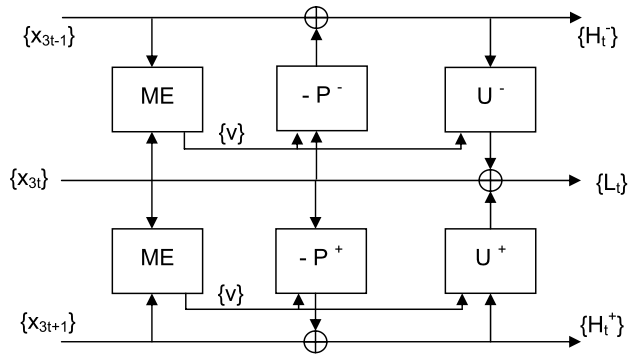


FIGURE 5.21: Three-band lifting scheme.

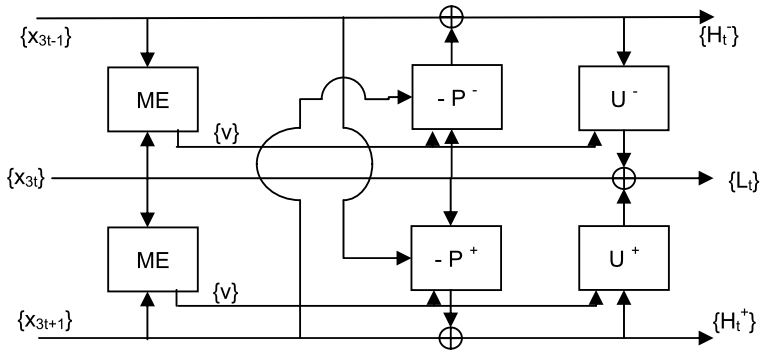


FIGURE 5.22: A three-band lifting-like scheme.

However, the simplest choice, corresponding to a Haar-like transform, is to have identity predict operators and linear update operators. In this case, the analysis equations become

$$\begin{cases} H_t^+(\mathbf{n}) = x_{3t+1}(\mathbf{n}) - x_{3t}\big(\mathbf{n} - v_{3t+1}^{3t}\big), \\ H_t^-(\mathbf{n}) = x_{3t-1}(\mathbf{n}) - x_{3t}\big(\mathbf{n} - v_{3t-1}^{3t}\big), \\ L_t(\mathbf{p}) = x_{3t}(\mathbf{p}) + \dfrac{1}{4}\big(H_t^+\big(\mathbf{p} + v_{3t+1}^{3t}\big) + H_t^-\big(\mathbf{p} + v_{3t-1}^{3t}\big)\big). \end{cases}$$

More complex *lifting-like* schemes (as in Figure 5.22) have been proposed in [26], as well as other possible M-band motion-compensated temporal structures.

These structures allow a frame rate adaptation from 30 to 10 fps, for example, or from 60 to 20 fps. Flexible frame rate changes can be achieved by cascading dyadic and M-band filter banks.

Another direction for the extension of spatiotemporal transforms is to replace the 2D wavelet decomposition by other spatial representations, such as wavelet packets [27] or general filter banks [29], which also allow for more flexible spatial scalability factors [28].

5.4.4 Switching Spatial and Temporal Transforms

The interframe wavelet video coding schemes presented in the previous sections employ MCTF before the spatial wavelet decomposition is performed. Throughout the chapter we refer to this class of interframe wavelet video coding schemes as $t + 2$D MCTF. Despite their good coding efficiency performance and low complexity, these types of MCTF structures have also several drawbacks.

1. *Limited motion-estimation efficiency*. $t + 2$D MCTF are inherently limited by the quality of the matches provided by the employed motion estimation algorithm. For instance, discontinuities in the motion boundaries are represented as high frequencies in the wavelet subbands, and the "Intra/Inter" mode switch for motion estimation is not very efficient in $t + 2$D MCTF schemes, as the spatial wavelet transform is applied globally and cannot encode the resulting discontinuities efficiently. Moreover, the motion estimation accuracy, motion model, and adopted motion estimation block size are fixed for all spatial resolutions, thereby leading to suboptimum implementations compared with nonscalable coding that can adapt the motion estimation accuracy based on the encoded resolution. Also, because the motion vectors are not naturally spatially scalable in $t + 2$D MCTF, it is necessary to decode a large set of vectors even at lower resolutions.
2. *Limited efficiency spatial scalability*. If the motion reference during $t + 2$D MCTF is, for example, at HD resolution and decoding is performed at a low resolution (e.g., QCIF), this leads to "subsampling phase drift" for the low resolution video.
3. *Limited spatiotemporal decomposition structures*. In $t + 2$D MCTF, the same temporal decomposition scheme is applied for all spatial subbands. Hence, the same levels of temporal scalability are provided independent of the spatial resolution.

A possible solution for the aforementioned drawbacks is to employ "in-band temporal filtering" schemes, where the order of motion estimation and compensation and that of the spatial wavelet transform (2D-DWT) are interchanged, which we denote as $2D+t$ MCTF schemes. The spatial wavelet transform for each frame is entirely performed first and multiple separate motion compensation loops are used for the various spatial wavelet bands in order to exploit the temporal correlation present in the video sequence (see Figure 5.23). In contrast to the method of Figure 5.15, where spatial decomposition steps were interleaved with the temporal tree, MCTF can now also be applied to spatial high-pass (wavelet) bands. Subsequently, coding of the wavelet bands after temporal decorrelation can be done using spatial-domain coding techniques such as bit plane coding followed by arithmetic coding or transform-domain coding techniques based on DCT, wavelets, and so on.

5.4.5 Motion Estimation and Compensation in the Overcomplete Wavelet Domain

Due to the decimation procedure in the spatial wavelet transform, the wavelet coefficients are not shift invariant with reference to the original signal resolution. Hence, translation motion in the spatial domain cannot be accurately estimated and compensated from the wavelet coefficients, thereby leading to a significant

FIGURE 5.23: Multiresolution motion compensation coder using "in-band prediction."

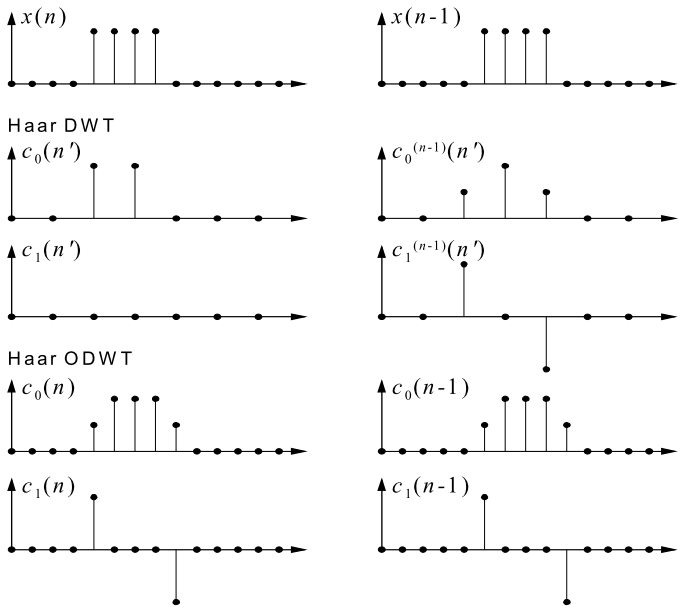


FIGURE 5.24: Shift variance of the Haar wavelet transform. Right signal shifted by one sample to the right, low-pass and high-pass coefficients in Haar DWT and Haar ODWT.

coding efficiency loss (see Haar 1D-DWT case in Figure 5.24). To avoid this inefficiency, motion estimation and compensation should be performed in the overcomplete wavelet domain rather than in the critically sampled domain (see Haar 1D-ODWT case in Figure 5.24). Overcomplete discrete wavelet data (ODWT) can be obtained through a similar process as the critically sampled discrete wavelet signals by omitting the subsampling step. Consequently, the ODWT generates more samples than DWT, but enables accurate wavelet domain motion compensation for the high-frequency components, and the signal does not bear frequency-inversion alias components.

Despite the fact that ODWT generates more samples, an ODWT-based encoder still needs to only encode the critically sampled coefficients. This is because the overcomplete transform coefficients can be generated locally within the decoder. Moreover, when the motion shift is known before analysis and synthesis filtering are performed, it is only necessary to compute those samples of the overcomplete representation that correspond with the actual motion shift.

The $t + 2$D MCTF schemes (Figure 5.25a) can be easily modified into 2D $+ t$ MCTF (Figure 5.25b).

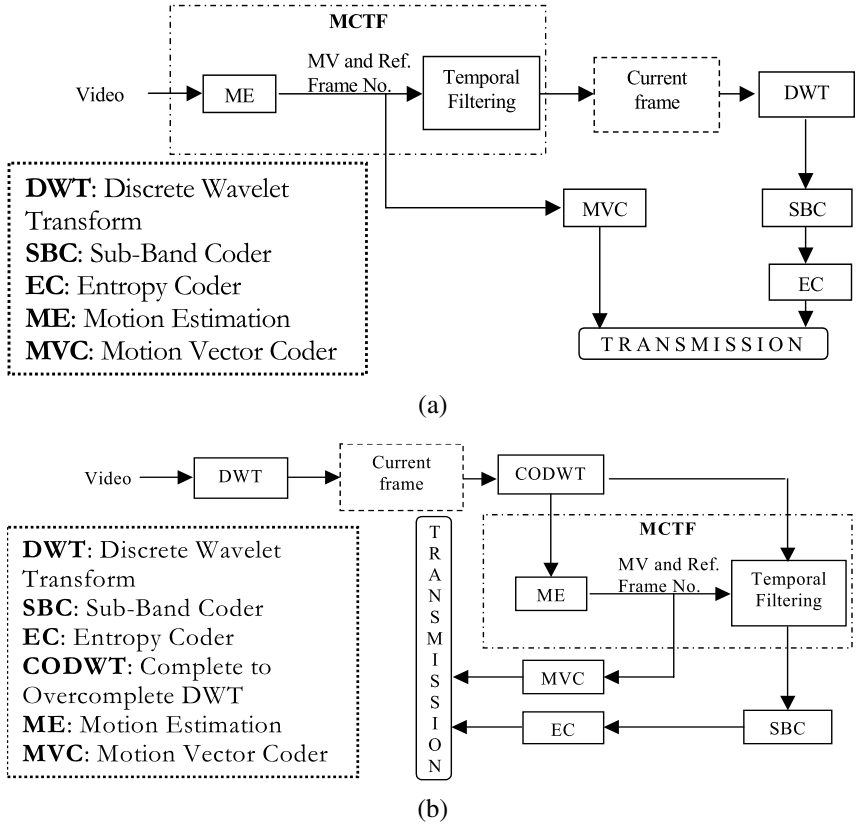


FIGURE 5.25: (a) The encoding structure that performs open-loop encoding in the spatial domain – t + 2D MCTF. (b) The encoding structure that performs open-loop encoding in the wavelet domain (in-band) – 2D + t MCTF.

More specifically, in 2D + t MCTF, the video frames are spatially decomposed into multiple subbands using wavelet filtering, and the temporal correlation within each subband is removed using MCTF (see [19,20]). The residual signal after the MCTF is coded subband by subband using any desired texture coding technique (DCT based, wavelet based, matching pursuit, etc.). Also, all the recent advances in MCTF can be employed for the benefit of 2D + t schemes, which have been first introduced in [46–48].

5.5 MPEG-4 AVC/H.264 SCALABLE EXTENSION

As scalable modes in other standards, MPEG-4 AVC/H.264 scalable extension enables scalabilities while maintaining the compatibility of the base layer to the MPEG-4 AVC/H.264 standard. MPEG-4 AVC/H.264 scalable extension provides temporal, spatial, and quality scalabilities. Those scalabilities can be applied simultaneously. In MPEG-4 AVC/H.264, any frame can be marked as a reference frame that can be used for motion prediction for the following frames. Such flexibility enables various motion-compensated prediction structures (see Figure 5.26).

The common prediction structure used in the MPEG-4 AVC/H.264 scalable extension is the hierarchical-B structure, as shown in Figure 5.26. Frames are categorized into different levels. B-frames at level i use neighboring frames at level $i - 1$ as references. Except for the update step, MCTF and hierarchical-B have the same prediction structure. Actually, at the decoder, the decoding process of hierarchical-B and that of MCTF without the update step is the same. Such a hierarchical prediction structure exploits both short-term and long-term temporal correlations as in MCTF. The other advantage is that such a structure can inherently provide multiple levels of temporal scalability. Other temporal scalability schemes compliant with MPEG-4 AVC/H.264 have been presented in [25] and are shown to provide increased efficiency and robustness on error-prone networks.

To achieve SNR scalability, enhancement layers, which have the same motion-compensated prediction structure as the base layer, are generated with finer quantization step sizes. At each enhancement layer, the differential signals to the previous layer are coded. Basically it follows the scheme shown in Figure 5.26.

To achieve spatial scalability, the lower resolution signals and the higher resolution signals are coded into different layers. Also, coding of the higher resolution signals uses bits for the lower resolution as prediction. In contrast to previous coding schemes, the MPEG-4 AVC/H.264 scalable extension can set a constraint on

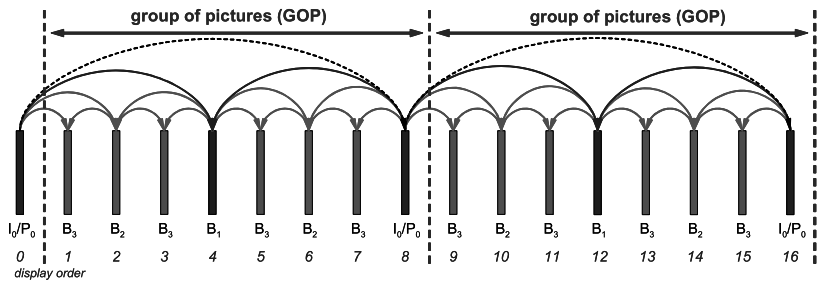


FIGURE 5.26: Four-level hierarchical-B prediction structure.

the interlayer prediction among different resolutions in which only intra-coded macroblocks are reconstructed to predict the higher resolution, whereas for inter-coded macroblocks, only the motion-compensated residue signals are allowed to predict the corresponding residue signals at the higher resolution. The advantage of such a constraint is that it reduces the decoding complexity because the decoder does not need to do motion compensation for the lower layer. The drawback is that such constraint may have a coding performance penalty.

5.6 SPATIOTEMPORAL–SNR TRADE-OFFS FOR IMPROVED VISUAL PERFORMANCE

In [41], it was shown that performing trade-offs among spatial, temporal, and SNR resolutions as a function of content characteristics often results in a considerably improved user experience for multimedia applications. Nevertheless, this multidimensional flexibility also brings two major challenges. First, no objective measure exists that can quantify the impact on the video quality after multidimensional adaptation (MDA) operations in a synergistic manner, as each component of MDA affects the video quality in a very distinct way. Second, even with an acceptable quality measurement, effective methods for modeling the relationship between video quality and various adaptation operations are important for deciding the right MDA given a resource constraint. To solve this challenge, a general classification-based prediction framework was used successfully in [41] for selecting the preferred MDA operations based on subjective quality evaluation. For this purpose, domain-specific knowledge or general unsupervised clustering was first deployed to construct distinct categories within which the videos share similar preferred MDA operations. Thereafter, a machine learning-based method was applied where the low-level content features extracted from the compressed video streams are employed to train a framework for the problem of joint spatiotemporal–SNR adaptation selection.

5.7 SUMMARY AND FURTHER READING

New perspectives in video compression are reinforced by recent advances in scalable video coding. MCTF-based coders provide high flexibility in bit stream scalability across different temporal, spatial, and quality resolutions. In addition, they provide better error resilience than conventional (prediction based) coders. In fact, MCTF-based coders are better able to separate relevant from irrelevant information. The temporal low-pass bands highlight information that is consistent over a large number of frames, establishing a powerful means for exploiting multiple frame redundancies not achievable by conventional frame-to-frame or multiframe

prediction methods. Moreover, noise and quickly changing information that cannot be handled by motion compensation appear in the temporal, high-pass bands, which can supplement the low-pass bands for more accurate signal reproduction whenever desirable, provided that a sufficient data rate is available. Hence, the denoising process that is often applied as a preprocessing step before conventional video compression is an integral part of scalable MCTF-based coders.

Due to the nonrecursive structure, higher degrees of freedom are possible for both encoder and decoder optimization. In principle, a decoder could integrate additional signal synthesis elements whenever the received information is incomplete, such as frame-rate up-conversion, film grain noise overlay, or other elements of texture and motion synthesis, which could easily be integrated as a part of the MCTF synthesis process without losing any synchronization between encoder and decoder. From this point of view, even though many elements of MCTF in the lifting interpretation can be regarded as extensions of proven techniques from MC prediction-based coders, this framework exhibits and enables a number of radically new options in video encoding. However, when a wavelet transform is applied for encoding of the low-pass and high-pass frames resulting from the MCTF process, the commonalities with 2D wavelet coding methods are obvious. If the sequence of spatial and temporal filtering is exchanged ($2\mathrm{D} + t$ instead of $t + 2\mathrm{D}$ wavelet transform), MCTF can be interpreted as a framework for further interframe compression of (intra frame restricted) 2D wavelet representations such as JPEG 2000. From this point of view, a link between the previously separate worlds of 2D wavelet coding with their excellent scalability properties and compression-efficient motion-compensated video coding schemes is established by MCTF. This shows the high potential for future developments in the area of motion picture compression, even allowing seamless transition between intra frame and inter frame coding methods, depending on the application requirements for flexible random access, scalability, high compression, and error resilience. Furthermore, scalable protection of content, allowing access management for different resolution qualities of video signals, is a natural companion of scalable compression methods.

Nevertheless, a number of topics can be identified that still require further research, but may also lead to even higher compression performance of this new class of video coding algorithms. These include

- Strategies for motion estimation and motion vector encoding, including consideration of prediction and update steps, bidirectional prediction, and update filtering, as well as combined estimation over different levels of the temporal wavelet tree.
- Application and optimization of nonblock-based motion compensation, which is more natural used in combination with spatial wavelet decomposition.
- Scalability of motion information.

- Optimum adaptation of the spatial/temporal decomposition trees, including consideration of integrated solutions of spatial/temporal filtering.
- Optimization of spatial/temporal encoding, including psychovisual properties.
- Rate–distortion optimum truncation of scalable streams, including the trade-offs at various rates.
- Creation of complexity-scalable video coding bit streams.

BIBLIOGRAPHY

[1] G. Karlsson and M. Vetterli. "Subband coding of video signals for packet switched networks," *Proc. Visual Commun. Image Process.*, SPIE vol. 845, pp. 446–456, 1987.

[2] T. Kronander. "Some aspects of perception based image coding," *Ph.D. Thesis*, Linköping University, 1989.

[3] J.-R. Ohm. "Three-dimensional subband coding with motion compensation," *IEEE Trans. Image Processing*, no. 3, pp. 559–571, 1994.

[4] S.-J. Choi and J. W. Woods. "Motion compensated 3D subband coding of video," *IEEE Trans. Image Processing*, vol. 8, pp. 155–167, 1999.

[5] B. Pesquet-Popescu and V. Bottreau. "Three-dimensional lifting schemes for motion-compensated video compression," *Proc. IEEE ICASSP*, pp. 1793–1796, Salt Lake City, USA, May 2001.

[6] H. Radha, M. van der Schaar, and Y. Chen. "The MPEG-4 Fine-Grained Scalable Video Coding Method for Multimedia Streaming over IP," *IEEE Trans. on Multimedia*, vol. 3, Issue 1, pp. 53–68, March 2001.

[7] W. Li. "Overview of Fine Granularity Scalability in MPEG-4 video standard," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 301–317, 2001.

[8] U. Horn, K. W. Stuhlmüller, M. Link, and B. Girod. "Robust internet video transmission based on scalable coding and unequal error protection," *Signal Processing: Image Commun.*, September 1999.

[9] K. Rose and S. L. Regunathan. "Toward optimality in scalable predictive coding," *IEEE Trans. Image Processing*, vol. 10, pp. 965–976, July 2001.

[10] M. van der Schaar and H. Radha. "Adaptive Motion-Compensation Fine-Granularity-Scalability (AMC-FGS) for Wireless Video," *IEEE Trans. on Circuits and Systems for Video Technology*, June 2002.

[11] M. van der Schaar and D. S. Turaga. "Unconstrained motion compensated temporal filtering framework for wavelet video coding," *Proc. IEEE ICASSP*, May 2003.

[12] F. Wu, S. Li, and Y.-Q. Zhang. "DCT-prediction based progressive fine granularity scalability coding," *Proc. IEEE ICIP 2000*, Vancouver, Canada, vol. 3, pp. 556–559, September 10–13, 2000.

[13] F. Wu, S. Li, and Y.-Q. Zhang. "A framework for efficient progressive fine granularity scalable video coding," *IEEE Trans. Circuits and Systems for Video Technology*, special issue on streaming video, vol. 11, no. 3, pp. 332–344, 2001.

[14] H. Huang, C. Wang, and T. Chiang. "A robust fine granularity scalability using trellis-based predictive leak," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 12, no. 6, pp. 372–385, 2002.

[15] A. Fuldseth and T. A. Ramstad. "Robust subband video coding with leaky prediction," in *Proc. DSP Workshop*, Loen, Norway, pp. 57–60, September 1996.

[16] Y. Gao and L.-H. Chau. "An Efficient Fine Granularity Scalable Coding Scheme Using Adaptive Leaky Prediction," *Fourth International Conference on Information, Communications & Signal Processing and Fourth Pacific-Rim Conference on Multimedia (ICICS-PCM 2003)*, Singapore, December 2003.

[17] X. Sun, F. Wu, S. Li, W. Gao, and Y.-Q. Zhang. "Macroblock-based progressive fine granularity scalable video coding," *Proc. IEEE ICME*, Japan, August 2001.

[18] Feng Wu, Shipeng Li, Xiaoyan Sun, Bing Zeng, and Ya-Qin Zhang. "Macroblock-based progressive fine granularity scalable video coding," *International Journal of Imaging Systems and Technology*, vol. 13, Issue 6, pp. 297–307, 2003.

[19] W. H. Peng and Y.-K. Chen. "Mode-adaptive fine granularity scalability," *Proc. IEEE ICIP*, pp. 993–996, Greece, 2001.

[20] W.-H. Peng and Y.-K. Chen. "Enhanced Mode-Adaptive Fine Granularity Scalability," *International Journal of Imaging Systems and Technology*, vol. 13, no. 6, pp. 308–321, March 2004.

[21] L. Luo, J. Li, S. Li, Z. Zhuang, and Y.-Q. Zhang. "Motion compensated lifting wavelet and its application in video coding," *Proc. IEEE ICME*, Tokyo, Japan, August 2001.

[22] D. Turaga, M. van der Schaar, and B. Pesquet. "Complexity Scalable Motion Compensated Wavelet Video Encoding," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 15, no. 8, pp. 982–993, August 2005.

[23] J. Barbarien, Y. Andreopoulos, A. Munteanu, P. Schelkens, and J. Cornelis. "Coding of motion vectors produced by wavelet-domain motion estimation," *Proc. PCS 2003*, pp. 193–198, April 2003.

[24] C. Tillier and B. Pesquet-Popescu. "3D, 3-Band, 3-Tap Temporal Lifting for Scalable Video Coding," *Proc. IEEE ICIP 2003*, pp. 14–17, Barcelona, Spain, September 2003.

[25] C. Bergeron, C. Lamy-Bergot, G. Pau, and B. Pesquet-Popescu. "Temporal scalability through adaptive M-band filterbanks for robust H264/MPEG-4 AVC coding," Special issue on Video Analysis and Coding for Robust Transmission. *JASP*, 2006.

[26] C. Tillier, B. Pesquet-Popescu, and M. van der Schaar. "3-Band Temporal Structures for Scalable Video Coding," *IEEE Trans. on Image Processing*, vol. 15, Issue 9, pp. 2545–2557, September 2006.

[27] G. Pau and B. Pesquet-Popescu. "Comparison of Spatial M-Band Filter Banks for t+2D Video Coding," *Proc. of SPIE/IEEE VCIP 2005*, Beijing, China, July 2005.

[28] G. Pau, B. Pesquet-Popescu, and G. Piella. "Modified M-Band Synthesis Filter Bank for Fractional Scalability of Images," *IEEE Signal Processing Letters*, vol. 13, Issue 6, pp. 345–348, June 2006.

[29] M. Trocan, C. Tillier, and B. Pesquet-Popescu. "Joint wavelet packets for group of frames in MCTF," *Proc. SPIE*, San Diego, USA, July 2005.

[30] A. Secker and D. Taubman. "Highly Scalable Video Compression with Scalable Motion Coding," *IEEE Trans. on Image Processing*, vol. 13, no. 8, pp. 1029–1041, August 2004.

[31] R. Xiong, F. Wu, J. Xu, S. Li, and Y.-Q. Zhang. "Barbell lifting wavelet transform for highly scalable video coding," *Proc. Picture Coding Symposium 2004*, San Francisco, CA, USA, December 2004.

[32] Bo Feng, Jizheng Xu, Feng Wu, Shiqiang Yang, and Shipeng Li. "Energy distributed update steps (EDU) in lifting based motion compensated video coding," *Proc. IEEE ICIP 2004*, Singapore, October 2004.

[33] A. Said and W. A. Pearlman. "A new fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. on Circ. and Systems for Video Techn.*, vol. 6, pp. 243–250, June 1996.

[34] D. Taubman. "High performance scalable image compression with EBCOT," *IEEE Trans. on Image Processing*, no. 9, pp. 1158–1170, 2000.

[35] S.-T. Hsiang and J. W. Woods. "Embedded image coding using zeroblocks of subband/wavelet coefficients and context modeling," *MPEG-4 Workshop and Exhibition at ISCAS 2000*, Geneva, Switzerland, May 2000.

[36] J. Xu, S. Li, and Y.-Q. Zhang. "A wavelet video coder using three dimensional embedded subband coding with optimized truncation (3D ESCOT)," *IEEE-PCM 2000*, Sydney, December 2000.

[37] J. Xu, Z. Xiong, S. Li, and Y.-Q. Zhang. "3D embedded subband coding with optimal truncation (3D-ESCOT)," *Applied and Computational Harmonic Analysis* 10, p. 589, May 2001.

[38] S.-T. Hsiang and J. W. Woods. "Embedded video coding using invertible motion compensated 3D subband/wavelet filter bank," *Signal Processing: Image Commun.*, vol. 16, pp. 705–724, May 2001.

[39] S.-T. Hsiang, J. Woods, and J.-R. Ohm. "Invertible temporal subband/wavelet filter banks with half-pixel accurate motion compensation," *IEEE Trans. on Image Processing* 13, pp. 1018–1028, August 2004.

[40] M. van der Schaar and Y. Andreopoulos. "Rate-distortion-complexity modeling for network and receiver aware adaptation," *IEEE Trans. on Multimedia*, June 2005.

[41] Y. Wang, M. van der Schaar, S. F. Chang, and A. Loui. "Content-Based Optimal MDA Operation Prediction for Scalable Video Coding Systems Using Subjective Quality Evaluation," *IEEE Trans. on Circuits and Systems for Video Technology*—Special issue on Analysis and Understanding for Video Adaptation, October 2005.

[42] J. R. Ohm, M. van der Schaar, and J. Woods. "Interframe wavelet coding: Motion Picture Representation for Universal Scalability," *EURASIP Signal Processing: Image Communication*, Special issue on Digital Cinema, 2004.

[43] D. Turaga, M. van der Schaar, Y. Andreopoulos, A. Munteanu, and P. Schelkens. "Unconstrained motion compensated temporal filtering (UMCTF) for efficient and flexible interframe wavelet video coding," *EURASIP Signal Processing: Image Communication*, 2004.

[44] M. van der Schaar and H. Radha. "A hybrid temporal-SNR Fine-Granular Scalability for Internet Video," *IEEE Trans. on Circuits and Systems for Video Technology*, March 2001.

[45] H. Radha, M. van der Schaar, and S. Karande. "Scalable Video TranScaling for the Wireless Internet," *EURASIP Journal of Applied Signal Processing* (JASP)—Special issue on Multimedia over IP and Wireless Networks, No. 2, pp. 265–279, 2004.

[46] Y. Andreopoulos, M. van der Schaar, A. Munteanu, J. Barbarien, P. Schelkens, and J. Cornelis. "Fully-Scalable Wavelet Video Coding using In-Band Motion-Compensated Temporal Filtering," *Proc. on IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2003.

[47] J. Ye and M. van der Schaar. "Fully Scalable 3-D Overcomplete Wavelet Video Coding Using Adaptive Motion Compensated Temporal Filtering," *Proc. SPIE Video Communications and Image Processing (VCIP)*, 2003.

[48] Y. Andreopoulos, A. Munteanu, J. Barbarien, M. van der Schaar, J. Cornelis, and P. Schelkens. "In-band motion compensated temporal filtering," *EURASIP Signal Processing: Image Communication* (special issue on "Subband/Wavelet Interframe Video Coding"), vol. 19, no. 7, pp. 653–673, August 2004.

6

Scalable Audio Coding

Jin Li

6.1 INTRODUCTION

High-performance audio codecs bring digital music into practical reality. The most popular audio compression technology today is MP3 [8], which stands for layer III of the MPEG-1 audio compression standard. Developed in the early 1990s, MP3 does not perform very well in terms of compression efficiency. More advanced audio compression technologies have been proposed later, such as the MPEG-4 Advanced Audio Codec [1,10], Real Audio, and Windows Media Audio (WMA). The latter two are commercial audio coders developed by RealNetworks and Microsoft, respectively. Most existing audio codecs optimize only on a single target compression ratio, striving to deliver the best perceptual audio quality given the length of the bit stream or to deliver the shortest length of the bit stream given a constraint on playback quality. However, such a goal is far from enough, especially considering the unique characteristics of audio (as well as other media file) compression. Unlike data compression, where all content must be exactly preserved during the compression, audio compression is elastic and tolerates distortion. It is always possible to compress the audio a little more or a little less, with slightly more or less distortion. In fact, in many applications, it is difficult to foresee the exact compression ratio required at the time the audio is compressed. The ability to quickly change the compression ratio afterward has important applications and led to better user experience in audio storage and transmission. For example, if the compression ratio is adjustable, the compressed audio can be stretched to meet the exact requirements of the customer. We can build a stretchable audio recording device, which at first uses the highest possible compression quality (lowest possible compression ratio) to store the compressed audio. Later, when the length of the compressed audio at the highest quality exceeds the memory of the device, the compressed bit stream of the existing audio file can be truncated to leave memory for newly recorded audio content. A device with scalable audio compression technology can perform this stretch step again and again,

continuously increasing the compression ratio of the existing media, and freeing up storage space to squeeze in new content. As discussed in Chapter 4, the ability to quickly adjust the compression ratio is also very useful in the media communication/streaming scenario, where the server and the client may adjust the size of the compressed audio to match the instantaneous bandwidth and condition of the network, and thus reliably deliver the best possible media quality over the network. Moreover, multiple description coding [9] may also be obtained from a scalable coded audio bit stream. The idea is to apply more protection (using an erasure code with more parity packets) toward the head of the bit stream and to apply less protection toward the tail of the bit stream. Thus, with any number of packet losses, a prefix of the compressed bit stream is always preserved. As a result, the quality of the delivered audio may degrade gracefully with an increase in packet loss probability.

The straightforward way of adjusting the compression ratio of a compressed audio file is to first decode the compressed media file and then re-encode it. The computational complexity involved in the decoding and re-encoding operation can be quite costly. Moreover, there is usually a performance penalty involved, as the re-encoded audio is usually lower quality compared with directly encoding the audio file at the target compression ratio. Transcoding technologies have been developed to adjust the compression ratio of traditionally compressed bit streams, such as MP3 bit streams. Compared with decoding and then re-encoding, transcoding achieves modest computation savings by skipping some of the compression operations, mainly the inverse and forward transform and (for video transcoding) motion estimation. Almost all existing transcoding techniques still need to perform the entropy decoding and re-encoding; therefore, the speed of the transcoding is not very fast, usually at least 25% of that of media encoding.

In comparison, scalable/embedded coders allow the compressed bit stream to be directly manipulated. Popularized by Shapiro in his embedded zerotree wavelet (EZW) [12] image coder, embedded coder has the attractive property that the high compression ratio bit stream is embedded in the low compression ratio bit stream. Increasing the compression ratio can thus be done very quickly by extracting from a master bit stream the subset of the bit stream that corresponds to the application bit stream. In the case of embedded image compression, this operation can be further simplified to truncating the existing bit stream. In the domain of image compression, it has been shown [4,11,13] that embedded coding cannot only achieve flexible bit stream adjustment, but also obtain state-of-the-art compression performance and reasonable computational complexity. In fact, the most recent image compression standard, JPEG-2000 [14], defines an embedded image coder.

It is a misconception that you have to pay for the scalable functionality with compression performance. Just as embedded image coding did not take off until highly efficient bit plane entropy coding was developed, highly efficient em-

bedded audio coding needs unique technologies that suit its need for embedded coding. In this chapter, we develop an embedded audio coder (EAC) with performance that exceeds or matches that of the best available audio coders. The key technology that empowers EAC with such high performance is the use of implicit auditory masking and a high-performance subbit plane entropy coder.

6.2 SCALABLE AUDIO CODING FRAMEWORK

The embedded audio coder is a fully scalable audio waveform coder. There are three components of the EAC: an encoder, a decoder, and a parser. The encoder turns the input audio waveform into a compressed bit stream with the highest desirable bit rate, audio sampling rate, and number of audio channels. We call the compressed bit stream formed by the encoder the master bit stream, since all scaled bit streams (which we call application bit streams) will be formed by extracting subsets of bits from the master bit stream. The decoder turns the compressed bit stream, whether the master bit stream or the application bit stream, back into an audio waveform. The parser extracts a subset of the master bit stream to form an application bit stream with a reduced rate, reduced sampling rate, or reduced number of audio channels.

The framework of the EAC encoder is shown in Figure 6.1. The input audio waveform first goes through a channel mixer (MIX). Each channel of mixed audio is then separately transformed and quantized. After that, the transformed and quantized audio coefficients are split into sections, with each section of audio

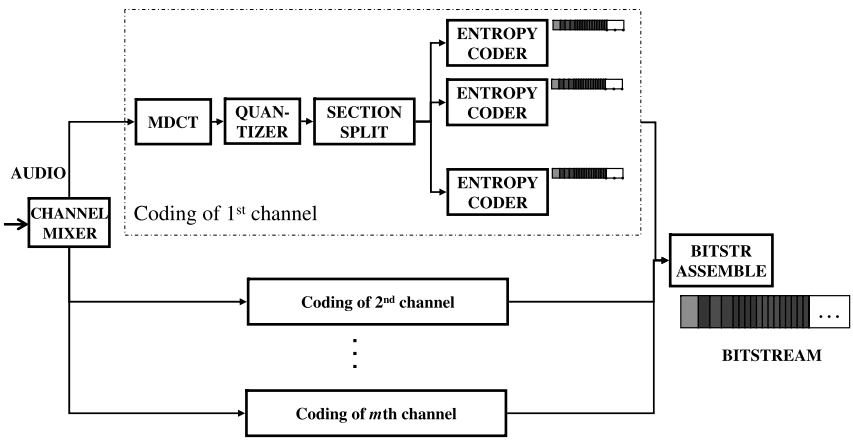


FIGURE 6.1: Embedded audio coding framework.

coefficients corresponding to one mixed channel of a particular time span and frequency range. Next, an embedded entropy coder is applied to each section to encode the coefficients into an embedded bit stream. Each section bit stream can be truncated after compression to trade distortion versus coding rate. Finally, a bit stream assembler puts together the embedded bit streams of the sections to form the master bit stream of the compressed audio.

Because the master bit stream is formed by concatenating together separately coded section bit streams, the master bit stream may be reshaped in a number of ways. In fact, the EAC parser takes the master bit stream as input and outputs an application bit stream with a possibly reduced bit rate, reduced number of audio channels, reduced sampling rate, or a combination of all. To scale by number of audio channels or audio sampling rate, the EAC parser simply drops the sections that are not needed any more. To scale by bit rate, the EAC parser further truncates the embedded bit stream of each section. The reshaped bit streams of needed sections are then put together to form the application bit stream.

The EAC decoder simply reverses the operation of the EAC encoder. It first demultiplexes the master bit stream or the application bit stream into a compressed bit stream for each section. Then, the compressed bit stream for each section is fed into a separate entropy decoder. The decoded coefficients are combined, inverse quantized, and transformed. Finally, a channel remixer recovers the playable audio waveform.

In the following, we will describe in detail the components of the EAC encoder: the channel mixer, the audio transform, the quantizer, the section splitter, the embedded entropy coder, and the bit stream assembler.

6.3 CHANNEL MIXER: SCALE BY NUMBER OF AUDIO CHANNELS

The channel mixer combines the input audio into a number of mixed channels. If the input audio is mono, the MIX simply passes through the audio. If the input audio is stereo, we may combine and mix the left (L) and right (R) audio channels into ch_1 and ch_2, as follows:

$$\begin{bmatrix} ch_1 \\ ch_2 \end{bmatrix} = \mathbf{M_1} \begin{bmatrix} L \\ R \end{bmatrix}, \tag{6.1}$$

where the mixing matrix $\mathbf{M_1}$ takes the form

$$\mathbf{M_1} = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}. \tag{6.2}$$

If the input audio has more than two channels, a multichannel mixer will be used to mix the multichannel audio. For example, the operation to mix six input audio

channels can be described as

$$\begin{bmatrix} ch_1 \\ ch_2 \\ ch_3 \\ ch_4 \\ ch_5 \\ ch_6 \end{bmatrix} = \begin{bmatrix} \mathbf{M}_1 & & 0 & \\ & 1 & & 0 \\ 0 & & \ddots & \\ & 0 & & 1 \end{bmatrix} \mathbf{M}_2 \begin{bmatrix} L \\ R \\ C \\ LS \\ RS \\ LFE \end{bmatrix}, \tag{6.3}$$

where the six input channels, Left Front (L), Right Front (R), Center Front (C), Low Frequency Enhancement (LFE), Left Surround (LS), and Right Surround (RS), are mixed into six output channels ch_i, in which i denotes the ith mixed channel. The matrix $\mathbf{M}_1$ is the stereo mixing matrix in (6.2), and $\mathbf{M}_2$ is the multichannel to stereo fold-down matrix,

$$\mathbf{M}_2 = \begin{bmatrix} 1 & & \alpha & \beta & & \chi \\ & 1 & \alpha & & \beta & \chi \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix},$$

where α, β and χ are constant fold-down parameters. Both mixing matrices $\mathbf{M}_1$ and $\mathbf{M}_2$ have the desirable properties that a small set of mixed audio channels may represent a scaled-down representation of the original multichannel audio. For example, ch_1 represents the audio component $L + R$ and may serve as a good mono representation of the stereo audio, should the playback device only support mono playback. ch_1 and ch_2 form a scaled-down stereo representation of the six-channel input audio. The MIX operation thus ensures that the compressed bit stream can be scaled by audio channels.

6.4 AUDIO TRANSFORM

After channel mixing, the waveform of each mixed audio channel is transformed by a modified discrete cosine transform (MDCT) or a modulated lapped transform (MLT) [5]. We switch the MDCT window between a long and a short window. The long window is used for homogeneous audio segments, while the short window is used for audio segments with large energy fluctuations to reduce the effect of pre-echoing. Assuming the input audio is sampled at 44.1 kHz, the size of the long MDCT window (W_l) is 2048 samples, while the size of the short MDCT window (W_s) is 256 samples.

The MDCT is defined

$$X(m) = \sqrt{\frac{2}{N}} \sum_{k=0}^{2N-1} w(k)x(k) \cos \frac{(2k+1+N)(2m+1)\pi}{4N}, \quad m = 0, \ldots, N-1, \tag{6.4}$$

where N is the length of the MDCT window (W_l or W_s), $X(m)$ is the value of the MDCT coefficient, $x(k)$ is the input audio samples, and $w(k)$ is the window function. The MDCT can be decomposed into two operations: windowing and time domain aliasing (TDA) and the discrete cosine transform of type IV (DCT-IV).

The windowing/TDA operation takes the form

$$\begin{pmatrix} x(k) \\ x(N-1-k) \end{pmatrix} \mapsto \begin{pmatrix} w(N-1-k) & w(k) \\ -w(k) & w(N-1-k) \end{pmatrix} \begin{pmatrix} x(k) \\ x(N-1-k) \end{pmatrix}, \quad \text{for } k = 0, \ldots, N/2-1, \tag{6.5}$$

where $w(k)$ is a window function that fulfills the time domain aliasing cancellation (TDAC) condition

$$w(k)^2 + w(N-1-k)^2 = 1, \quad k = 0, \ldots, N/2-1. \tag{6.6}$$

One of the most widely used window functions in audio compression is the Sine window,

$$w(k) = \sin\left[\frac{\pi}{N}\left(k + \frac{1}{2}\right)\right]. \tag{6.7}$$

Another popular window is the Kaiser–Bessel Derived (KBD) window, which does not have an analytic expression.

After the windowing/TDA operation, the DCT-IV is applied:

$$X(m) = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} x(k) \cos \frac{(2k+1)(2m+1)\pi}{4N}, \quad m = 0, \ldots, N-1. \tag{6.8}$$

The DCT-IV can be implemented by a prerotation, an FFT, and a postrotation.

The operation of the MDCT with switching window is depicted in Figure 6.2. Each channel of mixed audio is separated into frames of length W_l (the size of the long window). Each frame can be occupied by a single long window or can be split into W_l/W_s (in the default configuration, 8) short windows. If two consecutive frames are both long windows, a long window TDA operation is applied between the frames. Between two short windows, or between a long and a short window, a short window TDA operation is applied. After the TDA operation, a DCT-IV operation is applied to the signal.

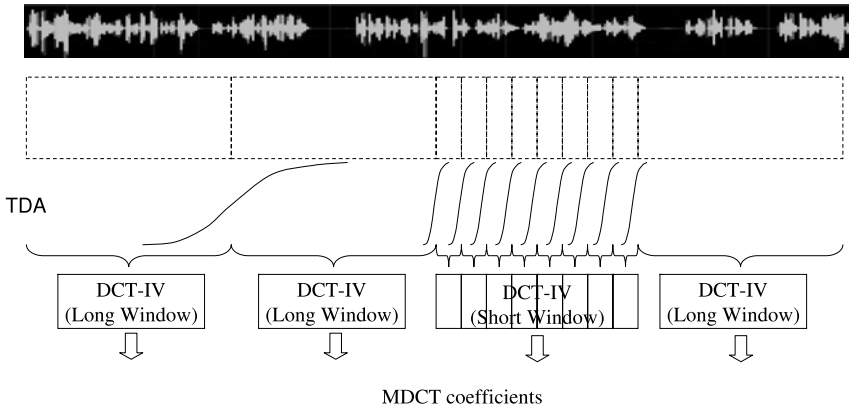


FIGURE 6.2: MDCT with a switching window.

The MDCT operation can be easily inverted, as both the TDA and the DCT-IV operation can be inverted. The inverse TDA operation is

$$\begin{pmatrix} x(k) \\ x(N-1-k) \end{pmatrix} \mapsto \begin{pmatrix} w(N-1-k) & -w(k) \\ w(k) & w(N-1-k) \end{pmatrix} \begin{pmatrix} x(k) \\ x(N-1-k) \end{pmatrix}, \quad \text{for } k = 0, \ldots, N/2-1, \tag{6.9}$$

and the inverse DCT-IV operation is

$$x(k) = \sqrt{\frac{2}{N}} \sum_{m=0}^{N-1} X(m) \cos \frac{(2k+1)(2m+1)\pi}{4N}, \quad k = 0, \ldots, N-1. \tag{6.10}$$

6.5 QUANTIZATION AND SECTION SPLIT

After the MDCT transform, all MDCT coefficients are uniformly quantized according to the rule

$$q(m) = sign\big(X(m)\big) \left\lfloor \frac{|X(m)|}{\delta} \right\rfloor, \tag{6.11}$$

where $X(m)$ is an MDCT coefficient, $q(m)$ is the quantization result, δ is the quantization step size, $sign(x)$ returns the sign of the coefficient x, and $\lfloor x \rfloor$ denotes the largest integer that is less or equal than x. The quantization process is conventional: uniform with a central dead zone twice the size of the quantization step size δ. However, the quantizer does not determine the ultimate quality of the

encoded audio. Because the quantized coefficients will be encoded by a subbit plane-embedded entropy coder with a truncatable bit stream, additional distortion can be introduced by the entropy coding module and the bit stream assembler module. Thus, the main functionality of the quantization module is to map the coefficients from a floating point representation to an integer representation so that they can be more efficiently processed by the entropy coding module. The default quantization step size in EAC is rather fine, for example, $\delta = 1/128$.

To improve the efficiency of the entropy coder, we group the quantized coefficients of a certain number of consecutive frames into a time slot. In the default configuration, a time slot consists of 16 frames, that is, 16 long MDCT windows or 128 short windows. A time slot therefore consists of 32,768 samples, which is about 0.74 second if the input audio is sampled at 44.1 kHz.

For sampling rate scalability, we may further split the coefficients in the time slot into a number of sections, each section covering the coefficients of a particular frequency range. For example, for a possible $2\times$ and $4\times$ sampling rate reduction, we split the coefficients into three sections of 0–0.25π, 0.25π–0.50π, and 0.50π–1.00π. By throwing away the coefficients corresponding to 0.50π–1.00π, and inversely transforming by a MDCT with a half window size for both long and short MDCT windows, we can decode the bit stream into audio with a $2\times$ sampling rate reduction. Similarly, by throwing away the coefficients corresponding to 0.25π–0.50π and 0.50π–1.00π, and inversely transforming by an MDCT with a quarter window size, we can decode the audio with a $4\times$ sampling rate reduction. Such an audio sampling rate reduction can be considered as passing the audio waveform through a low-pass filter that first transforms the audio by MDCT, throws away half (or three-quarters) of the coefficients, and then inversely transforms the coefficients with an MDCT at half (or quarter) window size. It provides an effective means of sampling rate reduction of the compressed audio, which is very useful if the decoding device does not have a good high frequency response or it wants to save computational power.

6.6 EMBEDDED SUBBIT PLANE ENTROPY CODING

The section of the quantized coefficients in a time slot is encoded by an embedded subbit plane entropy coder, which is one of the most complicated components in EAC. We will explain in detail the working of the subbit plane entropy coder in the following. First, we review the human auditory system in Section 6.6.1. Then we explain the implicit auditory masking approach in Section 6.6.2. We discuss the embedded coding unit (ECU) and the subbit plane entropy coder in Sections 6.6.3 and 6.6.4, respectively. We describe the arithmetic entropy coding unit in Section 6.6.5.

6.6.1 Human Auditory Masking

A detailed description of the human auditory system is beyond the scope of this chapter. The interested reader may refer to [7]. However, it is worth noticing that the characteristic of the human auditory system that most affects audio compression is auditory masking.

The human auditory system can be roughly divided into 26 critical bands, each of which is a bandpass filter bank with bandwidth on the order of 50 to 100 Hz for bands below 500 Hz and up to 5000 Hz for bands at high frequencies. Within each critical band, there is an auditory masking threshold, also referred to as the psychoacoustic masking threshold or the threshold of the just noticeable distortion (JND) [2]. Audio waveforms with an energy level below the JND threshold will not be audible. The auditory JND threshold is highly correlated to the spectral envelope of the signal. This is in contrast to the JND threshold in the human visual system, where the masking of a weak visual signal by a nearby strong signal occurs only over a very short range, and the dominant visual sensitivity is the same for a certain frequency regardless of the input signal. Let the auditory JND threshold of a critical band k at time i be $TH_{i,k}$. The JND threshold can be calculated as the maximum of a quiet threshold and a masking threshold. The quiet threshold TH_ST_k dictates the sensitivity of the auditory system for critical band k without the presence of any strong audio signal. It can be calculated through an equal loudness curve, such as the Fletcher–Munson curve [7] shown as the solid line in Figure 6.3. According to the quiet threshold, the sensitivity

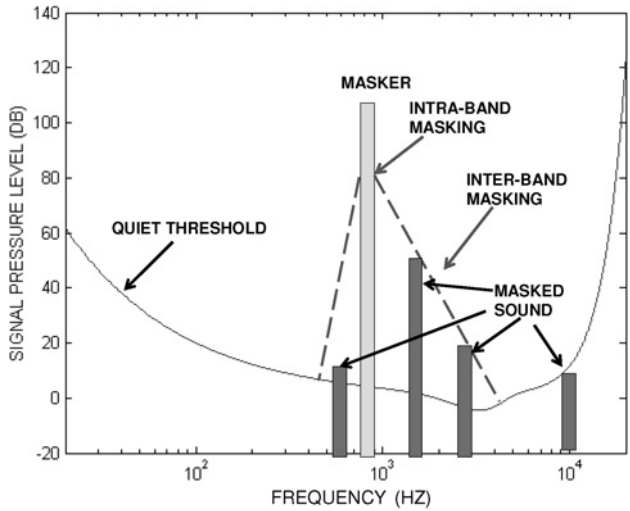


FIGURE 6.3: Auditory masking threshold: simultaneous masking.

of the ear is nearly constant for a large range (1–8 kHz) and drops dramatically before 500 Hz and after 10 kHz. Nevertheless, in audio compression, the auditory JND threshold is largely shaped by masking, which is an effect by which a low-level signal (the maskee) can be made inaudible by a simultaneously occurring strong signal (the masker) as long as the masker and the maskee are close enough to each other in time and frequency. The auditory masking threshold consists of three components: the simultaneous intra-band mask, the simultaneous inter-band mask, and the temporal mask. The most basic form of auditory masking is simultaneous intra-band masking, where the maskee and the masker are at the same time instant and within the same critical band. The intra-band masking threshold $TH\_INTRA_{i,k}$ is directly proportional to the average spectral energy $AVE_{i,k}$ of the masker in critical band k at the same time instant i, and can be expressed in dB as

$$TH\_INTRA_{i,k}(\text{dB}) = AVE_{i,k}(\text{dB}) - R_{\text{fac}}, \tag{6.12}$$

where R_{fac} is a constant offset value determined through the psychoacoustic experimentation. The second form of masking is simultaneous inter-band masking, where the maskee and the masker are at the same time instant, but at neighboring critical bands. The level of such inter-band masking $TH\_INTER_{i,k}$ can be formulated as

$$TH\_INTER_{i,k} = max(TH_{i,k-1} - R_{\text{high}}, TH_{i,k+1} - R_{\text{low}}), \tag{6.13}$$

where R_{high} and R_{low} are the attenuation factors toward the high- and low-frequency critical bands, respectively. The higher frequency coefficients are more easily masked; thus the attenuation R_{high} is smaller than R_{low}. Combining quiet, intra- and inter-band auditory masking, the auditory masking threshold created by a strong audio signal identified as the "masker" is illustrated in Figure 6.3, where the auditory JND threshold is shown as the dashed line. Any signal below the JND threshold, for example, compression distortion, will not be audible by human ears.

The third form of masking is temporal masking, which dictates that a strong signal can also mask a weak signal in the same critical band, but in the immediate preceding or following time interval. The duration within which premasking applies is less than one-tenth that of the postmasking, which is in the order of 50 to 200 ms. The temporal masking threshold $TH\_TIME_{i,k}$ can be expressed as

$$TH\_TIME_{i,k} = max(TH_{i-1,k} - R_{\text{post}}, TH_{i+1,k} - R_{\text{pre}}), \tag{6.14}$$

where R_{pre} and R_{post} are the attenuation factors for the preceding and following time intervals, respectively. A sample temporal masking generated by a masker is shown in Figure 6.4.

The combined auditory JND threshold is the maximum of the quiet threshold, the simultaneous intra- and inter-band masking, and the temporal masking threshold,

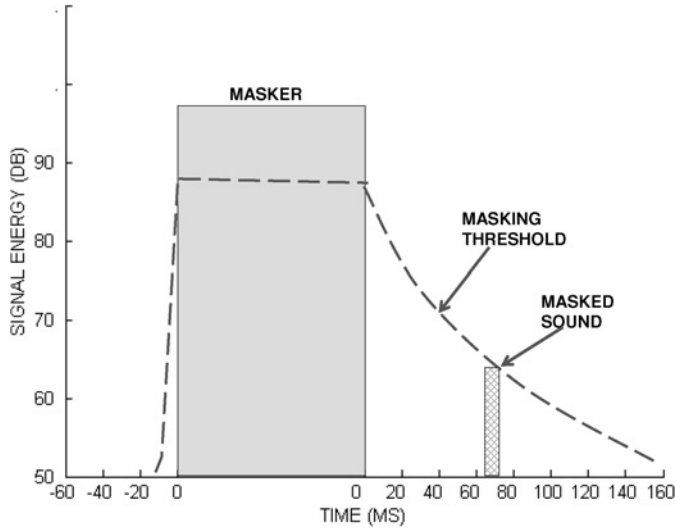


FIGURE 6.4: Auditory masking threshold: temporal masking.

$$TH_{i,k} = max(TH\_ST_k, TH\_INTRA_{i,k}, TH\_INTER_{i,k}, TH\_TIME_{i,k}). \quad (6.15)$$

Calculation of the JND threshold requires the iteration of (6.13)–(6.15). Thus, if the input audio consists of several strong maskers, the combined JND threshold will be the maximum of the masking threshold generated by the individual masks.

6.6.2 Implicit Auditory Masking

Using the auditory masking effect, an audio coder can devote fewer bits to the coefficients that are less sensitive to the human ear and more bits to the auditorily sensitive coefficients, thus improving the quality of the coded audio. In EAC, the auditory masking module is integrated with the embedded entropy coding module. It is done in a unique way with two distinctive features. First, the auditory JND threshold is derived from the partially coded coefficients and does not need to be transmitted. Second, the auditory JND threshold is used to determine the order that the bits of the coefficients are encoded, rather than to change the coefficients (by adopting a different quantizing step size for different critical bands). We call the approach implicit auditory masking because the auditory JND threshold is implicitly derived during the coding process.

To illustrate this distinctiveness, we show the process of encoding using traditional auditory masking in Figure 6.5 and that of the implicit auditory masking

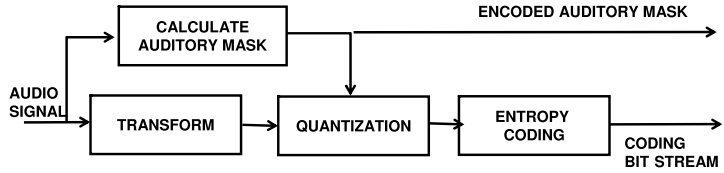


FIGURE 6.5: Encoding using traditional auditory masking.

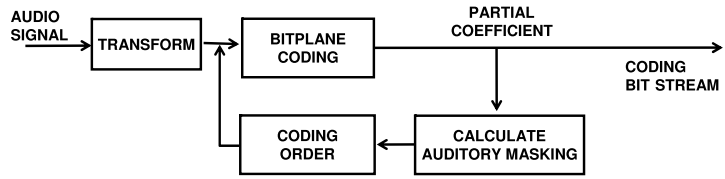


FIGURE 6.6: Encoding using implicit auditory masking.

in Figure 6.6. In traditional auditory masking, the encoder calculates the JND threshold based on the spectral envelope of the input audio waveform. The JND threshold is then encoded as a part of the compressed bit stream and is transmitted to the decoder. The encoder also quantizes the transform coefficients with a step size proportional to the JND threshold, that is, the coefficients are quantized coarsely in the critical bands with a larger JND threshold and are quantized finely in those with a smaller JND threshold. The approach is simple and suits a nonscalable coder. In scalable audio coding, it is not efficient. First, sending the auditory JND threshold consumes a nontrivial number of bits, which can be as much as 10% of the total number of coded bits. Since the auditory masking module is applied before the entropy coding module, the JND threshold must be transmitted with the same precision regardless of the compression ratio. The JND threshold overhead thus eats significantly into the bit budget, especially if the compressed bit stream is later scaled to a low bit rate. Second, as shown in Section 6.6.1, the JND threshold is shaped by the energy distribution of the input audio, while the same energy distribution is revealed through the bit plane coding process of the embedded entropy coder. As a result, the information is coded twice, which wastes precious coding bits.

The framework of implicit auditory masking is shown in Figure 6.6. Compared to Figure 6.5, the auditory masking operation is now integrated into the loop of the entropy coding module and is performed as follows. We first set the initial auditory JND threshold to the quiet threshold. A portion of the transform coefficients, for example, the top bit planes, is then encoded. Afterward, an updated auditory JND threshold is calculated based on the spectral envelope of the partially

coded transform coefficients. Since the decoder may derive the same auditory JND threshold from the same coded coefficients, the values of the auditory JND threshold need not be sent to the decoder. Using this implicitly calculated JND threshold, both the encoder and the decoder figure out which portion of the transform coefficients is to be encoded next. After the next portion of the coefficients has been encoded, the auditory JND threshold is updated again, which is then used to guide the coding order of the remaining portion of the coefficients. The process iterates among the operation of sending a portion of the quantized MDCT coefficients, updating the JND threshold, and using the updated JND threshold to determine the portions to be sent next. It only stops when a certain end criterion has been met, for example, the quantized coefficients have been encoded to the least significant bit plane (LSB), a desired coding bit rate has been reached, or a desired coding quality has been reached. By deriving the auditory masking threshold implicitly from the partially coded coefficients, bits normally required for the auditory JND threshold are saved. The saving can be especially significant at a low bit rate or when the coding bit stream is later truncated to a lower bit rate. Implicit auditory masking may thus significantly improve compression efficiency. Moreover, in all existing audio coders, the auditory JND threshold is carried as a header in the bit stream. In contrast, implicit auditory masking does not have an error-sensitive header. The EAC-compressed bit stream is thus less susceptible to transmission errors and therefore offers better error protection in a noisy channel, such as in a wireless environment. A third advantage of implicit auditory masking results from the fact that instead of coding the auditorily insensitive coefficients coarsely, the EAC encodes them at a later stage. By using auditory masking to govern the coding order, rather than to quantize the coefficients, the quality of the compressed audio becomes less sensitive to the accuracy of the JND threshold, as slight deviations in the threshold simply cause certain audio coefficients to be coded later.

6.6.3 Embedded Coding Unit

The section of quantized coefficients in a time slot is ultimately encoded by a subbit plane entropy coder. It encodes the audio coefficient bit by bit, and in a rate-distortion optimized order.

The subbit plane entropy coder of EAC is a general version of the simple bit plane coder, which works as follows. Let i index the time interval, j index the frequency component, and k index the critical band. Let $x_{i,j}$ be a coefficient at time interval i, frequency j, and $s_{i,k}$ be a critical band k at time interval i. Let each audio coefficient be represented in binary sign and magnitude form as

$$[\pm b_{L-1} b_{L-2} \cdots b_0], \tag{6.16}$$

where b_{L-1} is the most significant bit (MSB), b_0 is the least significant bit (LSB), and $\pm$ is the sign of the coefficient. A group of bits of the same significance

from different coefficients forms a bit plane. For example, bits b_{L-1} of all coefficients form the most significant $L-1$ bit plane. The bit plane coder encodes the coefficients bit plane by bit plane: first the most significant bit plane, then the second most significant bit plane, and so on. This way, if the output-compressed bit stream is truncated, at least part of each coefficient can be decoded.

The subbit plane coder in EAC goes one step further in recognizing that bits in the same bit plane can be different in their rate and distortion contributions. First, the coefficients represented by the bits may have different JND thresholds that lead to vastly different subjective distortions even if the objective distortions are the same. Second, the bits can be statistically different considering their neighbor coefficients and coding histories. An illustration of subbit plane is shown in Figure 6.7. Since the coefficients in EAC are actually arranged in a 2D array indexed by the time interval i and frequency j, the actual bit array is 3D. However, it is difficult to draw a 3D bit array; therefore, we show a slice of the bit array in 2D in Figure 6.7. Note that the sign of the coefficient is also part of the bit array, as the 'plus' and 'minus' signs can be represented by 0 and 1, respectively. Let b_M be a bit in a coefficient x, which is to be encoded. If all more significant bits in the same coefficient x are 0s, the coefficient x is said to be insignificant (because if the bit stream is terminated right before bit b_M has been coded, coefficient x will be reconstructed as zero), and the current bit b_M is to be encoded in the mode of significance identification. Otherwise, the coefficient is said to be significant, and the bit b_M is to be encoded in the mode of refinement. We distinguish between significance identification and refinement because a significance identification bit

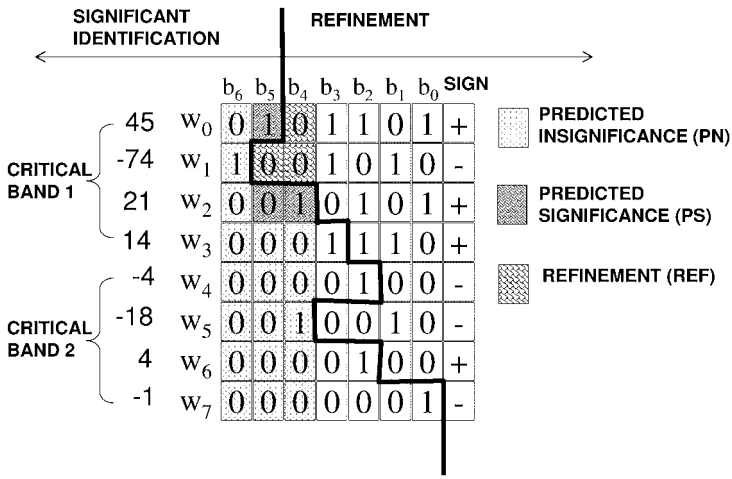


FIGURE 6.7: Subbit plane-embedded entropy coding.

has a very high probability of being 0, while a refinement bit is usually equally distributed between 0 and 1. The sign of the coefficient only needs to be encoded immediately after the coefficient turns significant, that is, a first nonzero bit in the coefficient is encoded. For the bit array in Figure 6.7, the significance identification and the refinement bits are separated by a solid bar. For a critical band $s_{i,k}$, we call the band insignificant if all the coefficients in the critical band are insignificant. It becomes significant when at least one coefficient is significant. EAC defines three subbit planes in a bit plane: the predicted significance (PS), the refinement (REF), and the predicted insignificance (PN). The PS subbit plane consists of bits of coefficients that are insignificant but has at least one neighbor known to be significant. The REF subbit plane consists of bits of coefficients that are already significant, that is, in the refinement mode. The PN subbit plane consists of bits of coefficients that are insignificant with no neighbors known to be significant. The subbit plane design is motivated by previous work on image coding [4] and the JPEG 2000 standard [14], which show that bits in different subbit planes contribute different decreases in average distortion per coding bit spent. For the sample bit array in Figure 6.7, we show the subbit plane types with different shades for the first three bit planes of the bit array.

We call a subbit plane of a critical band as an embedded coding unit (ECU). ECU is the smallest unit in the EAC reordering operation. The coding orders of ECUs are determined by the instantaneous JND threshold of the critical band. First, the initial auditory JND thresholds are calculated by using the quiet threshold. Using the initial threshold, the coding order of the ECUs is determined, and a set of high-priority ECUs is encoded. After a number of ECUs have been encoded or after a certain update interval, the auditory JND threshold is recalculated by both the encoder and decoder based on the partially coded coefficients at the moment. The updated JND threshold is then used to determine the formation and the coding order of the remaining ECUs. The process iterates until a certain end condition is met.

Note that we deliberately chose to update the JND threshold infrequently rather than updating after the encoding of one ECU or even after encoding one bit. This is in order to reduce the computational cost required of updating the JND threshold. Because in EAC, a slightly outdated JND threshold only leads to a slightly nonoptimal coding order of the ECUs, its impact on compression performance is minimal.

We mark the identity of each ECU by the critical band the ECU resides in and an ID that identifies the subbit plane. The ID is a rational number whose integer part is just the bit plane index and whose fractional part is assigned according to the subbit plane class. Currently, the PS, REF, and PN subbit planes are assigned with fractional values 0.875, 0.125, and 0.0, respectively. As an example, the ID of the PS subbit plane of bit plane 7 is 7.875. The fractional value is designed with the consideration of the average rate-distortion contribution of each subbit

plane class. Within each critical band, EAC encodes the ECUs according to the descending order of their IDs. For a critical band with a total of L bit planes, the first ECU to be encoded is the PN subbit plane of bit plane $L-1$ (ID: $L-1.0$) because all coefficients are insignificant at bit plane $L-1$. The next three subbit planes to be encoded are the PS (ID: $L-1.125$), REF (ID: $L-1.875$), and PN (ID: $L-2.0$) subbit planes of bit plane $L-2$. Subsequently, the subbit planes of bit plane $L-3$ are encoded. With the order of ECUs within a critical band already determined, the implicit auditory masking process only needs to determine the order of the ECUs among different critical bands. Conveniently, this can be done by determining the critical bands whose ECUs are next in line to be coded. We assign two important properties to each critical band: an instantaneous JND threshold and a progress indicator. The instantaneous JND thresholds are based on the partially reconstructed coefficient values of already coded coefficients, and the progress indicator records the ID of the next ECU to be encoded. It is the gap between the progress indicator and the instantaneous JND threshold that determines the coding order of ECUs. The coding process of the subbit plane entropy coder with implicit auditory masking can thus be described as follows.

1. Initialization.
   The maximum bit plane L of all coefficients is calculated. The progress indicators of all critical bands are set to the PN subbit plane of bit plane $L-1$ (with ID: $L-1$). The initial instantaneous JND threshold of each critical band is set according to the quiet threshold. We also mark all critical bands as insignificant.
2. Finding the current gap.
   For each critical band, we calculate a gap between its progress indicator and the instantaneous JND threshold. The gap is closely related to the level of the coding noise over the auditory JND threshold, the noise-to-mask ratio (NMR). The largest gap among all critical bands is defined as the current gap. The value of the current gap can be negative, which simply means that the coefficients with signal energy level below the auditory JND threshold are encoded. It can be easily proven that the instantaneous JND threshold is monotonically increasing and the progress indicator is monotonically decreasing. Therefore, the current gap shrinks in every iteration.
3. Encoding all critical bands with gap equal to the current gap.
   We encode all critical bands with gap value the same as the current gap in this iteration. Such a process leads to the encoding of the ECUs with the largest reduction of NMR per coding bit spent. This encoding step may further consist of the following substeps.
   (a) Critical band skipping.
       If a chosen critical band is insignificant (not a single coefficient is significant), a status bit is encoded to indicate whether the critical

band turns significant after the coding of the current bit plane. This is an optional step. However, it speeds up the coding/decoding operation significantly, as large areas of zero bits can be skipped with this step.

(b) Encoding the ECU of the critical band.
We locate the ECU, that is, the subbit plane that is next in line to be coded for each critical band. For each bit in the subbit plane, its context is calculated and the string of bit and context pairs are then compressed by a modern context adaptive entropy coder. The process of context calculation and subbit plane entropy coding is detailed in Section 6.6.4.

(c) Moving the progress indicator.
After the subbit plane is encoded, the progress indicator moves forward to the ID of the next subbit plane to be encoded.

4. Recording rate-priority points.
After all critical bands of the current gap have been encoded, we record the current coding rate R_i and the current gap S_i. These will be used in the bit stream assembler stage for rate-distortion optimization.
5. Updating the instantaneous JND threshold.
The instantaneous JND thresholds of all critical bands are updated based upon the already coded ECUs. There are tricks so that the encoder and decoder can recalculate the JND thresholds very efficiently, using on average less than one arithmetic operation per coefficient. For details, please refer to [3].
6. Repeating steps 1–5.
The steps 1–5 are repeated until a certain end criterion is reached, for example, the desired coding bit rate/quality has been reached, or all bits in all coefficients have been encoded.

6.6.4 Subbit Plane Context Adaptive Entropy Coder

The significance identification bits, refinement bits, and sign bits are not statistically equivalent even within their own categories. Statistical analysis demonstrates that if an MDCT coefficient $x_{i,j}$ has a large magnitude, its neighboring coefficients in time and frequency may have a higher probability of having large magnitudes as well. Moreover, its frequency harmonics (at double and/or triple frequency) may have large magnitudes too. To account for such statistical differences, we entropy encode the significance identification bits, refinement bits, and sign bits with context, each of which is a number derived from the already coded coefficients in the neighborhood of the current coefficient. The bits within the same context are assumed to be independent identically distributed (i.i.d.). The subsequent entropy coding can then automatically gather statistics for bits within each context, that is, the probability of being one, and use the statistics for effi-

cient entropy coding. Such technique is called context adaptive entropy coding and is frequently used in modern image/audio/video coding systems.

We first describe the contexts for the refinement bits and sign bits because they are simpler. The context for the refinement bits depends on the significance status of the four immediate neighbor coefficients, which for coefficient $x_{i,j}$ are the coefficients with the same frequency but for the preceding ($x_{i-1,j}$) and following ($x_{i+1,j}$) time intervals, and coefficients for the same time interval but at lower ($x_{i,j-1}$) and higher ($x_{i,j+1}$) frequencies. The refinement context is formed according to Table 6.1. Depending on the number of bit planes after significance identification, we assign the refinement bit to one of three refinement coding context categories: 10, 11, and 12. If one of the four neighbor coefficients is unreachable as it falls out of the current time slot or the current section or belongs to a frame with different window size, it is considered insignificant.

To determine the context for sign coding, we calculate a horizontal sign count h and a vertical sign count v. We separate the four neighbor coefficients into two pairs, a horizontal pair ($x_{i,j-1}$ and $x_{i,j+1}$) and a vertical pair ($x_{i-1,j}$ and $x_{i+1,j}$). For each pair, the sign count is calculated according to Table 6.2. The expected sign and the context of sign coding can thus be further calculated according to Table 6.3. Depending on the sign and significance status of the neighbors, the sign bit is assigned with one of five context categories: 13, 14, 15, 16, and 17. The context for the refinement and sign coding is designed with reference to the context used in the JPEG 2000 standard [14]. However, the significance identification context is specially tailored for audio coding. To calculate the context of the significance identification bit, we not only use the significance status of the four neighbor coefficients, but use the significance status of the half-harmonics $x_{i,j/2}$ and the MDCT window size. The use of the half-harmonic frequency is due to the fact that most sound-producing instruments produce harmonics of the base tone. Therefore, there is a strong correlation among the coefficient and its harmonics. The context used for the significance identification can be found in Table 6.4. De-

Table 6.1: Context for the refinement bit.

Context	Description
10	Current refinement bit is the first bit after significance identification and there is at least one significant coefficient among the immediate four neighbors.
11	Current refinement bit is the first bit after significance identification and there is no significant coefficient in the immediate four neighbors.
12	Current refinement bit is at least two bits away from significance identification.

Table 6.2: Calculation of sign count.

Sign count	Description
-1	Both coefficients are negative significant or one is negative significant and the other is insignificant.
0	Both coefficients are insignificant or one is positive significant and the other is negative significant.
1	Both coefficients are positive significant or one is positive significant and the other is insignificant.

Table 6.3: Expected sign and context for sign coding.

Sign count	h	-1	-1	-1	0	0	0	1	1	1
	v	-1	0	1	-1	0	1	-1	0	1
Expected sign		$-$	$-$	$+$	$-$	$+$	$+$	$-$	$+$	$+$
Context		13	14	15	16	17	16	15	14	13

Table 6.4: Context for significance identification (S: significant, N: nonsignificant, *: arbitrary).

Context	MDCT window size	Significance status of coefficient			
		$x_{i,j-1}$	$x_{i-1,j}$	$x_{i+1,j}$	$x_{i,j/2}$
0	2048	N	N	N	N
1	2048	*	S	*	*
2	2048	S	N	*	*
3	2048	N	N	S	*
4	2048	N	N	N	S
5	256	N	N	N	N
6	256	*	S	*	*
7	256	S	N	*	*
8	256	N	N	S	*
9	256	N	N	N	S

pending on the significance status of the four neighbors and the half-harmonics, we classify the significance identification bit into one of 10 contexts: 0–9.

As a summary, a total of 18 contexts are used for embedded audio coefficient coding. Of these, there are 10 contexts for significance identification, 3 for refinement coding, and 5 for sign coding.

6.6.5 Context Adaptive Entropy Coder

Through the aforementioned process, the section of quantized audio coefficients is turned into a sequence of bits, each of which is attached with a context. All bits

associated with the same context are assumed to be independently and identically distributed (i.i.d.). Let the number of contexts be N, and let there be n_i bits in context i, within which the probability of the bits to take value 1 is p_i. Using classic Shannon information theory, the entropy of such a bit-context sequence can be calculated as

$$H = \sum_{i=0}^{N-1} n_i \big[-p_i \log_2 p_i - (1 - p_i) \log_2 (1 - p_i) \big]. \tag{6.17}$$

The task of the context entropy coder is thus to convert the sequence of bit-context pairs into a compact bit stream representation with length as close to the Shannon limit as possible. Several coders are available for such task. The coders used in EAC include the adaptive Golomb coder [6] and the QM coder. It is observed that the adaptive Golomb coder has about the same compression efficiency as the QM coder, with roughly the same complexity.

In the following, we describe the implementation of the QM coder with focus on three key aspects: general arithmetic coding theory, fixed point arithmetic implementation, and probability estimation.

The Elias Coder

The basic theory of the MQ coder can be traced to the Elias Coder, or recursive probability interval subdivision. The process can be graphically illustrated in Figure 6.8. Let $S_0 S_1 S_2 \cdots S_n$ be a series of bits that is sent to the arithmetic coder.

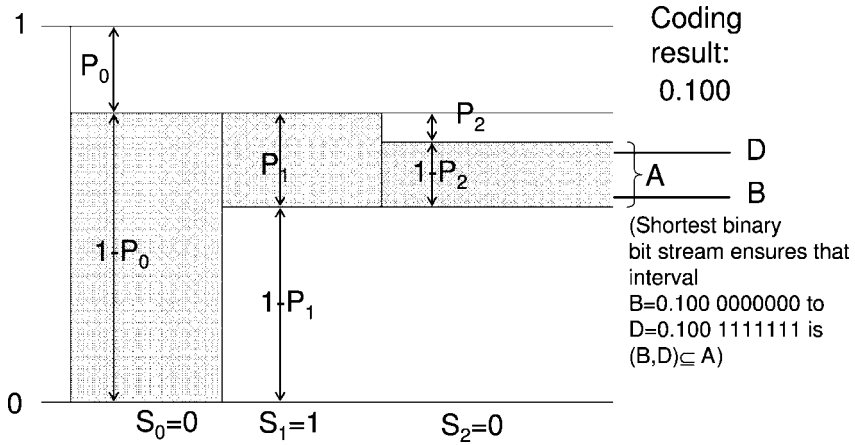


FIGURE 6.8: Elias coder: Probability interval subdivision.

Let P_i be the probability that the bit S_i is 1. We may form a binary representation (the coded bit stream) of the original bit sequence by the following process:

1. Initialization.
   Let the initial probability interval be (0.0, 1.0). We denote the current probability interval as $(C, C + A)$, where C is the bottom of the probability interval and A is the size of the interval. After initialization, we have $C = 0.0$ and $A = 1.0$.
2. Probability interval subdivision.
   The binary symbols $S_0 S_1 S_2 \cdots S_n$ are encoded sequentially. For each symbol S_i, the probability interval $(C, C + A)$ is subdivided into two subintervals $(C, C + A(1 - P_i))$ and $(C + A(1 - P_i), C + A)$. Depending on whether the symbol S_i is 1, one of the two subintervals is selected. That is,

$$\begin{array}{llll} C = C, & A = A(1 - P_i) & \text{for } S_i = 0, & \text{and} \\ C = C + A \cdot (1 - P_i), & A = A \cdot P & \text{for } S_i = 1. & \end{array} \tag{6.18}$$

3. Bit stream output.
   Let the final coding bit stream be referred to as $k_1 k_2 \cdots k_m$, where m is the compressed bit stream length. The final bit stream creates an uncertainty interval where the lower bound B and upper bound D can be expressed as

$$\begin{aligned} D &= 0.k_1 k_2 \cdots k_m 111 \cdots, \\ B &= 0.k_1 k_2 \cdots k_m 000 \cdots. \end{aligned} \tag{6.19}$$

   As long as the uncertainty interval (B, D) is contained in the probability interval $(C, C + A)$, the coding bit stream uniquely identifies the final probability interval, and thus uniquely identifies each subdivision in the Elias coding process. The entire binary symbol string $S_0 S_1 S_2 \cdots S_n$ can thus be recovered from the compressed representation. It can be shown that it is possible to find a final coding bit stream with length

$$m \leq \lceil -\log_2 A \rceil + 2 \tag{6.20}$$

   to represent the final probability interval $(C, C + A)$, where $\lceil x \rceil$ returns the smallest integer that is larger than or equal to x. Notice A is the probability of the occurrence of the binary strings $S_0 S_1 S_2 \cdots S_n$, and the entropy of the symbol stream can be calculated as

$$H = \sum_{S_0 S_1 \cdots S_n} -A \log_2 A. \tag{6.21}$$

   The arithmetic coder thus encodes the binary string within 2 bits of its entropy limit, no matter how long the symbol string is. This is very efficient.

The Arithmetic Coder: Finite Precision Arithmetic Operation

Exact implementation of Elias coding requires infinite precision arithmetic, an unrealistic assumption in real applications. Using finite precision, Elias coding becomes arithmetic coding. Observing the fact that the coding interval A becomes very small after a few operations, we may normalize the coding interval parameters C and A as

$$\begin{aligned} C &= 1.5 \cdot [0.k_1k_2 \cdots k_L] + 2^{-L} \cdot 1.5 \cdot C_x, \\ A &= 2^{-L} \cdot 1.5 \cdot A_x, \end{aligned} \tag{6.22}$$

where L is a normalization factor determining the magnitude of the interval A. The fixed-point integers A_x and C_x are fixed-point integers representing values between (0.75, 1.5) and (0.0, 1.5), respectively. Bits $k_1k_2 \cdots k_L$ are the output bits that have already been determined (in reality, certain carryover operations have to be handled to derive the true output bit stream). By representing the probability interval with the normalization L and fixed-point integers A_x and C_x, it is possible to use fixed-point arithmetic and normalization operations for the probability interval subdivision operation. Moreover, since the value of A_x is close to 1.0, we may approximate $A_x \cdot P_i$ with P_i. The interval subdivision operation (6.18) may thus be calculated

$$\begin{aligned} &C_x = C_x, && A_x = A_x - P_i && \text{for } S_i = 0, \quad \text{and} \\ &C = C + A_x - P_i, && A_x = P_i && \text{for } S_i = 1, \end{aligned} \tag{6.23}$$

which can be done quickly without any multiplication. The compression performance suffers a little, as the coding interval now has to be approximated with a fixed-point integer, and $A_x \cdot P_i$ is approximated with P_i. However, experiments show that the degradation in compression performance is less than 3%, which is well worth the saving in implementation complexity.

Probability Estimation

In the arithmetic coder it is necessary to estimate the probability P_i of being 1 for each binary symbol S_i. This is where context comes into play. Within each context, the symbols are coded as if they are independently distributed. We estimate the probability of the symbol within each context through observation of the past behaviors of symbols in the same context. For example, if we observe n_i symbols in context i, with o_i symbols being 1, we may estimate the probability that a symbol takes on the value 1 in context i through Bayesian estimation as

$$p_i = \frac{o_i + 1}{n_i + 2}. \tag{6.24}$$

In the QM coder, probability estimation is implemented through a state-transition machine. It may estimate the probability of the context more efficiently and may take into consideration the nonstationary characteristic of the symbol string. Nevertheless, the principle is still to estimate the probability based on past behavior of the symbols in the same context.

6.7 BIT STREAM ASSEMBLER

Finally, a bit stream assembler module allocates the available coding bits among the time slots, channels, and sections, and produces the final compressed bit stream.

Recall from Section 6.6.3 that each section of quantized MDCT coefficients is compressed separately into an embedded bit stream. We record a rate and a priority value each time the current gap shrinks. Let $R_{t,c,s,i}$ and $S_{t,c,s,i}$ be the rate and priority value for time slot t, channel c, section s, and truncation point i. The main functionality of the bit stream assembler module is thus to find the proper truncation point for each section of bit stream and to generate a combined bit stream.

The bit stream assembly module may operate in a number of modes. It may operate in distortion controlled mode. In this case, the user may define a desired NMR S_{desired} for the compressed bit stream. The bit stream assembler module then truncates all segments with distortion smaller than the desired NMR away, and leaves only those segments with distortion greater than threshold. The truncation point i for time slot t, channel c, and section s can be expressed as

$$i = \arg\max_{k}(S_{t,c,s,i} > S_{\text{desired}}). \tag{6.25}$$

The bit stream assembler may operate in rate controlled mode. The user defines a total bit rate R_{total} for the entire compressed audio file (for variable bit rate coding) or a bit rate R_s for each time slot (for constant bit rate coding). The bit stream assembler then searches for the right priority value S, which truncates the embedded section bit streams according to (6.25) and allows the total length of the truncated bit streams to stay just below the bit rate limit.

After the truncation points for each section are determined, the bit stream assembler combines the section bit streams into an EAC master bit stream. The bit stream syntax of the master bit stream is as follows. (It is also the syntax of the application bit stream that is derived from the master bit stream.) The EAC bit stream starts with a global header, which identifies the EAC bit stream and stores global codec information such as the parameters of the transform module and entropy coding module. The global header is then followed by the compressed bit streams of individual time slot. The time slot is again led by a header, which

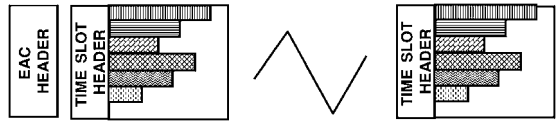


FIGURE 6.9: EAC bit stream syntax (master and application bit stream).

records the lengths of the compressed bit streams in the time slot, and the lengths of the compressed bit streams of the individual sections. The time slot header is then followed by the truncated embedded bit stream of each section of each audio channel of that time slot. The syntax of the EAC master bit stream is shown in Figure 6.9.

Finally, the bit stream assembler generates a companion file that holds the structural information of the EAC bit stream. In EAC, the companion file stores the rate $R_{t,c,s,i}$ and priority $S_{t,c,s,i}$ value pairs of all sections up to the truncation points. The companion file is not necessary for decoding an EAC bit stream or scaling the EAC bit stream by the number of audio channels or audio sampling rate. If the compressed audio is scaled by channels, the compressed bit streams of the unused channels are removed from the bit stream. If the compressed audio is switched into a lower sampling rate, the compressed bit streams of the sections corresponding to higher sampling rate are dropped. When the EAC bit stream is scaled by bit rate, the companion file is used. In this case, the EAC parser reads both the EAC master bit stream and the associated companion file. Then, the bit stream assembler is called upon to redetermine the truncation points of the bit streams of each section based on the new desired bit rate. An application bit stream of a different coding bit rate can thus be generated.

6.8 SUMMARY AND FURTHER READING

The main objective of this chapter is to get the reader familiar with scalable audio coding technology. We looked at the concepts, framework, and fundamental building blocks of scalable audio compression. Using EAC, an embedded audio coder, as an example, we provided a framework for fine grain scalable audio coding. We studied the individual building blocks of EAC, including the channel mixer, the audio transform, the quantization and section split unit, the auditory masking module, the embedded subbit plane entropy coder, and the bit stream assembler. Throughout the chapter, we explained how scalable audio coding differs from nonscalable audio coding in terms of technology used. This knowledge should aid the reader in building and using scalable audio coding technologies in his or her own work.

Scalable audio coding is a very attractive feature for audio storage and transmission. We gave a brief description on how scalable coded audio can be used in storage applications in Section 6.1. Scalable coding is one of the fundamental aspects of advanced media transmission technologies. Examples of using scalable coding to improve quality of service in media delivery can be found extensively in Chapters 2, 4, 9, 10, and 14 of this book.

REFERENCES

[1] G. Grill. MPEG-4 general audio coder. In *AES International Conference on High Quality Audio Coding*, Firenze, September 1999.

[2] N. Jayant, J. Johnston, and R. Safranek. Signal compression based on models of human perception. *Proceedings of the IEEE*, 81(10):1385–1422, 1993.

[3] Jin Li. Embedded audio coding (EAC) with implicit auditory masking. In *ACM Multimedia*, pages 592–601, 2002.

[4] Jin Li and Shawmin Lei. An embedded still image coder with rate-distortion optimization. *IEEE Transactions on Image Processing*, 8(7):913–924, 1999.

[5] H. S. Malvar. Lapped transforms for efficient transform/subband coding. *IEEE Trans. Acoust., Speech, Signal Processing*, 38(6):969–978, 1990.

[6] H. S. Malvar. Fast adaptive encoder for bi-level images. In *Data Compression Conference*, pages 253–262, 2001.

[7] B. C. J. Moore. *An Introduction to the Psychology of Hearing*. Academic Press, 5th edition, 2003.

[8] G. Noll. MPEG digital audio coding. *IEEE Signal Processing Magazine*, pages 59–81, September 2001.

[9] V. N. Padmanabhan, H. J. Wang, and P. A. Chou. Resilient peer-to-peer streaming. In *ICNP '03: Proceedings of the 11th IEEE International Conference on Network Protocols*, page 16, Washington, DC, USA, 2003. IEEE Computer Society.

[10] S. R. Quackenbush. Coding of natural audio in MPEG-4. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages VI:3797–3800, Seattle, Washington, 1998.

[11] A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, 1996.

[12] J. M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3463, December 1993.

[13] D. Taubman. High performance scalable image compression with EBCOT. *IEEE Transactions on Image Processing*, 9(7):1158–1170, 2000.

[14] D. S. Taubman and M. W. Marcellin. *JPEG 2000: Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

This page intentionally left blank

PART C

IP NETWORKING

This page intentionally left blank

7

Channel Protection Fundamentals

Raouf Hamzaoui, Vladimir Stanković, Zixiang Xiong, Kannan Ramchandran, Rohit Puri, Abhik Majumdar, and Jim Chou

7.1 INTRODUCTION

In many ways, the Internet (or a wireless network for that matter) can be regarded simply as a communication channel in a classical communication system. This chapter discusses the fundamentals of channel protection that lie beneath the error control techniques used to communicate multimedia over the Internet and wireless networks.

The goal of a classical communication system is to transfer the data generated by an information source efficiently and reliably over a noisy channel. The basic components of a digital communication system are shown in Figure 7.1: a source encoder, channel encoder, modulator, demodulator, channel decoder, and source decoder.

The source encoder removes the redundancy in the digital data produced by the information source and outputs an information sequence. If the information source is analog, its output must be digitized before it is processed by the source encoder. The channel encoder adds redundancy to the information sequence so that channel errors can be detected or corrected. The output of the channel encoder is a finite sequence of symbols called a channel codeword. The set of possible channel codewords is called a channel code. The modulator maps the channel codeword to a signal that is suitable for transmission over a physical channel. The demodulator converts the received signal into a discrete sequence of real numbers of the same length as the channel codeword. In hard decision decoding, each real number in the sequence is mapped to a channel code symbol before being processed by the channel decoder. When the real numbers are left unquantized or quantized to a number of levels that is greater than the size of the channel

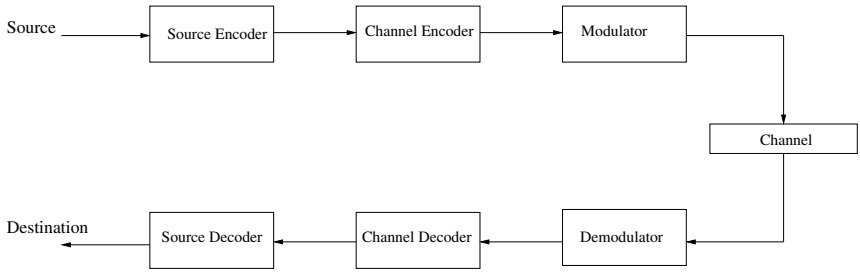


FIGURE 7.1: Basic components of a digital communication system.

code alphabet, one speaks of soft decision decoding. The channel decoder tries to recover the input to the channel encoder from the output of the demodulator. Finally, the source decoder produces an estimate of the information sequence.

In this chapter, we look in detail at the basic constituents of a communication system. In Section 7.2, we briefly explain the notions of entropy of an information source, mutual information, rate–distortion function, and capacity of a channel. These notions are needed to introduce four fundamental theorems due to Claude Shannon: the noiseless coding theorem [45], the source coding theorem [46], the channel coding theorem [45], and the source-channel coding theorem [46]. Our exposition mainly follows that of McEliece [17], where proofs of the theorems can be found. For simplicity, we focus on discrete memoryless sources and channels. References are provided for extensions and generalizations. Shannon's coding theorems give an insight on what can be achieved by a communication system. Unfortunately, the theorems are not constructive. Optimal codes were shown to exist, but it was not explained how to construct them. The remainder of the chapter is dedicated to practical system design. Section 7.3 contains an overview of the most important channel codes. Section 7.4 reviews state-of-the-art modulation techniques, focusing on a promising method called hierarchical modulation. Section 7.5 considers communication systems where feedback information can be sent from the receiver to the transmitter. In this situation, error control based on error detection and retransmission (automatic repeat request [ARQ]) can be more efficient than error correction alone (forward error correction [FEC]). We review the most important ARQ techniques and discuss hybrid methods that combine ARQ and FEC.

7.2 SHANNON'S SOURCE AND CHANNEL THEOREMS

An information source is given by a sequence of random variables X_n, each of which takes values in an alphabet A. We say that the source is *discrete* if the alphabet A is finite or countable. We say that the source is *memoryless* if the random

variables X_n are independent and identically distributed. For simplicity, a discrete memoryless source X_n will be denoted by a random variable X whose probability distribution $p(x)$ is the probability distribution common to all the random variables X_n.

Definition 7.1 (Entropy of a discrete memoryless source). Let X be a discrete memoryless source with alphabet A. The entropy of X is

$$H(X) = -\sum_x p(x) \log p(x),$$

where the sum is taken over all $x \in A$ for which $p(x) > 0$.

Any base of the logarithm can be used in the expression for the entropy. When this base is two, the entropy is measured in bits.

The entropy of a discrete memoryless source X measures the amount of uncertainty in the source. Since the entropy is completely defined by the probability distribution of X, we speak also of the entropy of the random variable X.

Given the discrete random variables $X, Y_1, \ldots, Y_n$, the *conditional entropy* $H(X \mid Y_1, \ldots, Y_n)$ is the average uncertainty remaining in X after observing $Y_1, \ldots, Y_n$. Formally, with $H(X \mid y_1, \ldots, y_n) = -\sum_x p(x \mid y_1, \ldots, y_n) \log p(x \mid y_1, \ldots, y_n)$, we set

$$H(X \mid Y_1, \ldots, Y_n) = \sum_{y_1, \ldots, y_n} p(y_1, \ldots, y_n) H(X \mid y_1, \ldots, y_n).$$

The difference $I(X; Y) = H(X) - H(X|Y)$ is known as the *mutual information* of the two random variables X and Y. It expresses the amount of information provided by Y on X.

7.2.1 Source Coding

A code $\mathcal{C}$ of size M over an alphabet B is a set of M words, possibly varying in length, made up of symbols from B. The words in $\mathcal{C}$ are called *codewords*. The length of a codeword $\mathbf{c}$ (i.e., the number of symbols in $\mathbf{c}$) is denoted $|\mathbf{c}|$. When all the codewords in $\mathcal{C}$ have the same length k, we say that $\mathcal{C}$ is a block code of length k.

7.2.1.1 Lossless Source Coding

To compress a discrete source X_n over alphabet A without information loss, one can use a lossless encoding scheme $(\mathcal{C}, f)$, which consists of a code $\mathcal{C}$ (known as

a source code) and an injective map $f : A \to \mathcal{C}$ (known as an encoder). A source code $\mathcal{C}$ is *uniquely decodable* if for all positive integers k and all codewords $\mathbf{c}_i, \mathbf{d}_i,\ i = 1, \ldots, k$, the equality $\mathbf{c}_1 * \cdots * \mathbf{c}_k = \mathbf{d}_1 * \cdots * \mathbf{d}_k$ implies that $\mathbf{c}_i = \mathbf{d}_i$ for all $i = 1, \ldots, k$. Here $*$ denotes concatenation.

The average codeword length of an encoding scheme $(\mathcal{C}, f)$ with respect to a discrete memoryless source X can be expressed $L = \sum_x p(x)|f(x)|$.

Shannon's noiseless coding theorem [45] says that the entropy of a discrete memoryless source X over alphabet A gives the smallest average number of code symbols (from the alphabet B) needed to losslessly represent one source symbol (from the alphabet A), when the base of the logarithm is $|B|$. Before stating the theorem more precisely, we must define the extension of a discrete source. The kth extension of a source X_n is the source $X_n^k = (X_{(n-1)k+1}, \ldots, X_{nk})$ obtained by blocking X_n into blocks of length k. Thus if X_n is a discrete memoryless source taking values in A, then X_n^k is a discrete memoryless source taking values in A^k, where A^k is the set of all words of length k over A.

Theorem 7.1 (Shannon's noiseless coding theorem). *Let X be a discrete memoryless source. Let X^k be its kth extension. Let L_k be the minimum average codeword length over all encoding schemes for X^k whose codes are uniquely decodable. Then*

$$H(X) \leq \frac{L_k}{k} \leq H(X) + \frac{1}{k}.$$

7.2.1.2 Lossy Source Coding

Suppose now that the symbols generated by a discrete memoryless source X over alphabet A are to be reproduced by symbols from a finite alphabet $\hat{A}$ called the reproducing alphabet. A single-letter distortion measure $d : A \times \hat{A} \to \mathbb{R}^+$ measures the distortion $d(x, y)$ when symbol $x \in A$ is reproduced as $y \in \hat{A}$. The distortion between a word $\mathbf{x} = (x_1, \ldots, x_k) \in A^k$ and a word $\mathbf{y} = (y_1, \ldots, y_k) \in \hat{A}^k$ is defined as $d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^k d(x_i, y_i)$.

A lossy compression scheme $(\mathcal{C}, k, M, f)$ for the memoryless source X is given by a block code $\mathcal{C}$ of length k and size M over the reproducing alphabet $\hat{A}$, and a mapping f from A^k to $\mathcal{C}$. This compression scheme allows us to represent any sequence of k source symbols with $\lceil \log_2 M \rceil$ bits. Thus the *rate* of a block code of length k and size M is defined to be $(\log_2 M)/k$ bits per symbol.

Example 1. Let $A = \{0, 1\}$, $\hat{A} = \{0, 1\}$, and $\mathcal{C} = \{00, 11\}$. Then the mapping $f : A^2 \to \mathcal{C}$ given by $f(00) = 00$, $f(01) = 00$, $f(10) = 11$, $f(11) = 11$ defines a lossy compression scheme $(\mathcal{C}, 2, 2, f)$ for the source X. By using the binary representation $00 \mapsto 0$ and $11 \mapsto 1$, any sequence of two source symbols can be represented by one bit.

The average distortion of the lossy compression scheme $(\mathcal{C}, k, M, f)$ with respect to the source X and the single-letter distortion measure d is

$$D(\mathcal{C}, k, M, f) = \frac{1}{k} \sum_{\mathbf{x} \in A^k} p(\mathbf{x}) d\big(\mathbf{x}, f(\mathbf{x})\big),$$

where for $\mathbf{x} = (x_1, \ldots, x_k)$, $p(\mathbf{x}) = \prod_{i=1}^{k} p(x_i)$.

Example 2. In Example 1, suppose that the distortion measure d is given by $d(x, y) = 0$ if $x = y$ and $d(x, y) = 1$, otherwise. Suppose also that $Pr\{X = 0\} = p(0) = p$, where $0 < p < 1/2$. Then the average distortion of the lossy compression scheme $(\mathcal{C}, 2, 2, f)$ is $p(1 - p)$.

Let Y be a random variable that is jointly distributed with X according to the joint probability distribution $p(x, y) = p(x)p(y|x)$ for some conditional probability distribution $p(y|x) = p_{Y|X}(y|x)$. As a function of the conditional probability distribution $p_{Y|X}$, the average distortion when X is reproduced as Y is thus

$$D(p_{Y|X}) = \sum_{x \in A, y \in \hat{A}} p(x)p(y|x)d(x, y).$$

Note that the smallest value that can be taken by $D(p_{Y|X})$ is

$$D_{\min} = \sum_{x \in A} p(x) \min_{y \in \hat{A}} d(x, y).$$

Definition 7.2 (Rate–distortion function). Let X be a discrete memoryless source over alphabet A, and let $\hat{A}$ be a reproducing alphabet. The rate–distortion function of the source X with respect to a single-letter distortion measure d : $A \times \hat{A} \to \mathbb{R}^+$ is the function

$$R(D) = \min_{p_{Y|X} : D(p_{Y|X}) \leq D} I(X; Y),$$

for all $D \geq D_{\min}$, where the minimum is taken over all conditional distributions $p(y|x)$ subject to the constraint $D(p_{Y|X}) \leq D$.

Henceforth we assume that the base of the logarithm in the mutual information is 2, so that $R(D)$ is measured in bits.

We can now state Shannon's source coding theorem [46].

Theorem 7.2 (Shannon's source coding theorem). *Let X be a discrete memoryless source over alphabet A, and let $\hat{A}$ be a reproducing alphabet. Let $R(D)$ be the rate–distortion function of the source X with respect to a single-letter distortion measure $d : A \times \hat{A} \to \mathbb{R}^+$. Then for any $D' > D$ and $R' > R(D)$, there exists a lossy compression scheme $(\mathcal{C}, k, M, f)$ such that $M \leq 2^{\lfloor kR' \rfloor}$ and $D(\mathcal{C}, k, M, f) < D'$.*

The theorem roughly says that one can reproduce the source symbols with an average distortion that is smaller than D by spending no more than $R(D)$ bits per source symbol.

7.2.2 Channel Coding

A discrete channel takes at every time unit n a symbol from a finite input alphabet A_X and outputs symbols from a finite output alphabet A_Y.

Let X_n be the random variable that gives the nth input symbol of the channel and let Y_n be the random variable that gives the nth output symbol of the channel. We say that the channel is memoryless if $Pr\{Y_n = y \mid X_n = x\} = p(y|x)$ is independent of n for all $y \in A_Y$ and $x \in A_X$, the transition probabilities $p(y|x)$ satisfy $p(y|x) \geq 0$ and $\sum_{y \in A_Y} p(y|x) = 1$, and $Pr\{Y_1 = y_1, \ldots, Y_n = y_n \mid X_1 = x_1, \ldots, X_n = x_n\} = \prod_{i=1}^{n} p(y_i|x_i)$ for all n.

Definition 7.3 (Channel capacity). The capacity of a discrete memoryless channel with input alphabet A_X, output alphabet A_Y, and transition probabilities $p(y|x),\ y \in A_Y, x \in A_X$ is

$$C = \max_{p_X} I(X; Y),$$

where X is a random variable that gives the input symbol to the channel and Y is a random variable that gives the corresponding output symbol according to the joint probability distribution $p(x, y) = p(x)p(y|x)$, for some source distribution $p(x) = p_X(x)$.

Since we assume that the base of the logarithm in the mutual information is 2, C is measured in bits.

Example 3. The binary symmetric channel (BSC) is a discrete memoryless channel with $A_X = A_Y = \{0, 1\}$ and transition probabilities $p(0|1) = p(1|0) = p$ and $p(0|0) = p(1|1) = 1 - p$. When a bit is sent over the BSC, it is either corrupted with probability p or correctly received with probability $1 - p$. It is easy to prove that the capacity of the BSC is $1 + p \log p + (1 - p) \log(1 - p)$ [15].

Example 4. The binary erasure channel (BEC) is a discrete memoryless channel with $A_X = \{0, 1\}$, $A_Y = \{0, 1, ?\}$ and transition probabilities $p(?|0) = p(?|1) = p$ and $p(0|0) = p(1|1) = 1 - p$. When a bit is sent over the BEC, it is either erased with probability p or correctly received with probability $1 - p$. The capacity of the BEC is $1 - p$ [15]. More details on the BEC are provided in Chapter 8.

We now consider the situation when the symbols generated by a discrete source U with source alphabet A are to be sent over a discrete channel with input alphabet A_X and output alphabet A_Y. To protect the source symbols against transmission errors, redundancy is added. For this, we use a channel code, which is a block code $\mathcal{C}$ of length n over A_X. We also use a channel encoding scheme, which is an injective function from A^k to $\mathcal{C}$. Using the channel encoding scheme, blocks $(u_1, \ldots, u_k)$ of source symbols of length k are mapped to channel codewords of length n. The channel codewords are then sent over the channel where errors may occur. Next, a function g from A_Y^n to $\mathcal{C}$ called a channel decoding scheme is used to map a received word $\mathbf{y} = (y_1, \ldots, y_n)$ to a channel codeword. Finally, this channel codeword is mapped to the corresponding source word $(\hat{u}_1, \ldots, \hat{u}_k)$ (since the encoding scheme is injective, this source word is unique when it exists). The rate of transmission of this system (or code rate of the code $\mathcal{C}$) is $\frac{k}{n}$. It characterizes the speed with which source information is transmitted over the channel or equivalently the redundancy introduced by the channel code.

An ideal channel decoding scheme for this system minimizes the probability of a word decoding error

$$P_e = \sum_{\mathbf{c} \in \mathcal{C}} \sum_{\mathbf{y} \in A_Y^n : g(\mathbf{y}) \neq \mathbf{c}} Pr\{\mathbf{Y} = \mathbf{y} \mid \mathbf{X} = \mathbf{c}\} Pr\{\mathbf{X} = \mathbf{c}\},$$

where $\mathbf{Y} = (Y_1, \ldots, Y_n)$ and $\mathbf{X} = (X_1, \ldots, X_n)$. This is realized with *maximum a posteriori decoding*, where the received word $\mathbf{y} = (y_1, \ldots, y_n)$ is mapped to a channel codeword $\mathbf{c} = (c_1, \ldots, c_n)$ that maximizes the probability $Pr\{\mathbf{X} = \mathbf{c} \mid \mathbf{Y} = \mathbf{y}\}$. In practice, however, one uses *maximum-likelihood decoding*, where $\mathbf{y}$ is mapped to a channel codeword $\mathbf{c}$ that maximizes the probability $Pr\{\mathbf{Y} = \mathbf{y} \mid \mathbf{X} = \mathbf{c}\}$. It is easy to see that maximum-likelihood decoding is equivalent to maximum a posteriori decoding when all channel codewords are generated with the same probability. Another important decoding scheme is known as *minimum distance decoding* where the received word $\mathbf{y} = (y_1, \ldots, y_n)$ is mapped to a channel codeword $\mathbf{c} = (c_1, \ldots, c_n)$ that has smallest Hamming distance to $\mathbf{y}$. Here the Hamming distance is defined as follows.

Definition 7.4 (Hamming distance). Let $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ be two words of the same length. The Hamming distance $d_H(\mathbf{x}, \mathbf{y})$ between $\mathbf{x}$ and $\mathbf{y}$ is equal to the number of indices $k \in \{1, 2, \ldots, n\}$ such that $x_k \neq y_k$.

Given a BSC with bit error probability p, we have

$$Pr\{\mathbf{Y} = \mathbf{y} \mid \mathbf{X} = \mathbf{c}\} = p^{d_H(\mathbf{c},\mathbf{y})}(1-p)^{n-d_H(\mathbf{c},\mathbf{y})}. \tag{7.1}$$

If $0 < p < 1/2$, the probability in (7.1) is largest when $d_H(\mathbf{c}, \mathbf{y})$ is smallest. Therefore minimum distance decoding and maximum-likelihood decoding are equivalent for this channel.

An alternative to minimizing the probability of a word decoding error is to minimize the information symbol error rate p_e defined as $p_e = \frac{1}{k}\sum_{i=1}^{k} p_e^{(i)}$, where $p_e^{(i)} = Pr\{\hat{u}_i \neq u_i\}$. When the symbols are bits, the information symbol error rate is called the information *bit error rate* (BER). Note that $\frac{1}{k}P_e \leq p_e \leq P_e$. To minimize the information symbol error rate, the decoder uses the *symbol maximum a posteriori* (MAP) rule, where for $i = 1, \ldots, k$, the reconstructed information symbol $\hat{u}_i$ is computed as a symbol $u \in A$ that maximizes the a posteriori probability $Pr\{U_i = u \mid \mathbf{Y} = \mathbf{y}\}$. Here U_i is the random vector that corresponds to the information symbol u_i, $i = 1, \ldots, k$.

The channel coding theorem [45] states that the source information can be transmitted reliably over a noisy channel, provided the rate of transmission is below the capacity of the channel. In other words, any rate below the channel capacity is achievable.

Theorem 7.3 (Shannon's channel coding theorem). *Consider a discrete memoryless channel with input alphabet A_X and capacity C. For any positive number $R < C$ and $\varepsilon > 0$, there exists a channel code $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_M\}$ of length n over A_X and a channel decoding scheme g such that*

1) $M \geq 2^{\lceil Rn \rceil}$.
2) *If codeword $\mathbf{c}_i$ is sent over the channel and word $\mathbf{y}$ is received, then $Pr\{g(\mathbf{y}) \neq \mathbf{c}_i\} < \varepsilon$ for all $i = 1, \ldots, M$.*

7.2.3 Source-Channel Coding

Suppose now that the output of the channel decoding scheme is mapped to a word of length k over a reproduction alphabet $\hat{A}$. The average distortion of the resulting transmission system is $\frac{1}{k}E[d(\mathbf{U}, \mathbf{V})]$, where the random vector $\mathbf{U}$ describes a word of k successive source symbols, the random vector $\mathbf{V}$ describes the corresponding word of k reconstructed symbols, and E denotes the expectation operator.

The source-channel coding theorem [46] says what a system can achieve in terms of average distortion and rate of transmission.

Theorem 7.4 (Shannon's source-channel coding theorem). *Given a discrete memoryless source characterized by rate–distortion function $R(D)$, a discrete*

memoryless channel characterized by capacity $C > 0$, any $D > D_{\min}$, and any $r < C/R(D)$, there exist for sufficiently large k and n an encoding scheme that maps source words of length k into channel words of length n and a decoder that maps channel output words of length n into reproduced words of length k such that the expected distortion is at most D and the transmission rate k/n is at least r.

The encoding scheme promised by the theorem is a concatenation of a lossy compression scheme and a channel encoding scheme. The theorem is also known as the separation theorem because the lossy compression scheme and the channel encoding scheme can be designed independently.

7.2.4 Extensions

Shannon's theorems can be extended to more general information sources. For example, we say that a discrete source X_n is *stationary* if the random process X_n is stationary. The nth marginal entropy of a stationary source is $H_n = H(X_n \mid X_{n-1}, \ldots, X_1)$. One can show that when the source is stationary, the sequence H_n is decreasing and bounded below by zero. This allows us to define the entropy of a stationary source as follows.

Definition 7.5 (Entropy of a stationary source). Let X_n be a stationary source. The entropy of the source (also often called the entropy rate of the source) is defined $\bar{H} = \lim_{n\to\infty} H_n$.

With this definition, Shannon's noiseless coding theorem can be extended to stationary sources that satisfy the asymptotic equipartition property [3]. The source coding theorem can also be extended to sources with abstract alphabets, including the set of real numbers in particular [22].

Shannon's channel coding theorem can be extended to other channels, the most famous one being the *additive white Gaussian noise* (AWGN) channel. In the time-discrete AWGN channel, both the channel input alphabet A_X and the channel output alphabet A_Y are the set of real numbers $\mathbb{R}$. The relationship between the random variable X_n that gives the nth input to the channel and the random variable Y_n that gives the nth output of the channel is given by $Y_n = X_n + Z_n$, where $\{Z_n\}$ is a sequence of independent, identically distributed, Gaussian random variables with mean 0 and variance $N_0/2$. One can show [15] that for this channel reliable transmission is possible as long as the rate of transmission is smaller than the capacity

$$C = \frac{1}{2}\log_2\left(1 + \frac{2P}{N_0}\right) \text{ bits per transmission,}$$

where P is a constraint on the expected value of the random variable X_n^2. If we denote by R the rate of transmission, by $E_s = P$ the symbol energy, and by $E_b = E_s/R$ the energy per bit, then the condition $R < C$ gives $E_b/N_0 > \frac{2^{2R}-1}{2R}$ for reliable transmission. Here E_b/N_0 is called the bit energy to noise spectral density ratio and $\frac{2^{2R}-1}{2R}$ is the Shannon bound. Since $R > 0$, we must also have $E_b/N_0 > \log_e 2$ or $10\log_{10} E_b/N_0 > -1.6$ dB, which is called the theoretical Shannon limit.

In the time-continuous AWGN channel, the relationship between the transmitted signal $s(t)$ (the output of the modulator) and the received signal $r(t)$ (the input of the demodulator) is $r(t) = s(t) + n(t)$, where $n(t)$ is a white Gaussian noise. The capacity of a band-limited AWGN channel is [15]

$$C = W \log_2\left(1 + \frac{P}{N_0 W}\right) \text{ bits per second,}$$

where W is the channel bandwidth in Hz, $N_0/2$ is the power spectral density of the noise, and P is a constraint on the average power. When P is much smaller than $N_0 W$, the channel is called a wideband AWGN channel. One can prove [11] that if binary modulation is used and the demodulated signal is sampled at a rate of $2W$, then E_b/N_0 must be larger than the practical Shannon limit of 0.2 dB to achieve a BER of 10^{-5} for a rate of transmission $R = 1/2$.

7.3 CHANNEL CODING AND ERROR CONTROL FOR BIT ERRORS AND PACKET LOSSES

Channel codes can be divided into two classes: linear and nonlinear. Linear codes are easier to implement and, as a result, have received a greater amount of attention historically. We will also confine our attention to linear codes in this section. We first describe linear block codes, including cyclic redundancy check (CRC) codes for error detection, Reed–Solomon codes, low-density parity-check (LDPC) codes, irregular repeat-accumulate (IRA) codes, tornado codes, digital fountain codes, and lattice codes. We then describe convolutional codes, rate-compatible punctured convolutional (RCPC) codes, and turbo codes. We discuss the properties of these codes and mention efficient algorithms for encoding and decoding, emphasizing their computational complexity. We also explain how the problem of burst errors (explained further in Chapter 8) can be alleviated with interleaving.

7.3.1 Linear Block Codes

In a linear block code, the codeword symbols are taken from a field. A formal definition of a field is beyond the scope of this text and may be found in a math-

ematics book on abstract algebra (see [12]). Informally, a field consists of a set of elements, together with two operations called addition and multiplication that must fulfill a given number of properties. Some examples of well-known fields are the set of real numbers and the set of rational numbers. These fields are known as infinite fields because they contain an infinite number of elements. Linear block codes, however, typically consist of elements from a finite field. In particular, consider the finite field $GF(2) = \{0, 1\}$. The addition operation for $GF(2)$ is modulo-2 addition and the multiplication operation is defined similarly to the multiplication operation for two binary numbers:

$$0+0=0 \quad 0+1=1 \quad 1+1=0$$
$$0*0=0 \quad 0*1=0 \quad 1*1=1.$$

Finite fields are also called Galois fields. The size of a Galois field must be a power of a prime. Conversely, for any prime power q, one can construct a Galois field of size q. Let $GF(q)$ be a finite field of size q and let n be a positive integer. Then it is easy to check that $[GF(q)]^n$ is a linear space over $GF(q)$. An (n, k) linear block code $\mathcal{C}$ over $GF(q)$ is a k-dimensional linear subspace of the linear space $[GF(q)]^n$. In particular, for any two codewords $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}$, the sum of the codewords is also a codeword, $\mathbf{c}_1 + \mathbf{c}_2 \in \mathcal{C}$. Since $\mathcal{C}$ is a k-dimensional linear space, we can find a set of k basis vectors so that every codeword can be expressed as a linear combination of the basis vectors. In vector-matrix notation, we can express every codeword $\mathbf{c}$ of $\mathcal{C}$ as

$$\mathbf{c} = \mathbf{u}\mathbf{G}, \tag{7.2}$$

where $\mathbf{u}$ is a $1 \times k$ vector of field elements and $\mathbf{G}$ is a $k \times n$ matrix whose k rows are k basis vectors. The matrix $\mathbf{G}$ is known as a generator matrix and elementary row operations can be performed on $\mathbf{G}$ to form another matrix $\mathbf{G}'$ that will generate an equivalent code. If $\mathbf{G}$ is manipulated to be of the form $\mathbf{G} = [\mathbf{I}_k|\mathbf{P}]$ where $\mathbf{I}_k$ is the $k \times k$ identity matrix and $\mathbf{P}$ is a $k \times (n-k)$ matrix, then $\mathbf{G}$ is said to be in systematic form and the first k symbols of the codeword $\mathbf{c}$ will be identical to the k symbols of $\mathbf{u}$. The final $n-k$ symbols of $\mathbf{c}$ are referred to as parity symbols.

The performance of a block code is often measured by the number of errors that it can correct or the amount of noise that it can remove. The performance is usually dependent on two things: (1) the decoder that is used to decode a received word to a codeword and (2) the distance between each pair of codewords in the block code. Let us first consider the distance between a pair of codewords. For block codes, the Hamming metric or the Euclidean metric is usually used to measure the distance between pairs of codewords. The Hamming distance is useful for measuring the distance between two codewords whose symbols belong to a

finite field. Sometimes in communications applications, however, each field element of the codeword is mapped to a real number. In such scenarios, it is useful to use a Euclidean metric to determine the distance between codewords. For example, if each codeword, $\mathbf{c}_i = (c_{i,1}, c_{i,2}, \ldots, c_{i,n})$, is mapped to a vector of real numbers, $\mathbf{r}_i = (r_{i,1}, r_{i,2}, \ldots, r_{i,n})$, then the Euclidean distance between two codewords $\mathbf{c}_i, \mathbf{c}_j$ may be defined as

$$d_E(\mathbf{c}_i, \mathbf{c}_j) = \sqrt{(r_{i,1} - r_{j,1})^2 + (r_{i,2} - r_{j,2})^2 + \cdots + (r_{i,n} - r_{j,n})^2}. \tag{7.3}$$

Now, if we let $d_{\min}$ represent the minimum distance between any pair of codewords, and if an arbitrary codeword is transmitted over a noisy channel, then the codeword may be successfully recovered if the decoder decodes the received word to the closest codeword and the amount of noise is less than $d_{\min}/2$. Note that if the block code is linear, then the minimum distance $d_{\min}$ is simply the smallest weight of a nonzero codeword. Here the weight of a codeword is the number of its nonzero symbols.

For the Hamming distance, successful decoding translates into there being less than $d_{\min}/2$ changes to the symbols of the original codeword, where $d_{\min}$ represents the minimum Hamming distance between any pair of codewords. For the Euclidean distance, successful decoding translates into the magnitude of the noise being less than $d_{\min}/2$. For the aforementioned, we may visualize the correct decoding region of each codeword to be a sphere with radius $d_{\min}/2$ (as in Figure 7.2), and thus if a codeword is corrupted by noise, as long as the noise does not perturb the codeword to be outside of its correct decoding region, then suc-

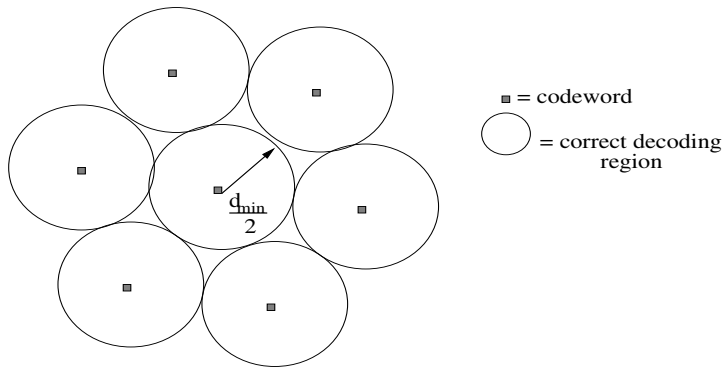


FIGURE 7.2: Example of a codebook that consists of several codewords with a minimum distance of $d_{\min}$. The correct decoding regions are shown as spheres centered around the codewords with a radius of $d_{\min}/2$.

cessful decoding will be guaranteed. It is apparent that for a given n and k, it is desirable to find a code that maximizes $d_{\min}$.

Once a code with parameters n, k, and $d_{\min}$ is found, efficient encoding and decoding algorithms for generating the code are necessary to enable the code to be practical.

In general, if the minimum distance of a linear code is t, then the receiver can detect up to $t-1$ transmission errors. However, by using minimum distance decoding, where the received word is decoded to a nearest codeword, a linear code of minimum distance t allows the correction of up to $\lfloor \frac{t-1}{2} \rfloor$ errors. It can be shown that a linear code of minimum distance t can simultaneously correct e_c errors and detect e_d errors if $e_c + e_d \leq t-1$ with $e_c \leq e_d$. Moreover it can correct e_e erasures and e_c errors simultaneously if $e_e + 2e_c \leq t-1$.

The minimum distance of an (n,k) linear code must be less than or equal to $n-k+1$. Linear (n,k) codes whose minimum distance is equal to $n-k+1$ are called maximum-distance separable (MDS) codes.

Linear codes can be simply modified to obtain new linear codes. Puncturing a linear code consists of removing a number of coordinate positions from each codeword. If an (n,k) linear MDS code is punctured, then the resulting code is an $(n-1,k)$ linear MDS code. Shortening a linear code consists of keeping only codewords with the same symbol in a given position and then deleting this position. If an (n,k) linear MDS code is shortened by keeping only codewords with the zero symbol in a given position, then the resulting code is an $(n-1,k-1)$ linear MDS code.

The encoding scheme for an (n,k) linear code can be implemented in $O(n^2)$ time. However, there is no efficient way to decode a general linear code with maximum-likelihood decoding [8]. Usually one uses syndrome decoding. To explain syndrome decoding, we must introduce the parity check matrix. The parity check matrix $\mathbf{H}$ of an (n,k) linear code with generator matrix $\mathbf{G}$ is an $(n-k) \times n$ matrix whose rows are orthogonal to the rows of the generator matrix, that is,

$$\mathbf{G}\mathbf{H}^T = \mathbf{0}. \tag{7.4}$$

The parity check matrix may be viewed as a generator matrix for a code that lies in the null space of $\mathbf{G}$. It is clear that for any codeword $\mathbf{c}$ that is generated by $\mathbf{G}$,

$$\mathbf{c}\mathbf{H}^T = \mathbf{0}. \tag{7.5}$$

Now, if we add an error vector, $\mathbf{e}$ to $\mathbf{c}$, then

$$(\mathbf{c}+\mathbf{e})\mathbf{H}^T = \mathbf{0} + \mathbf{e}\mathbf{H}^T = \mathbf{s}, \tag{7.6}$$

where we call $\mathbf{s}$ the syndrome of $(\mathbf{c}+\mathbf{e})$. If we let each syndrome correspond to an error vector, then the function of a syndrome decoder is to first compute the

Table 7.1: Standard array of a binary code. The first row contains all codewords of the code. Each following row is formed by taking a minimum weight vector, adding it to the first row, and then checking if the resulting addition is already part of the standard array. If the resulting addition is not part of the standard array, then it is added as a new row to the standard array. This process is continued until the standard array is filled.

$\mathbf{c}_1 = \mathbf{0}$	$\mathbf{c}_2$	$\dots$	$\mathbf{c}_{2^k}$
$\mathbf{e}_1$	$\mathbf{e}_1 + \mathbf{c}_2 \dots$	$\dots$	$\mathbf{e}_1 + \mathbf{c}_{2^k}$
$\mathbf{e}_2$	$\mathbf{e}_2 + \mathbf{c}_2 \dots$	$\dots$	$\mathbf{e}_2 + \mathbf{c}_{2^k}$
$\mathbf{e}_3$	$\mathbf{e}_3 + \mathbf{c}_2 \dots$	$\dots$	$\mathbf{e}_3 + \mathbf{c}_{2^k}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathbf{e}_{2^{n-k}-1}$	$\mathbf{e}_{2^{n-k}-1} + \mathbf{c}_2 \dots$	$\dots$	$\mathbf{e}_{2^{n-k}-1} + \mathbf{c}_{2^k}$

syndrome of the received vector and then subtract the corresponding error vector from the received vector. Another way of viewing syndrome decoding is through a standard array [36]. A standard array of a binary code is formed by setting aside a $2^{n-k} \times 2^k$ array and populating the first row of the array with all 2^k possible codewords with the all-zero codeword occupying the first column of the first row. Next, we generate all possible weight 1 error vectors and add each error vector to the first row to generate another row. This process is continued by increasing the weight of the error vector and filling the rows until the entire array is populated as in Table 7.1. If the result of an addition of an error vector with the first row equals a row that is already in the standard array, then the error vector is skipped and the next error vector is used to generate further rows. The result will be an array, where each row corresponds a shift of all of the codewords by an error vector. The first column will contain the error vectors and each row may be indexed by the syndrome. Therefore, syndrome decoding may be viewed as indexing a row of the standard array and then adding the first element of the row to the received vector.

Example 5. Consider a (3, 1) binary repetition code. This block code consists of two codewords, {000, 111}. A generator matrix for the code is

$$\mathbf{G} = [1\ 1\ 1]. \tag{7.7}$$

The corresponding parity check matrix is

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}. \tag{7.8}$$

And we can tabulate the standard array as follows:

$$\left\{\begin{array}{ll} 000 & 111 \\ 001 & 110 \\ 010 & 101 \\ 100 & 011 \end{array}\right\}. \tag{7.9}$$

Notice that the minimum weight codeword is (1, 1, 1) and therefore the minimum distance of the code is 3. This implies that one error may be *corrected* if a minimum distance decoder is used for decoding. Alternatively, up to two errors may be *detected*.

Since the decoding of general linear block codes is not efficient, special classes of linear codes with fast decoding algorithms were developed. The most popular of these is the class of cyclic codes. An (n, k) linear code is a cyclic code if for each codeword $(c_1, \ldots, c_{n-1}, c_n)$, the right shift $(c_n, c_1, \ldots, c_{n-1})$ is also a codeword.

7.3.1.1 CRC Codes

CRC codes are shortened cyclic binary codes used for error detection. Given a generator polynomial $g(x) = \sum_{i=0}^{r} g_i x^i,\ g_i \in \{0, 1\}$ of degree r, the codeword for a binary information sequence $\mathbf{u} = (u_1, \ldots, u_k)$ is the concatenation $\mathbf{u} * \mathbf{p}$ of $\mathbf{u}$ and the word $\mathbf{p}$ of length r associated to the polynomial $p(x) = x^r u(x) \bmod g(x)$. Here we use the unique correspondence between a word $\mathbf{w} = (u_1, \ldots, u_m)$ of length m and the polynomial $w(x) = w_1 + w_2 x + \cdots + w_m x^{m-1}$ of degree at most $m - 1$. Suppose that the codeword $\mathbf{u} * \mathbf{p}$ is sent over a binary symmetric channel and let $\mathbf{u}' * \mathbf{p}'$ be the received word. Here $\mathbf{u}'$ and $\mathbf{p}'$ are words having the same length as $\mathbf{u}$ and $\mathbf{p}$, respectively. Then the decoder computes $p''(x) = x^r u'(x) \bmod g(x)$ and declares an error if $p''(x)$ is not equal to $p'(x)$. Some of the most popular generator polynomials are the CRC-12 polynomial $1 + x + x^2 + x^3 + x^{11} + x^{12}$, the CRC-16 polynomial $1 + x^2 + x^{15} + x^{16}$, and the CRC-CCITT polynomial $1 + x^5 + x^{12} + x^{16}$. A CRC code with generator polynomial $g(x) = \sum_{i=0}^{r} g_i x^i,\ g_0 \neq 0, g_r \neq 0$ can detect any burst error of length $k \leq r$. Agarwal and Ivanov [2] provided an $O(nm2^{r+m})$ algorithm for computing the probability of undetected error for a CRC code of length n whose generator polynomial has degree r and m nonzero coefficients. The encoding and decoding of CRC codes can be efficiently implemented with shift register circuits.

7.3.1.2 Reed–Solomon Codes

Reed–Solomon codes are nonbinary linear block codes over a finite field $GF(q)$. Let α be an element of order n in $GF(q)$ [i.e., n is the smallest positive inte-

ger such that $\alpha^n = 1$, where 1 is the identity element for the multiplication in $GF(q)$]. Let $r \in \{1, \ldots, n\}$. The set of all vectors $(c_0, \ldots, c_{n-1})$ in $[GF(q)]^n$ such that $\sum_{i=0}^{n-1} c_i \alpha^{ij} = 0, \ \ j = 1, \ldots, r$, is called a Reed–Solomon code of redundancy r over $GF(q)$. This code is an $(n, n-r)$ cyclic code of minimum distance $r+1$. Thus, Reed–Solomon codes are MDS codes. Therefore an (n, k) Reed–Solomon code can correct e_0 symbol erasures and e_1 symbol errors simultaneously if $e_0 + 2e_1 \leq n - k$. In particular, in a channel where only erasures can occur, all codeword symbols of an (n, k) Reed–Solomon codeword can be correctly recovered if at least k symbols are received.

Reed–Solomon codes are suitable for the correction of burst errors. An (n, k) Reed–Solomon code can be decoded in $O(n^2)$ time with Berlekamp's iterative algorithm [7]. Guruswami and Sudan [23] developed a polynomial-time algorithm for Reed–Solomon codes that finds a list of all codewords within a distance $\lceil n - \sqrt{n(k-1)} - 1 \rceil$ from a received word. Thus, the algorithm is guaranteed to determine the list of all potentially sent codewords if at most $\lceil n - \sqrt{n(k-1)} - 1 \rceil$ errors occurred during transmission. The complexity of the algorithm is $O(n^{15})$ if exactly $\lceil n - \sqrt{n(k-1)} - 1 \rceil$ errors occurred and only $O(n^3)$ otherwise. The algorithm of Berlekamp [7] is a hard-decision decoding algorithm, which does not exploit all available information at the receiver when the demodulator allows soft decisions. Efficient soft-decision decoding algorithms for Reed–Solomon codes were proposed by Koetter and Vardy [28] and Jiang and Narayanan [26]. For example, the algorithm of Jiang and Narayanan [26] outperforms hard-decision decoding by up to 3.1 dB at decoding error probability 10^{-5} when decoding a (15, 7) Reed–Solomon code over a binary-input AWGN channel.

7.3.1.3 LDPC Codes

LDPC codes were introduced by Gallager [21]. They have attracted increased interest since MacKay and Neal [34,35] reported their outstanding performance on a binary-input AWGN channel. An (n, k) LDPC code is a linear code with a sparse parity-check matrix $H = (h_{ij})$. It can also be described with a bipartite graph, called Tanner graph, whose set of nodes consists of variable nodes and check nodes. Variable nodes correspond to the n codeword symbols, while check nodes correspond to the $(n-k)$ equations defined by the parity-check constraint. A variable node is connected to a check node if the codeword symbol corresponding to the variable node is involved in the parity equation defining the check node. That is, check node i is connected to variable node j if $h_{ij} = 1$. In a regular LDPC code, each column has the same number d_v of ones and each row has the same number d_c of ones. Thus, in the Tanner graph of the code each variable node has degree d_v and each check code has degree d_c, as shown in Figure 7.3. In an irregular LDPC code, the degrees of the variable nodes and check nodes are chosen according to some nonuniform distribution. Efficient encoding of LDPC codes

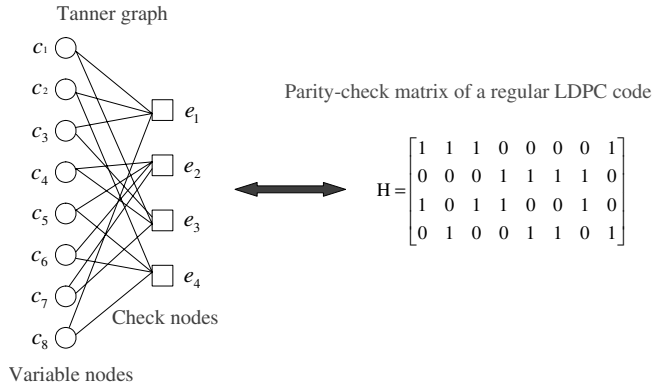


FIGURE 7.3: Tanner graph of a regular LDPC code of length 8. The degree of the variable nodes is $d_v = 2$, and the degree of the check nodes is $d_c = 4$.

is discussed in [40], where it is shown in particular that some of the best LDPC codes can be encoded in $O(n)$ time with high probability. Because of the sparseness of their Tanner graph, LDPC codes can be decoded in $O(n)$ time with an iterative procedure known as probabilistic decoding [21], message passing, sum-product algorithm [49], or belief propagation [37]. These algorithms alternately pass information between adjacent variable nodes and check nodes to compute estimates of the a posteriori probabilities of the codeword symbols. The decoded codeword is based on the estimates obtained after convergence or if a maximum number of iterations is reached. Chung *et al.* [14] were able to design a rate-$\frac{1}{2}$ irregular LDPC code of length 10^7 bits that is only 0.04 dB away from the Shannon limit for a binary-input AWGN channel and a bit error rate of 10^{-6}.

7.3.1.4 IRA Codes

Irregular repeat-accumulate (IRA) codes were introduced by Jin and colleagues [27] as a generalization of the repeat-accumulate (RA) codes of [16]. IRA codes can be encoded in linear time. They are decoded with the sum-product algorithm, achieving on the binary-input AWGN channel a performance competitive with that of the best LDPC codes of comparable complexity.

7.3.1.5 Tornado Codes

Tornado codes [31,32] are (n, k) erasure codes that allow encoding and decoding with time complexity linear in the block length n. This speed-up over Reed–Solomon codes is obtained at the cost that slightly more than k encoding symbols

are required to reconstruct all k information symbols. More precisely, Luby *et al.* [32] prove that for any $\varepsilon > 0$, one can construct a Tornado code that recovers all k information symbols from only $(1+\varepsilon)k$ encoding symbols with probability $1 - O(n^{-3/4})$.

7.3.1.6 Digital Fountain Codes

Luby [30] recently introduced a new class of powerful erasure correcting codes called Luby Transform (LT) codes. LT codes are rateless in the sense that a potentially limitless stream of encoding symbols (or a digital fountain) can be generated for a given information sequence. Thus, in contrast to classical block codes, one need not design the code a priori for a fixed n. With LT codes, each encoding symbol can be generated from k information symbols in $O(\log k)$ time on average, and one can recover all k information symbols from $k + O(\sqrt{k}\log^2(k/\delta))$ encoding symbols with probability $1 - \delta$ in $O(k \log k)$ time, on average.

By concatenating an LDPC code as an outer code and an LT code as an inner code, Shokrollahi [47] was able to construct rateless codes called Raptor codes whose erasure correcting performance is similar to that of LT codes, but can be encoded and decoded in only $O(k)$ time.

7.3.1.7 Lattice Codes

Codes over finite fields can also be interpreted as codes over real numbers by mapping each element of the finite field to a real number. For example, in the binary $(3, 1)$ repetition code, the binary digit 0 can be mapped to the real value $-a$ and the binary digit 1 can be mapped to the real value $+a$ so that the two codewords are $(-a, -a, -a)$ and $(+a, +a, +a)$. Minimum distance decoding then means decoding the received vector to the codeword that is closest in Euclidean distance. In this section, we will consider a class of codes called lattice codes, which contain codewords that are amenable to Euclidean distance decoding instead of Hamming distance decoding.

Informally, a lattice Λ is an infinite regular array of points that covers an m-dimensional space uniformly. A lattice is defined by a set of basis vectors so that any point in the lattice can be represented as a linear combination of the basis vectors. More precisely, if the basis vectors are given as $\mathbf{v}_1 = (v_{1,1}, v_{1,2}, \ldots, v_{1,m})$, $\mathbf{v}_2 = (v_{2,1}, v_{2,2}, \ldots, v_{2,m}), \ldots, \mathbf{v}_n = (v_{n,1}, v_{n,2}, \ldots, v_{n,m})$ where $m \geq n$, then we can define a generator matrix, $\mathbf{G}$, to be a matrix that contains the basis vectors as the rows of the matrix and any lattice point can be written as

$$\lambda = \zeta \mathbf{G},$$

where ζ is an n-dimensional vector of integers [13]. For example, a generator matrix for the m-dimensional integer lattice (often written as $\mathbb{Z}^m$) is the m-

dimensional identity matrix. The lattice points of the m-dimensional integer lattice consist of all the possible m-dimensional vectors of integers.

The conventional method of using a lattice for channel coding is to take a finite subset of lattice points and define a one-to-one mapping between the lattice points and binary vectors that represent the information that is to be sent over a channel. The goal of using lattice codes for channel coding is to maximize the amount of information that can be conveyed over the channel for a given power constraint. As an example, consider the problem of sending bits over an AWGN channel. One method of addressing this problem is to choose a lattice and then map vectors of bits to a finite subset of the lattice. The lattice points will then represent the real values that are sent over the channel and corrupted by noise. The decoder will receive a noisy sequence of points and attempt to recover the bits by decoding the noisy values to the closest lattice points in Euclidean distance. The decoding region for each lattice point is often referred to as its *Voronoi region* and is defined to be the set of points whose Euclidean distance to the given lattice point is closer than that to any other lattice point.

To illustrate the above concepts, consider the hexagonal lattice defined by the generator matrix

$$\mathbf{G} = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix}.$$

A pictorial representation of the lattice points that are generated by the aforementioned generator matrix is given in Figure 7.4, where a finite subset of the

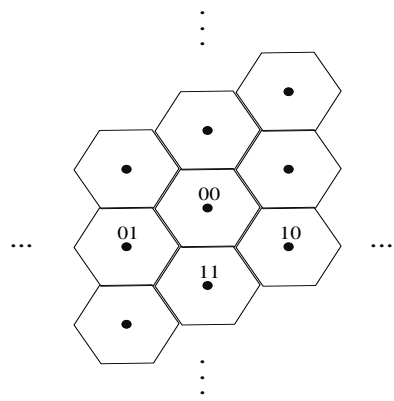


FIGURE 7.4: Example of a hexagonal lattice. A finite subset of the lattice points is shown, and the Voronoi region of each lattice point is a hexagon.

hexagonal lattice is shown. If we take four of the lattice points as our finite subset, then we can define a mapping of this subset to binary vectors of length two. In Figure 7.4, we chose four lattice points and arbitrarily assigned two-dimensional bit vectors to the lattice points. A transmitter may then parse a bit string into vectors of length two and map each vector to a lattice point. Each of the lattice points represent a two-dimensional real vector that will be corrupted by additive white Gaussian noise. The Voronoi region of each lattice point is shown as a hexagon. Therefore, if the additive noise is not large enough to perturb a lattice point outside of its Voronoi region, then the decoder will be able to successfully decode the bits sent by the encoder.

7.3.2 Convolutional Codes

A class of codes that are often used with both Hamming distance decoders and Euclidean distance decoders are convolutional codes. For simplicity, we restrict our description to binary convolutional codes. Like an (n, k) linear block code, an (n, k) convolutional code maps length-k blocks of information symbols into length-n blocks of output symbols, but each output block depends on the current and previous information blocks. A convolutional code can in general be defined by a linear finite state machine (LFSM). For a binary (n, k) convolutional code of memory ν, the LFSM can be expressed as ν stages of k shift registers that are connected by n different modulo-2 adders, as in Figure 7.5. At each time instant, k bits are shifted in to the LFSM and n bits are output from the LFSM. The shift registers in combination with the modulo-2 adders serve to constrain the possible output sequences. The goal of designing a convolutional code is to constrain the possible output sequences to be separated by a large distance. For example, if

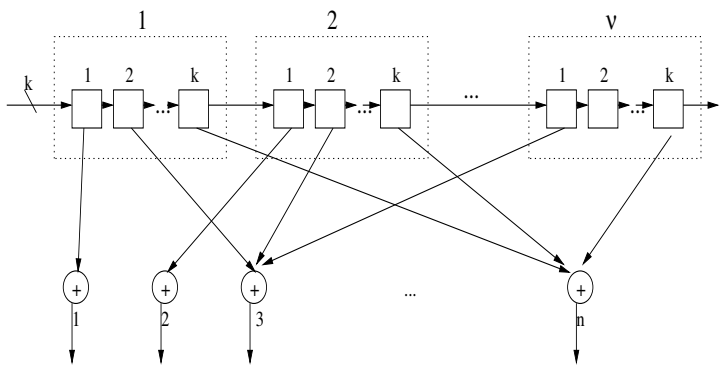
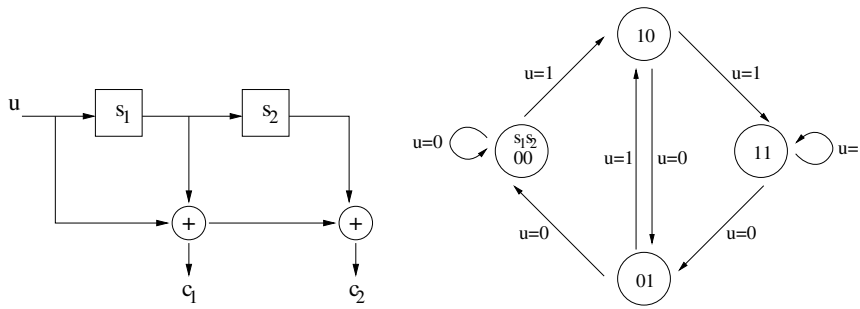


FIGURE 7.5: A linear finite state machine representation of a convolutional code. There are k input bits, n output bits, and the memory is ν.

the convolutional code is to be used with a Hamming distance decoder, then it is desirable to design the convolutional code so that the possible output bit sequences are separated by a large Hamming distance. However, if the convolutional code is to be used with a Euclidean distance decoder, then a mapping between the output bit sequences and vectors of real values must be defined and it is desirable to design the convolutional code so that the possible vectors of real values are separated by a large Euclidean distance. In general, better convolutional codes can be found as the memory is increased.

As is common with LFSMs, it is often beneficial to express the LFSM as a state transition diagram. The states represent the contents of the registers in the LFSM, and the transitions between states are determined by the input bits. As an example, consider the convolutional code shown in Figure 7.6a. The parameters of the code are given as $\nu = 2$, $k = 1$, and $n = 2$. In the example, bits are shifted into the registers one at a time and the input bit is represented as the variable u. The contents of the registers that represent the state of the convolutional code are given as variables s_1 and s_2. The output bits of the convolutional code are given as variables c_1 and c_2. We can represent the convolutional code as a state diagram by assigning a circle to each possible state (as in Figure 7.6b) and representing the transitions between states with arrows. As can be seen from Figure 7.6 the input bit, in combination with the current state of the convolutional code, will determine the following state of the convolutional code. For example, if at a given time instant the state of the registers is given as 01 and the input bit is 1, then 1 will be shifted out of the right-most register and the input bit will be shifted into the left-most register. As a result, the following state of the convolutional code will be 10.



(a) (b)

FIGURE 7.6: (a) Convolutional code example with parameters $\nu = 2$, $k = 1$, and $n = 2$. (b) State diagram representation of convolutional code.

Any convolutional code can also be expressed as a vector-matrix product

$$\vec{\mathbf{c}}(D) = \vec{\mathbf{u}}(D)\mathbf{G}(D), \tag{7.10}$$

where $\vec{\mathbf{c}}(D) = [c_1(D), c_2(D), \ldots, c_i(D), \ldots, c_n(D)]$ is a row vector of n polynomials, with the ith polynomial representing the ith output bit sequence. A bit sequence, $\{b_0, b_1, \ldots, b_m\}$ can be represented as a polynomial $b_0 + b_1 D + \cdots + b_m D^m$ by weighting the ith bit in time by D^i where D is a variable representing delay. Similarly, $\vec{\mathbf{u}}(D) = [u_1(D), u_2(D), \ldots, u_j(D), \ldots, u_k(D)]$ is a row vector of k polynomials, with the jth polynomial representing the jth bit sequence. The matrix $\mathbf{G}(D)$ is a $k \times n$ matrix that contains generator polynomials that specify the relationship between the input polynomials and the output polynomials. For example, consider the convolutional code shown in Figure 7.6a. The first output bit is the modulo-2 addition of the current input bit and the previous input bit in time. We can write a polynomial equation for the first output bit as

$$c_1(D) = u(D) + Du(D),$$

where the variable D represents delay. The second output bit is equal to the modulo-2 addition of the current input bit, the bit from the previous time instant, and the bit from two time instants ago. We can write a polynomial equation for the second output bit as

$$c_2(D) = u(D) + Du(D) + D^2u(D). \tag{7.11}$$

Now, the two equations just given can be combined into the form of (7.10) where the generator matrix can be expressed as

$$\mathbf{G}(D) = \begin{bmatrix} 1 + D & 1 + D + D^2 \end{bmatrix}$$

and $\vec{\mathbf{c}}(D) = [c_1(D),\ c_2(D)]$, $\vec{\mathbf{u}}(D) = [u(D)]$. The aforementioned representation of a convolutional code is often useful for analyzing the performance characteristics of a code.

Another representation of a convolutional code is as a trellis. A pictorial representation of a trellis can be formed by aligning all of the possible states in a vertical column for each time instant and then connecting the states in accordance with the state transition diagram. The trellis representation of a convolutional code is particularly useful for decoding, as quick decoding algorithms such as the Viterbi decoding algorithm can be derived from the trellis representation. An example of a trellis representation of the convolutional code in Figure 7.6a is given in Figure 7.7, where the states are represented as dots and the transitions are labeled by the input bit that causes the transition and the resulting output bits.

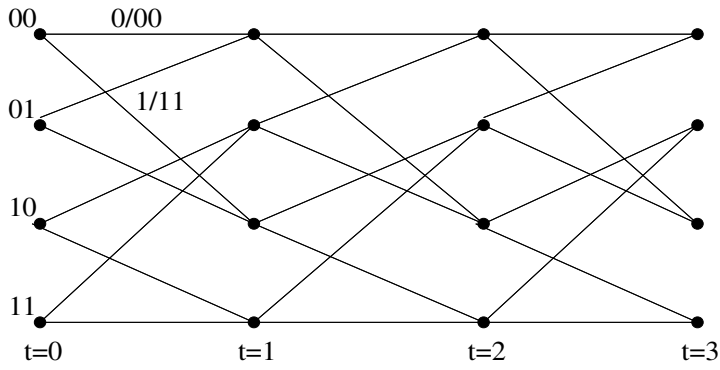


FIGURE 7.7: Trellis representation of convolutional code given in Figure 7.6a.

The goal of the decoder is to find the codeword from the convolutional code closest to the received sequence in either Hamming distance or Euclidean distance (if the output bits of the convolutional code are mapped to real values). This can be done efficiently by using the trellis diagram. If we let $y^{(i)}$ represent the received block (of length n) at time instant i and let $c^{(i)}_{u_i, s_i \to s_{i+1}}$ represent the output block (of length n) at time instant i corresponding to the transition between states s_i and s_{i+1} that results from input block u_i (of length k), then mathematically the goal of the decoder is to find the output sequence $\hat{\mathbf{c}}$ that is closest in distance to the received sequence,

$$\hat{\mathbf{c}} = \arg\min_{\mathbf{c} \in \mathcal{C}} \mathbf{d}(\mathbf{c}, \mathbf{y}).$$

In the equation just given, $\mathcal{C}$ represents the set of valid codewords, $\mathbf{c} = \{c^{(0)}_{u_0, s_0 \to s_1}, c^{(1)}_{u_1, s_1 \to s_2}, \ldots, c^{(m)}_{u_m, s_m \to s_{m+1}}\}$ represents a valid sequence of output blocks, and $\mathbf{y} = \{y^{(0)}, y^{(2)}, \ldots, y^{(m)}\}$ represents a sequence of received blocks. Furthermore, $\mathbf{d}(\mathbf{c}, \mathbf{y})$ is the distance metric between the output sequence and the received sequence and can be written as a summation of distances between the received blocks and the output blocks at the various time instants,

$$\mathbf{d}(\mathbf{c}, \mathbf{y}) = \sum_{i=0}^{m} d\left(c^{(i)}_{u_i, s_i \to s_{i+1}}, y^{(i)}\right). \tag{7.12}$$

Assuming that the trellis starts in state zero (i.e., all of the registers of the convolutional code are cleared to zero), a naive approach to finding the codeword sequence that is closest to the received sequence would be to calculate the distance of the received sequence to each path in the trellis that starts in state zero and then

declare the path that is closest to the received sequence as the decoded codeword. This method is inefficient because the amount of computation grows exponentially with the length of the sequence. A more efficient decoding algorithm can be realized by using the Viterbi algorithm. The Viterbi algorithm begins at the first stage of the trellis by calculating the distance between all branches of the trellis that emerge from state zero and the corresponding block in the received sequence (i.e., $d(c^{(0)}_{u_0,s_0\to s_1}, y^{(0)})$). In general, there will be 2^k branches that emerge from any state, so the Viterbi algorithm starts by calculating 2^k distance metrics. For the next stage in the trellis, we can prune paths that end in the same state s_1. More specifically, for all paths that converge to the same state, we can keep the path that has the minimum distance up to that state and prune all other paths. This works because if multiple paths converge to the same state, then any path that may emerge from this state will have an associated distance that will be added to the distance associated with the path that ends in that state. Mathematically, we can break up the total distance metric for any path that goes through a state at time i as the distance metric from state 0 at time 0 to state s_i at time i and the distance metric from state s_i to state s_{m+1} at the end of the trellis:

$$\mathbf{d}(\mathbf{c},\mathbf{y}) = \sum_{j=0}^{i-2} d\left(c^{(j)}_{u_j,s_j\to s_{j+1}}, y^{(j)}\right) + d\left(c_{u_{i-1},s_{i-1}\to s_i}, y^{(i-1)}\right) + \sum_{j=i}^{m} d\left(c^{(j)}_{u_j,s_j\to s_{j+1}}, y^{(j)}\right). \tag{7.13}$$

From (7.13), we see that all paths that merge at state s_i will have the same possible distances $\sum_{j=i}^{m} d(c^{(j)}_{u_j,s_j\to s_{j+1}}, y^{(j)})$ added to the existing distance of the path and therefore a path with a larger distance at state s_i cannot achieve a smaller overall distance than a path with a smaller distance at state s_i. As a result, we can prune the total number of paths to be no larger than the total number of states. In other words, at each time instant, at most $2^{\nu k}$ paths are kept (one path for each state). At time instant m, the minimum-distance path can be determined and traced back to state 0 at time instant 0. The output sequence associated with the minimum distance path is the decoded codeword.

Because the time complexity of the Viterbi algorithm is exponential in the memory order, faster but suboptimal sequential decoding algorithms (e.g., the Fano and Stack algorithms [50]) are used in many time-critical applications. A generalization of the standard Viterbi algorithm is the list Viterbi algorithm (LVA) [44,41], which finds the L most likely paths instead of only the most likely one.

Symbol MAP decoding (see Section 7.2.2) of convolutional codes can be done with soft-input soft-output algorithms. Two of the most prominent ones are the

BCJR algorithm of Bahl and colleagues [4] and the soft-output Viterbi algorithm of Hagenauer and Hoeher [25]. Both algorithms output for each information bit $u_i,\ i = 0, \ldots, km$, an a posteriori log-likelihood ratio (LLR)

$$\Lambda(u_i) = \log \frac{Pr\{U_i = 1 \mid \mathbf{Y} = \mathbf{y}\}}{Pr\{U_i = 0 \mid \mathbf{Y} = \mathbf{y}\}}$$

whose sign specifies the reconstructed source bit $\hat{u}_i$.

A family of convolutional codes can be generated from a single convolutional code, called a mother code, with rate $\frac{1}{n}$. Some output symbols of the mother encoder are punctured, which allows the construction of a family of codes with rates $\frac{p}{np}, \frac{p}{np-1}, \ldots, \frac{p}{p+1}$, where p is the puncturing period. To obtain RCPC codes [24], all protection symbols of the higher rate-punctured code are used by the lower rate codes (the higher rate codes are embedded into the lower rate codes). A nice feature of RCPC codes is that if a higher rate code does not provide enough protection, one can switch to a lower rate code simply by adding extra redundant symbols. Another good feature of RCPC codes is that the same Viterbi trellis can be used for all rates.

As mentioned earlier, a convolutional code may be used as either a Hamming distance code or a Euclidean distance code. If the convolutional code is used as a Euclidean distance code, then a mapping between the possible output bits at any given time instant and a set of real values must be defined. One method of defining a mapping is to first choose a constellation of real values such as a finite subset of lattice points and then define a bijective mapping between the lattice points and the possible output bit vectors. For example, the convolutional code shown in Figure 7.6a has two output bits, which can assume one of four possible two-bit combinations, so we can define a mapping between the four possible two-bit combinations and the four lattice points shown in Figure 7.4. Recall, however, that the goal of code design is to maximize the minimum distance between possible output sequences, and the aforementioned procedure may not maximize the minimum distance for a given convolutional code and a given set of constellation points. A proper Euclidean distance code design should jointly consider the convolutional code and the set of constellation points in defining the mapping between bits and constellation points. This concept was first introduced by Ungerboeck [48], and the resulting codes are often referred to as trellis-coded modulation (TCM) codes.

TCM codes are usually formed by letting a convolutional code index a partition of constellation points [48]. Forney [20] and Conway and Sloane [13] independently utilized this heuristic to define a set of codes that are derived from a convolutional code that indexes a lattice partition. More specifically, both Forney and Conway and Sloane showed that good trellis codes can be obtained by partitioning a well-known lattice and then searching for a convolutional code to index

the partition. In this chapter, we will denote a lattice partition as Λ/Λ', where Λ' is a sublattice of Λ and partitions Λ into *cosets* of Λ'. A coset of Λ' is formed by choosing a lattice point, $\lambda \in \Lambda$, and adding this element to all of the lattice points in Λ'. We denote the coset as $\Lambda' + \lambda$. For example, consider the lattice partition $\mathbb{Z}/4\mathbb{Z}$, where $\mathbb{Z}$ is the integer lattice formed by the 1×1 identity matrix and $4\mathbb{Z}$ is a sublattice of $\mathbb{Z}$ that is formed by scaling the integer lattice by 4. Four disjoint cosets may be formed from $4\mathbb{Z}$ by adding the lattice points $\{0, 1, 2, 3\}$ to $4\mathbb{Z}$. Notice that the union of the four cosets is equal to $\mathbb{Z}$. One method of arriving at the four cosets of $4\mathbb{Z}$ is to use a partition tree, $\mathbb{Z}/2\mathbb{Z}/4\mathbb{Z}$. The first level of the tree is a partition of $\mathbb{Z}$ into two cosets that consist of the even and odd lattice points of $\mathbb{Z}$ (i.e., $2\mathbb{Z}$ and $2\mathbb{Z}+1$). The next level of the partition tree further partitions $2\mathbb{Z}$ into $4\mathbb{Z}$ and $4\mathbb{Z}+2$ and partitions $2\mathbb{Z}+1$ into $4\mathbb{Z}+1$ and $4\mathbb{Z}+3$. A pictorial representation of the partition tree, $\mathbb{Z}/2\mathbb{Z}/4\mathbb{Z}$, is given in Figure 7.8. Each of the branches of the partition tree is labeled as either 0 or 1. This labeling defines a mapping between two-bit vectors and cosets of $4\mathbb{Z}$. For example, the coset $4\mathbb{Z}+1$ corresponds to the bit label 10. Now, if we allow the output of a rate-$\frac{1}{2}$ convolutional code to index the labeling of the partition tree for each time instant, then a trellis code may be formed from $\mathbb{Z}/4\mathbb{Z}$ by searching all rate-$\frac{1}{2}$ convolutional codes of a given constraint length to find the convolutional code that maximizes the minimum Euclidean distance between codewords. Note that in the aforementioned, a finite subset of the lattice points must be used to form the trellis code to ensure that there is no ambiguity in decoding a sequence of lattice points to a bit sequence.

The performance of a TCM code is measured by the signal-to-noise ratio (SNR) that is needed to achieve a given probability of error. For high SNRs, it has been

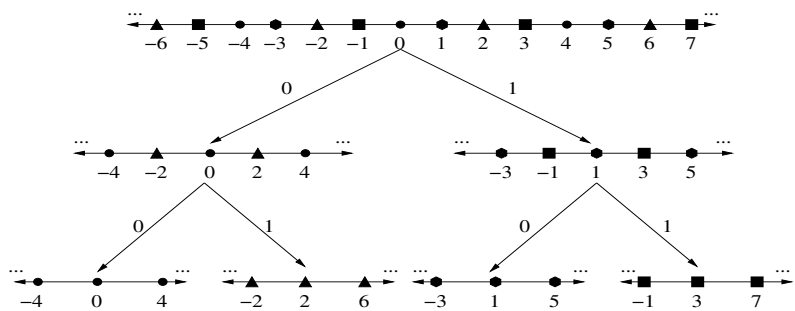


FIGURE 7.8: An example of the partition tree $\mathbb{Z}/2\mathbb{Z}/4\mathbb{Z}$. The branches of the tree are labeled by either 0 or 1 and represent a mapping from bits to cosets of $4\mathbb{Z}$.

shown that the probability of error for a TCM code can be approximated as

$$P_e \approx K_{\min} Q\left(\sqrt{\frac{d_{\min}^2}{4\sigma_N^2}}\right), \tag{7.14}$$

where Q is the Q function, $d_{\min}$ represents the minimum distance of the TCM code and $K_{\min}$ represents the number of codewords that have a distance of $d_{\min}$ from a given codeword. We use σ_N^2 to represent the variance of the channel noise. Though effective, the performance of TCM codes is considerably worse than information theoretic bounds.

7.3.3 Interleaving

While the codes described earlier are convenient for memoryless channels with small error rates, most of them are not suited to the protection against errors that occur in bursts. When errors occur in bursts, as in fading channels, a transmitted codeword is either free of errors or contains a large number of successive errors. The problem of burst errors can be alleviated with special codes (e.g., Fire codes [17]). An alternative is interleaving, which shuffles the symbols from different codewords before transmission. When a long burst error occurs, the erroneous symbols are distributed among many codewords where they appear as short burst errors. In block interleaving (Table 7.2), the channel codewords are placed in the rows of an array, and the codeword symbols are sent columnwise. In cross (or convolutional) interleaving, as shown in Figure 7.9, a set of ordered shift registers with linearly increasing memory size is used to separate the output symbols of the channel encoder.

7.3.4 Turbo Codes

In 1993, Berrou *et al.* [9,10] amazed the coding community by introducing a novel class of error-correcting codes, turbo codes, which, for a binary-input AWGN

Table 7.2: Block interleaver of size 4×7. To transmit four codewords of length 7, the codeword symbols are sent columnwise, in the order $1, 8, 15, 22, \ldots, 7, 14, 21, 28$. A burst error of length four produces no more than a single error in a transmitted codeword.

1. Codeword	1	2	3	4	5	6	7
2. Codeword	8	9	10	11	12	13	14
3. Codeword	15	16	17	18	19	20	21
4. Codeword	22	23	24	25	26	27	28

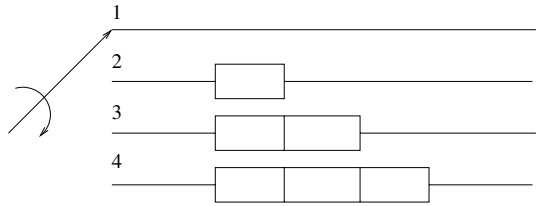


FIGURE 7.9: Cross interleaver with four shift registers. The memory sizes of the shift registers are 0, m, $2m$, and $3m$, respectively. At time unit i, a symbol is inserted into shift register i, which outputs its right-most symbol. Suppose that $m = 1$ and the input symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, ... After interleaving, the symbols are sent in the order 0, 4, 1, 8, 5, ...

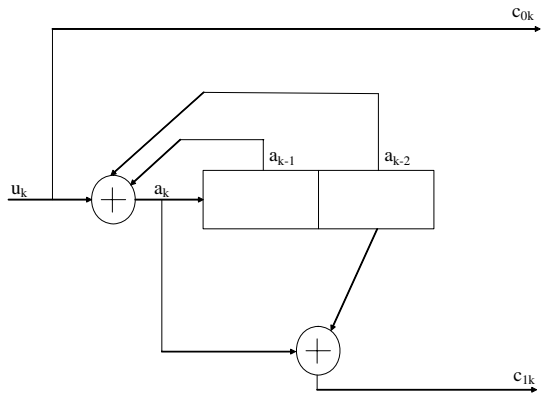


FIGURE 7.10: A recursive systematic convolutional code.

channel, achieved a BER of 10^{-5} with code rate $1/2$ and E_b/N_0 as close as 0.5 dB to the practical Shannon limit (see Section 7.2.4).

A turbo code is a parallel concatenation of two or more codes connected by pseudo-random interleavers. The constituent codes are usually identical, recursive systematic convolutional (RSC) codes of rate $1/2$. An example of an RSC encoder is shown in Figure 7.10. Its main property is the existence of a feedback in the shift-register realization.

Figure 7.11 shows a classical turbo encoder with two constituent RSC codes. In contrast to a serial code concatenation where the output of one encoder forms the input for the next one, in a parallel concatenation, both encoders operate on the same input block. In Figure 7.11, an input information block of length k bits,

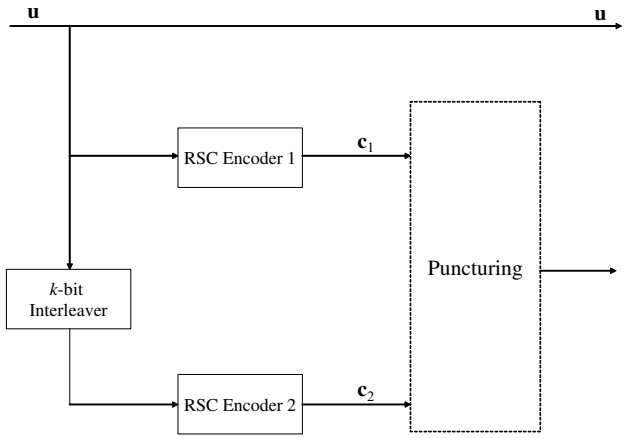


FIGURE 7.11: A classical turbo encoder with two RSC codes.

$\mathbf{u} = (u_1, \ldots, u_k)$, is encoded by the first RSC encoder; at the same time, it is passed through a k-bit interleaver and fed into the second RSC encoder. For each input bit u_i, the output consists of that bit u_i and the two parity-check bits $c_{1,i}$ and $c_{2,i}$ from the two RSC encoders. The output corresponding to the input block $\mathbf{u}$ is the codeword $\mathbf{c} = (u_1, \ldots, u_k, c_{1,1}, \ldots, c_{1,k}, c_{2,1}, \ldots, c_{2,k})$. The code rate of the turbo code is 1/3. Higher code rates can be obtained by puncturing the output bits of the two encoders [1]. For example, rate $1/2$ can be obtained by alternately puncturing the parity bits of the two RSC encoders. A turbo code is essentially a block code, thus encoding can be seen as a multiplication of the information block by a generator matrix.

One of the many novelties in the turbo code realization is the existence of a block interleaver between the two RSC coders. The interleaver introduces randomness to the code while leaving enough structure in it so that decoding is physically feasible. The size of the interleaver (the length of the information block) is usually very large (in the order of thousands bits) to ensure good performance. If the size is large enough, any pseudo-random interleaver will perform well. However, for short interleaver sizes, the performance of the code can be significantly enhanced by a clever design of the interleaver [5].

A typical turbo decoder consists of two soft-input soft-output decoders (see Section 7.3.2), two k-bit interleavers identical to the encoder interleaver, and a deinterleaver, as shown in Figure 7.12. The decoding is based on the symbol MAP rule (see Section 7.2.2). The a posteriori LLRs for the information bits $u_1, \ldots, u_k$ are estimated in an iterative way by exchanging information between the two constituent decoders. Suppose that the systematic part of the codeword, $\mathbf{c}_0 = (u_1, \ldots, u_k)$, is received as $\mathbf{y}_0$, while the two parity parts, $\mathbf{c}_1 = (c_{1,1}, \ldots, c_{1,k})$ and

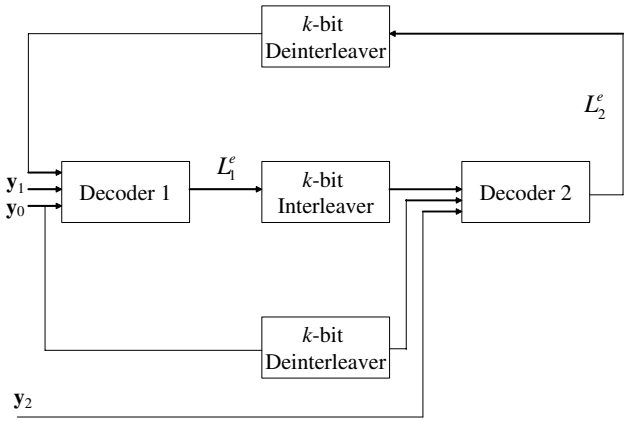


FIGURE 7.12: Block scheme of a classical turbo decoder.

$\mathbf{c}_2 = (c_{2,1}, \ldots, c_{2,k})$, are received as $\mathbf{y}_1$ and $\mathbf{y}_2$, respectively. In the first iteration, the first decoder generates a reliability information $L_1^e(i)$ for each information bit u_i, $i = 1, \ldots, k$, based on its input, $(\mathbf{y}_0, \mathbf{y}_1)$. This soft-decision output, called extrinsic information, is interleaved and fed to the second decoder. Using its input ($\mathbf{y}_2$ and the interleaved version of $\mathbf{y}_0$), the second decoder computes a reliability information $L_2^e(i)$ for each information bit. Next, the extrinsic information from the first decoder, the extrinsic information from the second decoder, and a channel log-likelihood ratio $\log \frac{Pr\{Y_i = y_{0,i} | U_i = 1\}}{Pr\{Y_i = y_{0,i} | U_i = 0\}}$ are summed to provide a first approximation of the a posteriori LLRs. In the second iteration, the extrinsic information $L_2^e(i)$ is deinterleaved and sent to the first decoder, which exploits this new information to update its extrinsic information. The procedure repeats until the a posteriori LLRs converge or a maximum number of iterations is reached.

Turbo coding with iterative decoding is currently one of the best error-correcting techniques. It significantly outperforms convolutional codes of the same constraint length. One of the key properties of turbo codes is the sharp performance improvement with the increase of the input block length. Thus, to achieve near-capacity performance, large block lengths are needed, which cause huge latency. Therefore, applications of turbo codes are currently limited to those that are not delay sensitive. For example, the new CCSDS telemetry channel coding standard for satellite and deep-space communications uses turbo codes. SMART-1, launched in September 2003 by the European Space Agency, is the first probe that exploits turbo codes. Turbo codes have also been adopted by the leading third-generation (3G) cellular standards, such as CDMA2000 and UMTS.

7.4 HIERARCHICAL MODULATION

Hierarchical modulation [19] is a digital modulation technique that enables transmission of two independent information bit streams with unequal priority on a single channel. As part of the digital terrestrial television standard DVB-T [18], it offers new possibilities in organizing scarce radio frequency bandwidth. In this section, we first outline the main concepts underlying hierarchical modulation and compare it to standard digital nonhierarchical modulation techniques; then, we give examples of possible applications.

Figure 7.13 shows constellations of four basic linear digital modulation techniques [38,39]. Each possible digital state (constellation point) in the phase diagram (represented by a dot in Figure 7.13) uniquely determines one phase of the carrier signal. Each transmitted bit stream is assigned to one constellation point. The performance of a digital modulation technique can be measured using its achieved data rate (or, equivalently, the number of bits assigned to each dig-

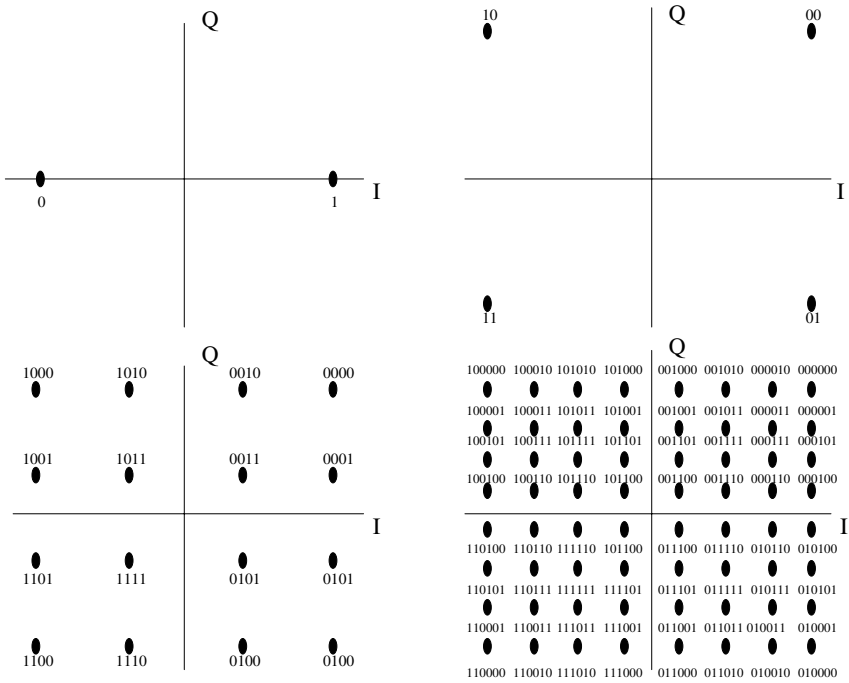


FIGURE 7.13: Constellations of four standard digital modulation techniques: BPSK (top left), 4-QAM (top right), 16-QAM (bottom left), and 64-QAM (bottom right).

ital state) and minimum tolerated signal-to-noise ratio for reliable demodulation (which reflects robustness to channel noise). Normally, higher level modulation techniques achieve larger data rates at the expense of a lower robustness.

Binary Phase Shift Keying (BPSK) allows transmission of one bit per modulation signal. The phase of a carrier signal takes two possible values (separated by π) depending on the transmitted bit. 4-Quadrature Amplitude Modulation (4-QAM), also referred to as Quadrature Phase Shift Keying (QPSK or 4-PSK), transmits two bits on each carrier. Thus, it achieves twice the data rate of BPSK. In Figure 7.13, one possible constellation realization is presented, where the carrier phases are $\pi/4$, $3\pi/4$, $5\pi/4$, and $7\pi/4$. In 16-QAM and 64-QAM, because there are 4×4 and 8×8 different constellation points, respectively, four and six bits, respectively, can be sent per modulation signal. The assignment of the bit streams to the digital states is usually determined using Gray-code mapping so that the assignments of the closest constellation points differ in one bit. The data rate is increased compared to 4-QAM at the expense of a lower noise tolerance (due to smaller distances between neighboring states in the phase diagram). For example, compared to 4-QAM with the same code rate, the minimum tolerated signal-to-noise ratio is approximately 6 dB and 12 dB higher with 16-QAM and 64-QAM, respectively [19].

Note that in all modulation techniques discussed so far, a single information bit stream (possibly coded) is transmitted per one modulation signal. Hierarchical modulation, however, enables transmission of two separate information bit streams in a single modulation signal. One bit stream, called high-priority (HP) bit stream, is embedded within another, called low-priority (LP) bit stream. The main idea is to decouple the bit stream assigned to a digital state into two substreams: the first substream is HP, which determines the number of the quadrant (0, 1, 2, or 3) where the digital state is located; the second substream (LP) carries the information about the position of the digital state in the specified quadrant. As a result, hierarchical modulation can be viewed as a combination of 4-QAM (used for the HP bit stream) and either 4-QAM or 16-QAM (used for the LP bit stream).

Two hierarchical modulation constellations are shown in Figure 7.14. In the first constellation (the upper figure), 4-QAM is embedded in 16-QAM (thus, it is called "4-QAM in 16-QAM"); in the second one, 4-QAM is embedded in 64-QAM ("4-QAM in 64-QAM"). In both cases, the first two bits constitute the HP bit stream intended for an HP service/client; the remaining two or four bits are the LP bit stream intended for an LP service/client. In the example shown in Figure 7.14 (bottom), 10 is sent to the HP clients and 0101 to the LP clients.

Note that the HP bit stream is always modulated as 4-QAM. Thus, as in classic nonhierarchical 4-QAM, it carries two bits per modulation signal. However, because the LP bit stream can be seen at the receiver as an additional noise in the

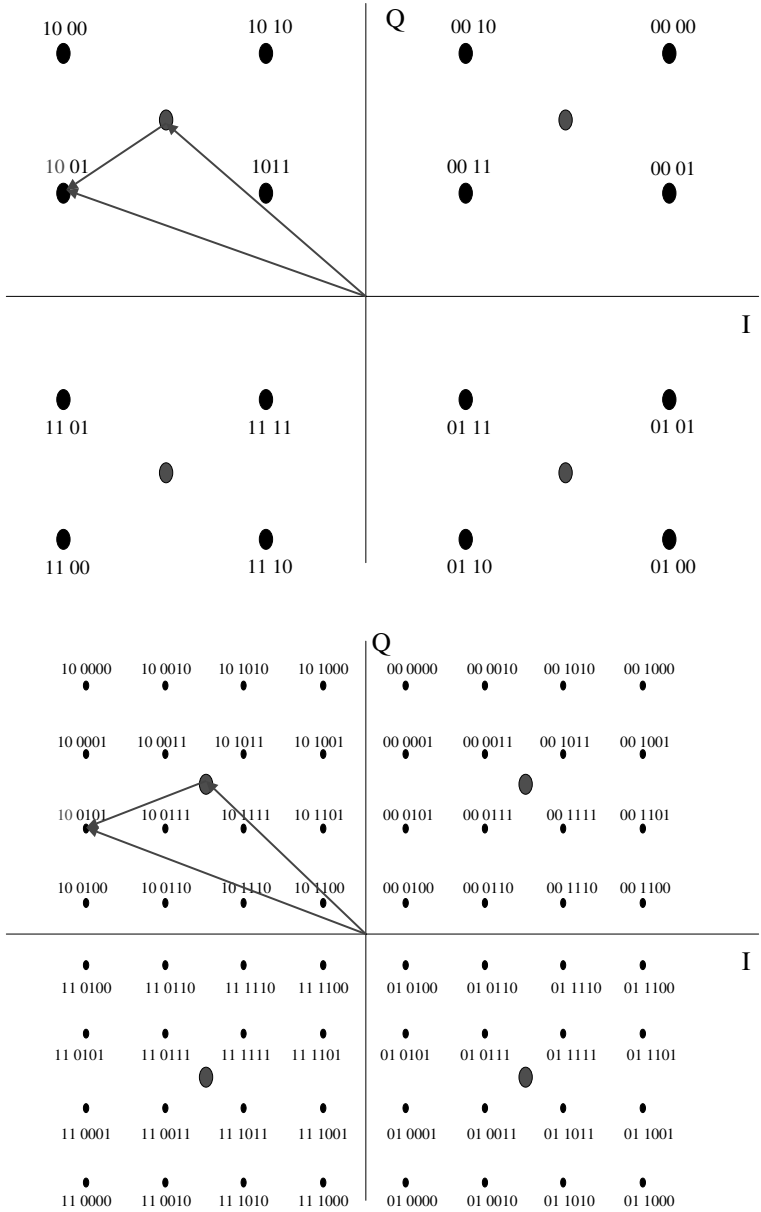


FIGURE 7.14: Hierarchical modulation: "4-QAM in 16-QAM" (top) and "4-QAM in 64-QAM" (bottom).

quadrant of the received signal, the HP bit stream is less robust than nonhierarchical 4-QAM (i.e., a higher minimum tolerated signal-to-noise ratio is needed).

The LP bit stream is essentially either 4-QAM [Figure 7.14 (top)] or 16-QAM [Figure 7.14 (bottom)] modulated. Thus, it carries two or four bits and has the same data rate as the corresponding nonhierarchical modulation method. The noise sensitivity is comparable to that of the whole constellation [16-QAM in Figure 7.14 (top) or 64-QAM in Figure 7.14 (bottom)]. Note that the total rate of the HP and LP bit streams is equal to the rate of the whole nonhierarchical constellation (16-QAM or 64-QAM).

The HP bit stream is obviously more robust to channel noise than the LP bit stream; indeed, a transition of the carrier phase (due to channel noise) from one digital state to the other within a quadrant is more likely to occur than a transition to a state in another quadrant. However, the robustness of the HP and LP bit streams can be further improved by channel coding (i.e., by adding error protection) or by changing the constellation's α factor, as in Figure 7.15. The $\alpha = a/b$ factor [18] is defined as the ratio between a, the minimum distance separating two constellation points that carry two different HP bit streams, and b, the minimum distance separating any two constellation points. Constellations with $\alpha > 1$ are called nonuniform constellations. The increase of α makes the HP bit stream more robust at the expense of a less robustness of the LP bit stream. (The DVB-T standard uses $\alpha \in \{1, 2, 4\}$.) Thus, hierarchical modulation splits the actual communication channel in two virtual channels whose characteristics depend on the

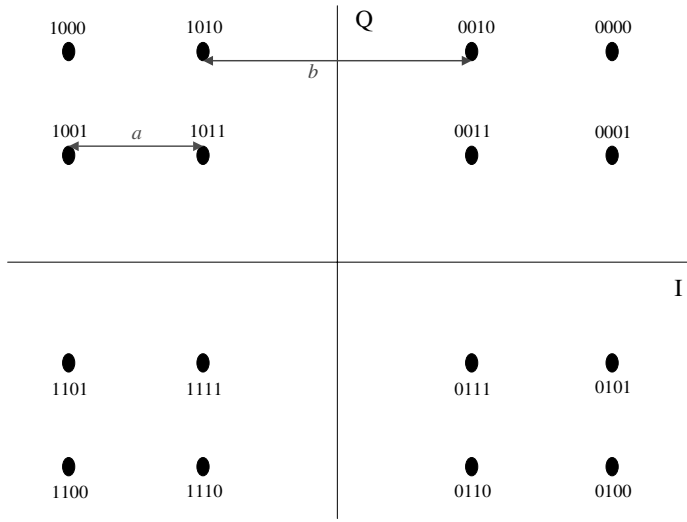


FIGURE 7.15: A nonuniform 16-QAM constellation with $\alpha = b/a = 2$.

whole constellation (64-QAM), α factor, and code rates of the HP and LP bit streams.

Hierarchical modulation was originally proposed to enable two different coverage areas for a given transmitter in digital terrestrial TV. It offers great design flexibilities and simplifies network planning. Its value has become even more apparent with recent increasing demands for delivery of different services over heterogeneous networks, where communication channels between the sender and the clients are extremely diverse in available bandwidths and channel noise.

For example, suppose that two digital TV programs are to be transmitted simultaneously. With nonhierarchical modulation, the two programs must be broadcast over two separate frequency channels: 4-QAM can be used for the first channel (achieving a data rate of two bits per modulation signal) and 16-QAM for the second channel (with a data rate of four bits per modulation signal). With hierarchical modulation ("4-QAM in 64-QAM"), only one channel is needed: the first program can be transmitted as an HP bit stream (at a data rate of two bits per modulation signal), while the second TV program can be transmitted as an LP bit stream (at a data rate of four bits per modulation signal). Then, the coverage radius (which is determined by the noise tolerance) of the second TV program will be roughly the same as in the nonhierarchical case; the coverage radius of the first program, however, will be smaller than with nonhierarchical 4-QAM, but can be enlarged by increasing the α factor (at the expense of a smaller coverage radius of the second TV program) or by using error protection (at the expense of decreasing the information rate). Thus, one immediate advantage of hierarchical modulation over a nonhierarchical one is the savings in transmission channels because two streams with different data rates and different coverage areas can be transmitted on a single frequency channel.

Hierarchical modulation efficiently addresses the problem of heterogeneity in clients' available bandwidths, receiver resolution capabilities, and channel conditions. For example, a single frequency channel can be used to broadcast a video bit stream to mobile (or portable) receivers and fixed receivers. The mobile receivers will decode the HP bit stream, whereas the fixed receivers will be able, in addition, to decode the LP bit stream (due to their large roof top antenna gains).

Hierarchical modulation can be combined with quality/resolution scalable video coders. Then, the LP bit stream plays the role of the enhancement layer, which improves the quality/resolution of the HP (base layer) bit stream. Depending on transmission conditions, the receiver will be able to decode at the higher or lower quality/resolution level.

Another application is simulcast of the High Definition TV formats, together with the Standard Definition formats. (Transmitting the Standard Definition together with the High Definition formats is necessary because all the receivers do not have screens that support the latter formats.) Here, the HP bit stream carries

the Standard Definition TV formats, and thus will be available to all receivers, whereas the LP bit stream carries the High Definition TV formats only.

Comparisons between hierarchical and nonhierarchical modulation in different scenarios can be found in [42].

7.5 AUTOMATIC REPEAT REQUEST, HYBRID FEC/ARQ

In this section, we present error protection techniques that use retransmissions. Here we assume the presence of a feedback channel from the receiver to the transmitter. We first describe pure ARQ techniques, which are based on error detection and retransmission of the corrupted packets. Then we explain type I hybrid ARQ protocols that combine error correction coding and ARQ techniques. Finally, we overview type II hybrid-ARQ protocols where the transmitter answers a retransmission request by sending additional parity symbols.

7.5.1 Pure ARQ Protocols

In a pure ARQ system, an information block of length k is encoded into a channel codeword of length n with an error-detecting code. The codeword is sent over the channel and the received word is decoded. If no errors are detected, the transmitted codeword is assumed to be received correctly and need not be retransmitted. Otherwise, the codeword must be sent again until it is received correctly. To send feedback information to the transmitter, the receiver can use a positive acknowledgment (ACK) to indicate that the codeword was received correctly or a negative acknowledgment (NACK) to indicate a transmission error. The efficiency of an ARQ scheme is measured by its reliability and throughput. The reliability is the probability that the receiver accepts a word that contains an undetectable error. The throughput is the ratio of the average number of bits successfully accepted per unit of time to the total number of bits that could be transmitted per unit of time [29]. In the following, we overview the most important ARQ schemes. Details can be found in [29] and [50].

7.5.1.1 Stop-and-Wait ARQ

In stop-and-wait ARQ, the transmitter sends a codeword and waits for an acknowledgment for that codeword. If an ACK is received, the next codeword is sent. If an NACK is received, the same codeword is retransmitted until it is received correctly, as in Figure 7.16. Stop-and-wait ARQ has a simple implementation. In particular, the codewords are not numbered. Its major drawback is the idle time spent by the transmitter waiting for an ACK.

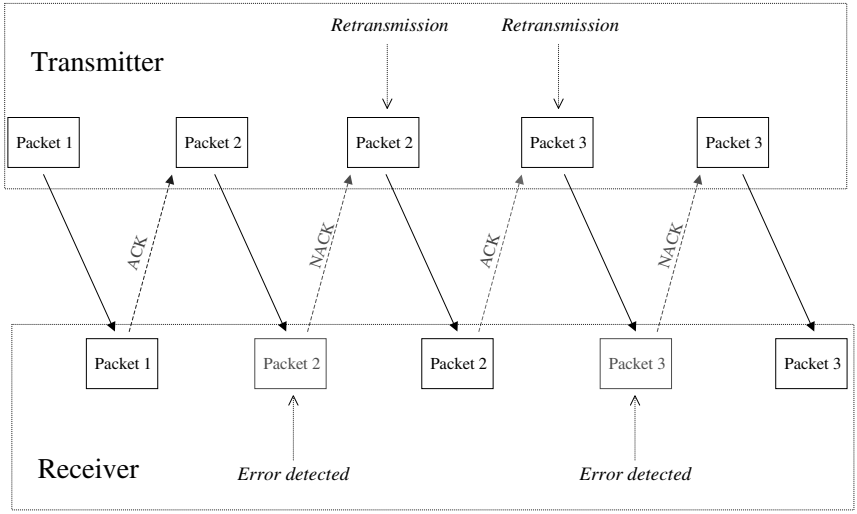


FIGURE 7.16: Stop-and-wait ARQ.

7.5.1.2 Go-Back-N ARQ

In go-back-N ARQ, the transmitter sends the codewords continuously without waiting for an acknowledgment. Suppose that the acknowledgment for codeword c_i arrives after codewords $c_i, \ldots, c_{i+N-1}$ have been sent. If this acknowledgment is of the ACK type, the transmitter sends codeword c_{i+N}. Otherwise, the codewords $c_i, \ldots, c_{i+N-1}$ are sent again, as in Figure 7.17. On the receiver side, when an error is detected in a received word, this word and the $N-1$ subsequently received ones are ignored. Note that a buffer for N codewords is required at the transmitter side.

7.5.1.3 Selective-Repeat ARQ

Selective-repeat ARQ is similar to go-back ARQ. The difference is that when an NACK for codeword c_i is received, only c_i is retransmitted before the transmission proceeds where it stopped, as in Figure 7.18. In addition to the N-codeword buffer at the transmitter, a buffer is needed at the receiver so that the decoded codewords can be delivered in the correct order. This buffer must be large enough to avoid overflow. Selective-repeat ARQ with a finite-size buffer is presented in [29]. An alternative is to combine selective-repeat ARQ with go-back-N ARQ [29]. Here the transmitter switches from selective-repeat ARQ to go-back-N ARQ whenever μ retransmissions of a codeword have been done without receiving an

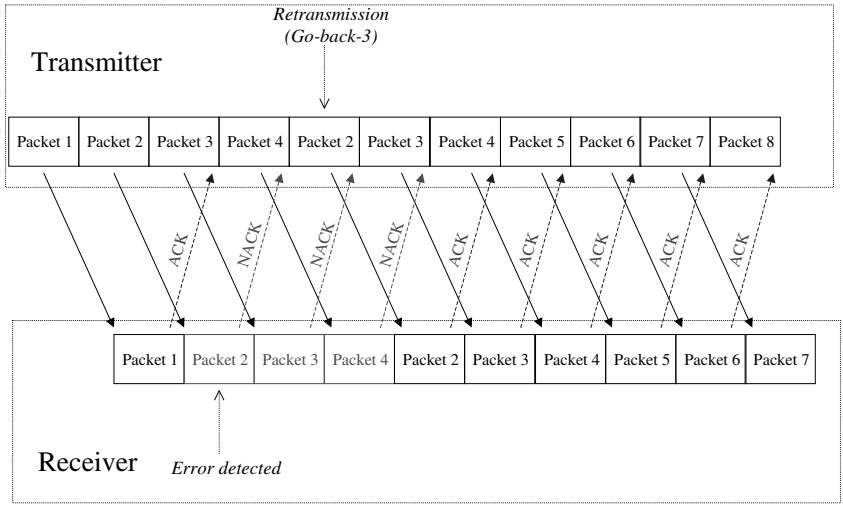


FIGURE 7.17: Go-back-*N* ARQ.

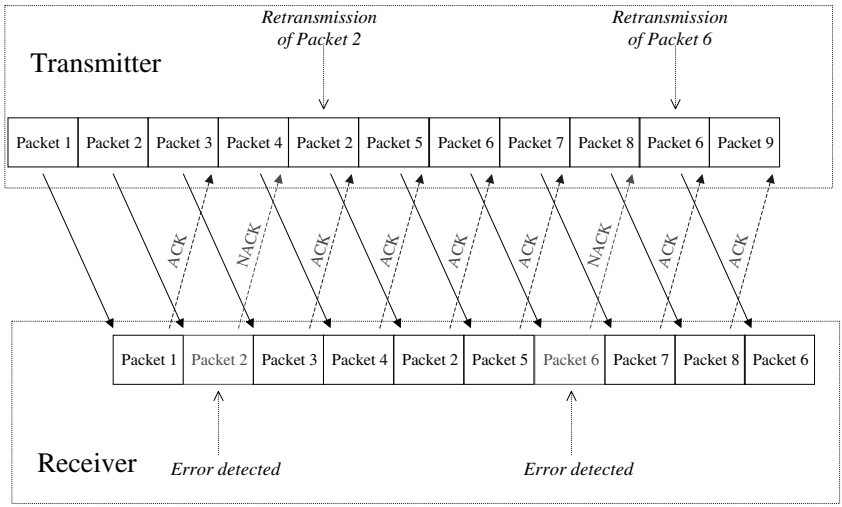


FIGURE 7.18: Selective-repeat ARQ.

ACK. It switches back to selective-repeat ARQ as soon as an ACK is received. In this way, the buffer size of the receiver can be limited to $\mu(N-1)+1$.

7.5.2 Hybrid ARQ Protocols

FEC and ARQ can be combined to provide for channels with high error rates better reliability than FEC alone and larger throughput than ARQ alone.

7.5.2.1 Type-I Hybrid ARQ Protocols

In a type-I hybrid ARQ system, each information block is encoded with a channel code with error detecting and error correcting capabilities. This can be a single linear code (see Section 7.3.1) or a concatenation of an error detection code as an outer code and an error correction code as an inner code. If the received word can be correctly decoded, then the decoded codeword is accepted. Otherwise, a retransmission is requested for the codeword.

7.5.2.2 Type-II Hybrid-ARQ Protocols

The basic difference between a type-I hybrid ARQ protocol and a type-II hybrid ARQ protocol is that in the latter the transmitter sends additional parity bits instead of the whole codeword when it receives a retransmission request for this codeword. The following example [50] illustrates the method. An (n, k) MDS code $\mathcal{C}$ is used to encode the information block. The resulting codeword is split in two. The first half can be seen as a codeword $\mathbf{c}_1$ from an $(n/2, k)$ code $\mathcal{C}_1$ and the second one as a codeword $\mathbf{c}_2$ from an $(n/2, k)$ code $\mathcal{C}_2$. Here the two codes $\mathcal{C}_1$ and $\mathcal{C}_2$ are obtained by puncturing the code $\mathcal{C}$. The transmitter starts by sending $\mathbf{c}_1$. If the received word $\mathbf{y}_1$ cannot be correctly decoded, a retransmission is requested. The transmitter then sends the codeword $\mathbf{c}_2$, which is received as $\mathbf{y}_2$. The receiver concatenates $\mathbf{y}_1$ and $\mathbf{y}_2$ and uses the stronger code $\mathcal{C}$ to decode the resulting word.

7.6 SUMMARY AND FURTHER READING

The first part of this chapter presented the fundamental results of information theory, which culminate in Shannon's joint source-channel coding theorem. While this theorem is useful in understanding the theoretical performance bounds for the communication of data over an unreliable channel, it does not explain how a practical communication system should be designed. Practical system design should consider source coding, channel control, and modulation. Practical source coding for media data is described in other chapters of this book (Chapter 5 for video coding and Chapter 6 for audio coding). State-of-the-art channel coding techniques are overviewed in the second part of the chapter. The main message is

that channel coding techniques, in particular Turbo codes and LDPC codes, have reached a level of maturity that allows them to achieve performance close to the theoretical bounds announced by Shannon. Another important achievement in the area of channel coding is development of the class of digital fountain codes for protection against packet loss. The third part of the chapter discussed hierarchical modulation, an emerging modulation technique for digital video broadcasting. The last part of the chapter gave a brief survey of error control techniques that rely on data retransmission. These techniques, which require a two-way channel, can be used with error detection only or combined with error correcting codes.

We conclude this chapter with suggestions for further reading. A rigorous treatment of source coding can be found in [6] and [17]. Excellent descriptions of modern channel codes are given in [43] and [33]. The best reference for the latest advances in source and channel coding is the IEEE Transactions on Information Theory.

REFERENCES

[1] O. Acikel and W. Ryan. "Punctured turbo-codes for BPSK/QPSK channels," *IEEE Trans. Commun.*, vol. 47, pp. 1315–1323, September 1999.

[2] V. K. Agarwal and A. Ivanov. "Computing the probability of undetected error for shortened cyclic codes," *IEEE Trans. Commun.*, vol. 40, pp. 494–499, March 1992.

[3] R. B. Ash. *Information Theory*, Dover, New York, 1965.

[4] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, pp. 284–287, March 1974.

[5] A. Barbulescu and S. Pietrobon. "Interleaver design for turbo codes," *Electronics Letters*, vol. 30, pp. 2107–2108, December 1994.

[6] T. Berger. *Rate Distortion Theory*, Prentice Hall, 1971.

[7] E. R. Berlekamp. *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

[8] E. R. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg. "On the intractability of certain coding problems," *IEEE Trans. Inform. Theory*, vol. 24, pp. 384–386, May 1978.

[9] C. Berrou and A. Glavieux. "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Commun.*, vol. 44, pp. 1261–1271, October 1996.

[10] C. Berrou, A. Glavieux, and P. Thitimajshima. "Near Shannon limit error-correcting coding and decoding: Turbo codes," *Proc. IEEE ICC-1993 International Conference on Communications*, pp. 1064–1070, Geneva, Switzerland, May 1993.

[11] S. A. Butman and R. J. McEliece. "The ultimate limits of binary coding for a wideband Gaussian channel," JPL Deep Space Network Progress Report 42–22, pp. 78–80, 1974.

[12] L. N. Childs. *A Concrete Introduction to Higher Algebra*, Springer, New York, 1995.

[13] J. Conway and N. Sloane. *Sphere Packings and Error-Correcting Codes*, Springer-Verlag, New York, 1988.

[14] S.-Y. Chung, G. D. Forney, T. J. Richardson, and R. Urbanke. "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Commun. Letters*, vol. 5, pp. 58–60, February 2001.

[15] T. M. Cover and J. A. Thomas. *Elements of Information Theory*, Wiley, 1991.

[16] D. Divsalar, H. Jin, and R. J. McEliece. "Coding theorems for 'turbo-like' codes," *Proc. 36th Allerton Conf. Communication, Control, and Computing*, pp. 201–210, Allerton, Illinois, September 1998.

[17] R. J. McEliece. *The Theory of Information and Coding*, Cambridge University Press, 2002.

[18] ETSI EN 300 744: Digital video broadcasting (DVB); framing structure, channel coding and modulation for digital terrestrial television, June 2004.

[19] ETSI: Digital video broadcasting (DVB); implementation guidelines for DVB terrestrial services; transmission aspects, Technical Report TR 101 190, December 1997.

[20] G. Forney. "Coset codes – part 1: Introduction and geometrical classification," *IEEE Trans. Inform. Theory*, vol. 34, pp. 1123–1151, September 1988.

[21] R. G. Gallager. *Low Density Parity-Check Codes*, MIT Press, Cambridge, 1963.

[22] R. Gray. *Entropy and Information Theory*, Springer-Verlag, New York, 1990.

[23] V. Guruswami and M. Sudan. "Improved decoding of Reed–Solomon and algebraic-geometry codes," *IEEE Trans. Inform. Theory*, vol. 45, pp. 1757–1767, September 1999.

[24] J. Hagenauer. "Rate-compatible punctured convolutional codes (RCPC codes) and their applications," *IEEE Trans. Commun.*, vol. 36, pp. 389–400, April 1988.

[25] J. Hagenauer and P. Hoeher. "A Viterbi algorithm with soft-decision outputs and its applications," *Proc. GLOBECOM*, vol. 3, pp. 1680–1686, Dallas, Texas, November 1989.

[26] J. Jiang and K. Narayanan. "Iterative soft decoding of Reed–Solomon codes," *IEEE Commun. Letters*, vol. 8, pp. 244–246, April 2004.

[27] H. Jin, A. Khandekar, and R. McEliece. "Irregular repeat-accumulate codes," *Proc. 2nd Int. Symposium. Turbo Codes*, pp. 1–8, Brest, France, September 2000.

[28] R. Koetter and A. Vardy. "Algebraic soft-decision decoding of Reed–Solomon codes," *IEEE Trans. Inform. Theory*, vol. 49, pp. 2809–2825, November 2003.

[29] S. Lin and D. Costello, Jr. *Error Control Coding*, 2nd ed., Prentice Hall, 2004.

[30] M. Luby. "LT codes," *Proc. 43rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 271–282, 2002.

[31] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. "Practical loss-resilient codes," *29th ACM Symposium Theory Computation*, pp. 150–159, 1997.

[32] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman. "Efficient erasure correcting codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 569–584, February 2001.

[33] D. J. C. MacKay. *Information Theory, Inference and Learning Algorithms*, Cambridge University Press, 2003.

[34] D. J. C. MacKay and R. M. Neal. "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, pp. 1645–1646, August 1996.

[35] D. J. C. MacKay. "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–431, March 1999.

[36] M. MacWilliams and N. Sloane. *The Theory of Error-Correcting Codes*, Karlin, North-Holland, 1992.

[37] J. Pearl. "Fusion, propagation, and structuring in belief networks," *Artificial Intell.*, vol. 29, pp. 241–288, 1986.
[38] J. G. Proakis and M. Salehi. *Communication Systems and Engineering*, Prentice-Hall, New Jersey, 2002.
[39] T. S. Rappaport. *Wireless Communications*, Prentice-Hall, New Jersey, 1996.
[40] T. J. Richardson and R. Urbanke. "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 638–656, February 2001.
[41] M. Röder and R. Hamzaoui. "Fast tree-trellis list Viterbi algorithm," *IEEE Trans. Commun.*, vol. 54, pp. 453–461, March 2006.
[42] A. Schertz and C. Weck. "Hierarchical modulation," *EBU Technical Review*, April 2003.
[43] C. Schlegel. *Trellis Coding*, John Wiley & Sons, 1997.
[44] N. Seshadri and C.-E. W. Sundberg. "List Viterbi decoding algorithms with applications," *IEEE Trans. Commun.*, vol. 42, pp. 313–323, February–April 1994.
[45] C. E. Shannon. "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, 1948.
[46] C. E. Shannon. "Coding theorems for a discrete source with a fidelity criterion," *I.R.E. Nat. Conv. Rec.*, part 4, pp. 142–163, 1959.
[47] A. Shokrollahi. "Raptor codes," *IEEE Trans. Inform. Theory*, vol. 52, pp. 2551–2567, June 2006.
[48] G. Ungerboeck. "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory*, vol. 28, pp. 55–67, January 1982.
[49] N. Wiberg, H.-A. Loeliger, and R. Kötter. "Codes and iterative decoding on general graphs," *Eur. Trans. Telecommun.*, vol. 6, pp. 513–525, September–October 1995.
[50] S. Wicker. *Error Control Systems for Digital Communication and Storage*, Prentice-Hall, New Jersey, 1995.

8

Channel Modeling and Analysis for the Internet

Hayder Radha and Dmitri Loguinov

8.1 INTRODUCTION

Performance modeling and analysis of channels and networks play a crucial role in the design and development of multimedia applications. In particular, having an insight into the expected number of packet losses, which could occur when sending video or audio content over the Internet, provides multimedia application designers an important premise for developing resilience techniques to protect that content. Furthermore, real-time multimedia applications are sensitive to end-to-end delay parameters, including delay jitter. These parameters influence the particular techniques used for recovering lost packets. For example, depending on the application and its level of tolerance for end-to-end delay, the application designer may choose to adopt a strategy for recovering lost packets that is based on retransmission, Forward Error Correction (FEC), or both.

This chapter covers fundamental analysis tools that are used to characterize the loss performance of channels and networks that carry multimedia packets. We focus on models and analysis tools for Internet multimedia applications. In addition to performance analysis and modeling tools, experimental performance studies are crucial for designing multimedia applications and services. Hence, this chapter consists of two major parts. The first part emphasizes core and relatively simple analysis tools that lead to key results and widely used formulas. Although some of these results and formulas are basic, rather abstract, and generic in nature (i.e., applicable to a variety of applications), their use for performance analysis of multimedia applications is invaluable. The second part of this chapter describes a comprehensive Internet video study conducted for gaining insight into a variety of end-to-end performance parameters that are crucial for real-time multimedia applications. The study reveals many interesting and practical issues, and it provides

significant insight that is difficult (if not impossible) to gain based on pure analysis or modeling. The later (second) part also analyzes the performance parameters collected from the aforementioned Internet video study.

The analytical tools needed for characterizing channels and networks lie within basic concepts from probability theory, random processes, and information theory. Here, it is assumed that the reader has the appropriate background in probability theory and random processes. We later focus on some of the key, basic and relevant concepts and definitions from information theory that can be used for characterizing Internet links and routes. For popular Internet multimedia applications, packet losses represent the most crucial performance parameter. The information theory concepts covered in this chapter identify *performance bounds* for given loss measures.

8.2 BASIC INFORMATION THEORY CONCEPTS OF CHANNEL MODELS

Information theory [1–3] provides core channel models that are used to represent a wide range of communication and networking scenarios. We begin by highlighting the information-theoretic definition of a discrete memoryless channel (DMC) and then focus on simple DMC channel models applicable to basic links and routes over the Internet (Figure 8.1).

A DMC is characterized by the relationship between its input X and its output Y, where X and Y are two (hopefully) dependent random variables. Therefore, a DMC is usually represented by the conditional probability $p(y|x)$ of the channel output Y given the channel input X. Furthermore, and since X and Y are dependent on each other, their mutual information $I(X;Y)$ has a nonzero (i.e., strictly positive) value,

$$I(X;Y) = \sum_{x}\sum_{y} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} > 0.$$

An important measure is the maximum amount of information that Y can provide about X for a given channel $p(y|x)$. This measure can be evaluated by maximizing the mutual information $I(X;Y)$ over all possible sources characterized by the marginal probability mass function $p(x)$ of the channel input X. This maximum measure of the mutual information is known as the "information" channel capacity C:

$$C = \max_{p(x)} I(X;Y).$$

Based on this definition, the channel capacity C is a function of the parameters that characterize the conditional probability $p(y|x)$ between the channel input X and the channel output Y. The following section focuses on a particular channel.

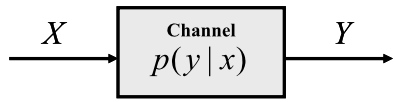


FIGURE 8.1: A representation of a DMC channel.

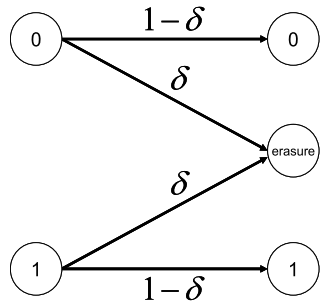


FIGURE 8.2: A representation of the Binary Erasure Channel.

8.2.1 The Binary Erasure Channel (BEC) Channel

The simplest DMC channel model that could be used for representing an Internet link or route is the Binary Erasure Channel (Figure 8.2). The BEC is characterized by the following.

- The input X is a binary (Bernoulli) random variable that can be either a zero or a one.
- A loss parameter δ, which represents the probability that the input is lost ("erased" or "deleted") when transmitted over the BEC channel.
- The output Y is a ternary random variable that could take on one of three possible values: zero, one, or "erasure." The latter output occurs when the channel loses the transmitted input X.

More specifically, a BEC is characterized by the conditional probability measures

$$\begin{aligned}
&\Pr\big[Y = \text{“}erasure\text{”} \mid X = 0\big] = \delta \quad \text{and} \quad \Pr\big[Y = \text{“}erasure\text{”} \mid X = 1\big] = \delta, \\
&\Pr\big[Y = 0 \mid X = 0\big] = 1 - \delta \quad \text{and} \quad \Pr\big[Y = 1 \mid X = 1\big] = 1 - \delta, \\
&\Pr\big[Y = 1 \mid X = 0\big] = 0 \quad \text{and} \quad \Pr\big[Y = 0 \mid X = 1\big] = 0.
\end{aligned}$$

Therefore, no errors occur over a BEC, as $\Pr[Y = 0 \mid X = 1] = \Pr[Y = 1 \mid X = 0] = 0$.

Due to the loss symmetry of the BEC (i.e., the conditional probability of losing a bit is independent of the bit value), it can be easily shown that the overall loss probability is also the parameter δ. In other words,

$$\Pr[Y = \text{“}erasure\text{”}] = \delta.$$

By using the definition of information channel capacity $C = \max_{p(x)} I(X; Y)$, it can be shown [1,2] that the channel capacity of the BEC is a rather intuitive expression,

$$C = 1 - \delta.$$

This capacity, which is measured in “bits” per “channel use,” can be achieved when the channel input X is a uniform random variable with $\Pr[X = 0] = \Pr[X = 1] = 1/2$.

Despite its simplicity, the BEC provides a rough, yet very useful estimate of the maximum throughput one can achieve over an Internet link, or an end-to-end route between a server and a client. For example, if one measures the average packet loss probability over a link or route to be δ, then the throughput of the route is $1 - \delta$, which is the same as the capacity of a BEC with parameter δ. The next sections expand on the basic BEC channel in three aspects that provide more realistic modeling of practical links and routes: (1) cascaded channels, (2) channels with input vectors of bits (i.e., packets) rather than binary bits, and (3) channels with feedback from the receiver to the transmitter.

8.2.2 Cascaded BEC Channels

Packets that carry multimedia content usually traverse multiple links over a path between the source and the receiver. Hence, these links can be modeled as cascaded channels, and here we begin with cascaded BEC channels. First, let’s assume that we have two BEC channels that are in cascade with each other. This, for example, could represent two Internet links over which multimedia packets are routed. The two BEC channels could have different loss probabilities,

$$\Pr\big[Y_1 = \text{“}erasure\text{”} \mid X_1\big] = \delta_1 \quad \text{and} \quad \Pr\big[Y_2 = \text{“}erasure\text{”} \mid X_2\big] = \delta_2,$$

where (X_1, Y_1) and (X_2, Y_2) are the input–output pairs for the first and second links, respectively. In this case, we know that the maximum throughput (as measured by the channel capacity) that can be received at the output Y_1 of the first channel is $C_1 = 1 - \delta_1$. Hence, the second link can be used only $(1 - \delta_1)$ fraction of the time. We also know that the maximum throughput of the second link is $C_2 = 1 - \delta_2$. Therefore, the overall throughput of the cascaded channel is

$C = (1 - \delta_1)(1 - \delta_2)$. This channel capacity assumes that the two channels are independent of each other and that both are DMC channels.

This result can be generalized to L cascaded links of BEC-independent channels. In this case, each link could have a different loss probability,

$$\Pr\big[Y_i = \text{“}\textit{erasure}\text{”} \mid X_i\big] = \delta_i, \quad i = 1, 2, \ldots, L.$$

The overall channel capacity of the L cascaded BEC links is

$$C = \prod_{i=1}^{L} (1 - \delta_i).$$

This end-to-end path of L BEC channels is equivalent to a BEC channel with an effective end-to-end loss probability

$$\delta = 1 - \prod_{i=1}^{L} (1 - \delta_i).$$

Note that $C = 1 - \delta = 1 - (1 - \prod_{i=1}^{L}(1 - \delta_i))$. As in the single channel case, the capacity in this cascaded case is measured in bits per channel use. Moreover, the overall capacity C of the end-to-end route is bounded by the capacity $C_{\min}$ of the link with the smallest capacity among all cascaded channels in the route. In other words, $C \leq C_{\min} = \min_i(C_i)$. Hence, knowledge of the minimum capacity link provides an easy way for identifying the performance bound of the end-to-end route. As mentioned earlier, it is important to note that this bound is measured in terms of “per channel use.” Therefore, $C_{\min}$ should not be confused by the “bottleneck” bandwidth $B_{\min}$ that is commonly referred to by the networking community. In this case, the bottleneck bandwidth usually represents the maximum transmission rate (e.g., 1.544 Megabits per second for a T1 line) that a particular link within the end-to-end path could support, and where this link has the minimum transmission rate: $B_{\min} = \min_i(B_i)$. Here, B_i can be thought of as the number of channel uses per second for link i. In general, a multimedia application must use a total rate R_{total} taking into consideration both the bottleneck bandwidth and the minimum end-to-end capacity. For example, let’s assume that the transmission rates and link bandwidths are measured in bits per second. We also know that a BEC link is based on a “per channel use” where “channel use” is measured in bits (i.e., every time we use the BEC channel, we are transmitting a bit). Hence, the effective (maximum) throughput of a BEC link i in terms of bits per second can be expressed as $R_i = B_i C_i$. Hence, the total rate R_{total} used by an application should be bounded by the following effective performance throughput: $R_{\text{total}} \leq \min_i(R_i) = \min_i(B_i C_i)$ in bits per second.

8.2.3 The "Packet" Erasure Channel (PEC)

A simple generalization of the BEC is needed to capture the fact that multimedia content is usually packetized and transmitted over Internet links as "integrated vectors" of bits rather than individual bits. In other words, when a multimedia packet is lost, that packet is lost in its totality. Hence, for bits that belong to the same packet, these bits are 100% dependent on each other: either all the bits are transmitted successfully (usually without errors) or all the bits are erased (e.g., lost due to congestion).

We refer to this simple generalization as the Packet Erasure Channel. (In some literature, this type of channel may be referred to as an M-ary Erasure Channel as a generalization of the Binary Erasure Channel.) In this case, the input is a vector of random variables: $\overline{X} = (X_1, X_2, \ldots, X_n)$, where each element X_i is a binary random variable. The output of the channel includes the possible "erasure" outcome and all possible input vectors. In other words, we have the following conditional probability measures for the PEC:

$$\Pr\left[\overline{Y} = \text{"}\textit{erasure}\text{"} \mid \overline{X}\right] = \delta \quad \text{and} \quad \Pr\left[\overline{Y} = \overline{X} \mid \overline{X}\right] = 1 - \delta.$$

Note that these conditional probability measures are independent of the particular input vector $\overline{X}$ (i.e., packet). Consequently, it is not difficult to show that the PEC has the same basic measures, such as channel capacity, as the BEC. Therefore, $C = 1 - \delta$. The capacity in this case is measured in "packets" per "channel use." Similarly, a cascade of L links of PEC channels has an effective loss probability $\delta = 1 - \prod_{i=1}^{L}(1 - \delta_i)$ and end-to-end capacity $C = 1 - \delta = 1 - (1 - \prod_{i=1}^{L}(1 - \delta_i)) = \prod_{i=1}^{L}(1 - \delta_i)$ in packets per channel use.

8.2.4 The BEC Channel with Feedback

It is quite common for many Internet applications, including multimedia ones, to support some form of feedback from the receiver to the transmitter. This could include feedback regarding requests for retransmissions of lost packets, a process that is commonly used in transport layer protocols such as TCP and in multimedia-specific variations of such protocols. For example, retransmissions of UDP/RTP packets carrying video or audio content are quite common over unicast Internet streaming sessions and are usually based on timely feedback from the receiver to the transmitter.

In this case, a crucial question is: what happens to the overall performance bounds of such channels with *feedback*? In particular, can we improve the maximum throughput or channel capacity C_{FB} by supporting feedback assuming that the feedback messages do not consume any of the capacity used in the forward

direction (i.e., between the transmitter and the receiver)? Information theory provides an interesting and arguably a surprising answer [1,2]. For any discrete memoryless channel (DMC), including BEC and PEC channels, feedback does not improve (or worsen for that matter) the throughput/capacity performance of the channel: $C = C_{FB}$.

Therefore, the results listed earlier for the basic (without feedback) BEC and PEC channels are also applicable for these channels with feedback. For the scenario of cascaded BEC/PEC channels, this is true if the feedback is implemented on an end-to-end basis. Here, end-to-end feedback means that only the transmitter and the receiver are involved in the feedback (i.e., the final receiver node in the chain is providing feedback to the very first transmitter node without the involvement of any of the intermediate nodes in the feedback process). Hence, for cascaded BEC/PEC channels with feedback on an end-to-end basis, we have

$$C_{FB} = C = 1 - \delta = 1 - \left(1 - \prod_{i=1}^{L}(1 - \delta_i)\right) = \prod_{i=1}^{L}(1 - \delta_i).$$

However, if the feedback is done on a link-by-link basis, then the overall channel capacity of a cascaded set of links is bounded by the capacity of the "bottleneck" link. In other words, the capacity in this case is

$$C_{FB} = C_{\min} = \min_i(C_i) = \min_i(1 - \delta_i).$$

It is important to note that the relationship $C = C_{FB}$ (in the case of end-to-end feedback) does not imply that a multimedia application should not use feedback on an end-to-end basis. On the contrary, feedback is crucial for the following reason. End-to-end feedback helps an application achieve (or at least get close to) the end-to-end capacity $C_{FB} = \prod_{i=1}^{L}(1 - \delta_i)$, which may not be achievable "in practice" without feedback. It is well known, for example, that multimedia streaming applications could benefit from employing feedback to recover lost packets through retransmission. In particular, consider a case when a multimedia application is streaming a multimedia content that is coded with a (source) rate R packets per second. Let's assume that $R < BC$, where B is the available bandwidth (in packets per second) and C is the effective (end-to-end) capacity (in packets per use on an end-to-end basis). Hence, the probability of a packet loss is $\delta = 1 - C$. Therefore, without any feedback and retransmission, the effective throughput that the application can achieve is $R - \delta R = (1 - \delta)R$ packets per second. Naturally, $R > (1 - \delta)R$. However, if the application employs feedback with retransmission, then the application can recover the lost packets (assuming delay is not an issue), and consequently it can achieve a throughput of R packets per second (i.e., streaming the multimedia source reliably). In other words, feedback with retransmission can help the application use access bandwidth that is

not being fully utilized by the application to achieve better reliability. Note that, in practice, even when the application (basic) source rate R is lower than the effective capacity, $R < BC$, packets will be lost, and therefore, retransmission can be very useful.

The aforementioned results for channel capacity with feedback are applicable to memoryless channels. Meanwhile, it has been well established that channels with memory could increase their capacity by employing feedback. The performance aspects of channels with memory are addressed next.

8.3 PACKET LOSSES OVER CHANNELS WITH MEMORY

Packet losses over Internet links and routes exhibit a high level of correlation and tend to occur in bursts. This observation, which has been well established by several Internet packet-loss studies, indicates that Internet links and routes exhibit memory. Consequently, although the DMC channel models discussed earlier could be useful for providing rough estimates of the loss behavior over the Internet, improved models are needed for more accurate estimates of the actual loss patterns. The most popular analysis and modeling tool used to capture memory is based on Markov chains. Channels that are modeled using Markov chains are sometimes referred to as Markov channels.

Bounds for the performance of channels with memory, including Markov channels, are significantly more difficult to derive and express as compared to DMC channels. In particular, performance bounds, such as capacity of channels with memory, do not have simple closed-form expressions as the case for DMC models. Recursive formulas for evaluating the channel capacity of general Markov channels have been developed though [4,5]. A special case of Markov channels is the Gilbert–Elliott channel, which consists of a two-state Markov chain. Recursive formulas for evaluating the channel capacity of the Gilbert–Elliott channel [4] and for (more general) finite-state Markov channels [5] have been developed.

This chapter focuses on the most basic (and probably most popular) Markov-based erasure channel model, which is the two-state Markov-state channel. This two-state Markov chain model of an erasure channel with memory is also known as the Gilbert model.

The Gilbert model of the two-state Markov chain is shown in Figure 8.3. Here, G and B represent the Good state and the Bad state, respectively. If the process (channel) is in the Good state, the transmitted packet is received without any errors; if the process is in the Bad state, the transmitted packets are lost (i.e., "erased"). At time zero, the system can start from the Good or the Bad state; this is known as the *initial state*. The system could also end in the Good or the Bad state.

This Gilbert channel is characterized by two parameters. A common parameter pair that is used for representing a Gilbert channel is the pair of transitional prob-

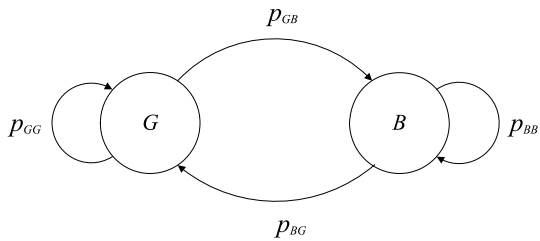


FIGURE 8.3: State diagram of the Gilbert model.

abilities: p_{GB} and p_{BG}. These probabilities are conditional probabilities with the following interpretations. p_{GB} is the probability that the channel transits to the bad state given that the channel is in the good state. Similarly, p_{BG} is the probability that the channel transits to the good state given that the channel is in the bad state. From these two conditional probabilities, one can measure the other transitional probabilities of staying in the same state: $p_{GG} = 1 - p_{GB}$ and $p_{BB} = 1 - p_{BG}$.

From the transitional probabilities p_{GB} and p_{BG}, one can express the overall ("average") probability $\pi(G)$ of being in the good state and the overall probability $\pi(B)$ of being in the bad state:

$$\pi(G) = \frac{p_{BG}}{p_{GB} + p_{BG}} \quad \text{and} \quad \pi(B) = \frac{p_{GB}}{p_{GB} + p_{BG}}.$$

Note that the probability $\pi(B)$ of being in a bad state provides the average loss probability of the two-state Markov channel. In other words, $\pi(B)$ plays the same role as the loss probability δ of the BEC channel. However, while the BEC channel could be completely characterized by a single parameter (i.e., δ), the Gilbert model needs two parameters as highlighted earlier. Also note that $\pi(G) + \pi(B) = 1$.

An important performance measure for multimedia applications is the number of packets received given that the transmitter sends n packets over routes with memory. This measure, for example, could help application developers identify the level of resilience needed when transmitting a block of n video packets; this block of n packets may correspond to the number of packets in a Group of Pictures (GoP) of an MPEG stream. Another example could arise when the n packets may represent a Forward-Error-Correction (FEC) block with both k media data (e.g., video) packets and $(n - k)$ parity packets that are used to recover lost packets within the n-packet FEC block. Although there is no closed-form solution for the channel capacity of Markov channels, it is possible to derive closed-form expressions for certain probability measures of losses over these channels. We present a closed-form expression for the probability of receiving an arbitrary number i of packets when the transmitter sends n packets over a two-state Markov channel.

Let $\phi(n,i)$ be the probability that the sender transmits n packets over a Gilbert channel and the receiver correctly receives i packets. It can be shown [6,7] that this probability can be expressed as

$$\phi(n,i) = \pi(G)\big(\phi_{G_0G_i}(n) + \phi_{G_0B_i}(n)\big) + \pi(B)\big(\phi_{B_0G_i}(n) + \phi_{B_0B_i}(n)\big),$$

where

$$\phi_{G_0G_i}(n) = \begin{cases} \displaystyle\sum_{m=1}^{i} \binom{i}{m}\binom{n-i-1}{m-1} p_{GB}^{m} p_{BG}^{m} p_{GG}^{i-m} p_{BB}^{n-i-m} & 0 < i < n, \\ 0 & i = 0, \\ p_{GG}^{n} & i = n, \end{cases}$$

$$\phi_{G_0B_i}(n) = \begin{cases} \displaystyle\sum_{m=0}^{i} \binom{i}{m}\binom{n-i-1}{m} p_{GB}^{m+1} p_{BG}^{m} p_{GG}^{i-m} p_{BB}^{n-i-m-1} & 0 \le i < n, \\ 0 & i = n, \end{cases}$$

$$\phi_{B_0G_i}(n) = \begin{cases} \displaystyle\sum_{m=0}^{i-1} \binom{i-1}{m}\binom{n-i}{m} p_{GB}^{m} p_{BG}^{m+1} p_{GG}^{i-m-1} p_{BB}^{n-i-m} & 0 < i \le n, \\ 0 & i = 0, \end{cases}$$

$$\phi_{B_0B_i}(n) = \begin{cases} \displaystyle\sum_{m=0}^{i-1} \binom{i-1}{m}\binom{n-i}{m+1} p_{GB}^{m+1} p_{BG}^{m+1} p_{GG}^{i-m-1} p_{BB}^{n-m-i-1} & 0 < i < n, \\ p_{BB}^{n} & i = 0, \\ 0 & i = n. \end{cases}$$

Here, $\phi_{G_0G_i}(n)$ is the probability that the sender transmits n packets and the receiver receives i packets given that the channel starts in a good state and ends in a good state. Similarly, interpretations can be inferred for $\phi_{G_0B_i}(n)$, $\phi_{B_0G_i}(n)$, and $\phi_{B_0B_i}(n)$. For example, $\phi_{G_0B_i}(n)$ is the probability that the sender transmits n packets and the receiver receives i packets given that the channel starts in a good state and ends in a bad state.

8.3.1 Packet Correlation over Channels with Memory

It is worth noting that the desired probability measure $\phi(n,i)$ can be completely evaluated using any two parameters that characterize the underlying Gilbert erasure channel. Traditionally, the transitional probabilities p_{GB} and p_{BG} (or $p_{GG} = 1 - p_{GB}$ and $p_{BB} = 1 - p_{BG}$) are used for such characterization.

A more useful insight and analysis can be gained by considering other parameter pairs. In particular, the average loss rate p and the *packet correlation* ρ can be used to represent the state transition probabilities, where $p_{GB} = p(1-\rho)$ and $p_{BG} = (1-p)(1-\rho)$.

The steady-state probabilities are directly related to the average loss rate p: $\pi(G) = 1-p$ and $\pi(B) = p$. The packet erasure correlation ρ provides an average measure of how the states of two consecutive packets are correlated to each other. In particular, when $\rho = 0$, the loss process is memoryless and the aforementioned probability measures reduce to the special case of a memoryless BEC. However, as the value of ρ increases, then the states of two consecutive packets become more and more correlated. Hence, we find that the parameters p and ρ provide an intuitive, insightful, and broad characterization for the impact of channel coding on networks with losses.

Figure 8.4 plots the probability that a receiver correctly receives i packets when the source send n packets over the Gilbert channel. Here, n is set to 30, the average loss rate p is set to 1%, and the packet correlation ρ is changed from 0 to 0.9. As compared with the Binomial model (where $\rho = 0$), we can see that as ρ increases, the probability of receiving a smaller number of packets increases. For a given ρ, as i increases, $\phi(n,i)$ increases exponentially; this increase slows down as ρ increases. When $\rho = 0.9$, we can see that $\phi(n,i)$ has a small spike at $i = 0$ and a big spike at $i = 30$. This observation is consistent with the analytical intuition; as the correlation is strong, once the process initially starts in a bad or a good state, it has the inertia to stay at that state. For $p = 0.01$ and $\rho = 0.9$, the transition probabilities are $p_{GG} = 0.999$, $p_{GB} = 0.001$, $p_{BG} = 0.099$, and $p_{BB} = 0.901$, respectively.

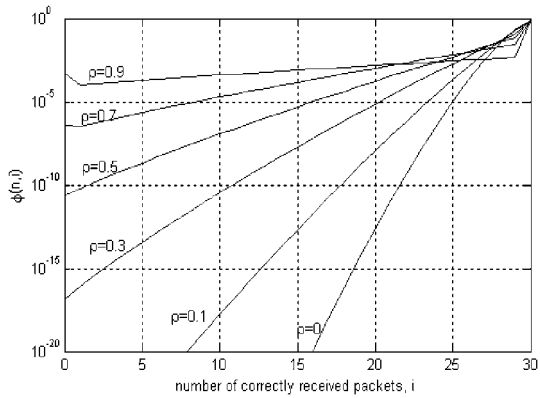


FIGURE 8.4: Probability of receiving i packets given that the transmitter sends $n = 30$ packets, for loss probability $p = 0.1$.

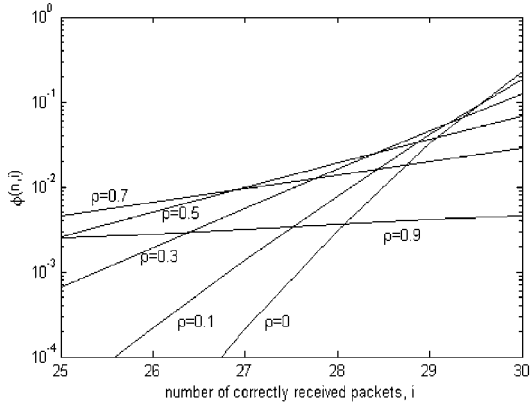


FIGURE 8.5: A detailed view of the probability of receiving i packets given that the transmitter sends n packets.

Figure 8.5 is a detailed view when the received packets i changes from 25 to 30. Figure 8.5 shows that as the packet correlation increases, the probability of receiving a higher number of packets decreases.

8.3.2 Packet Losses over Cascaded Channels with Memory

As highlighted earlier, Internet routes and paths consist of multiple links that can be modeled as cascaded channels. Further, each of these links/channels usually exhibits memory. Hence, the case of cascaded channels with memory represents an important scenario for modeling the performance of Internet end-to-end paths. In this section, we extend the results of the aforementioned section while making the simplifying assumption that the cascaded links are independent of each other.

Let $\phi_j(n_j, i_j)$ be the probability of receiving i_j packets while transmitting n_j packets over a channel j with memory (e.g., a Markov channel). First, let's assume that we have only two channels that are cascaded with each other, and hence $j = 1, 2$. We are interested in measuring the probability $\phi(n, i)$ of receiving i packets at the output of these cascaded channels (i.e., the output of the second channel with index $j = 2$) when the transmitter sends n packets into the input of the first channel (i.e., the input of the first channel with index $j = 1$). Based on the notation adopted earlier, we have $n = n_1$, $i_1 = n_2$, and $i_2 = i$. Note that receiving $i_2 = i$ packets at the output of the second channel (which is the output of the overall two-cascaded channels) implies that the number of packets $n_2 = i_1$ transmitted into the input to the second channel, $j = 2$, must be at least $i_2 = i$; in other words, $n_2 = i_1 \geq i_2 = i$. Further, since $n \geq i_1$, $(n = n_1) \geq (n_2 = i_1) \geq (i_2 = i)$ (Figure 8.6).

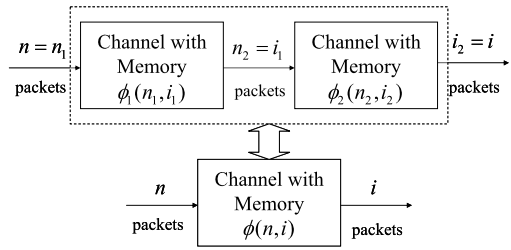


FIGURE 8.6: A representation of the reception of i packets when transmitting n packets over two-cascaded channels with memory.

Hence, the desired probability $\phi(n, i)$ of receiving i packets at the output of these two cascaded channels when the transmitter sends n packets into the input of the first channel can be expressed as

$$\phi(n_1, i_2) = \sum_{i_1(=n_2)=i}^{n} \phi_1(n_1, i_1)\phi_2(n_2, i_2).$$

In other words,

$$\phi(n, i) = \sum_{i_1=i}^{n} \phi_1(n, i_1)\phi_2(i_1, i).$$

Hence, if the cascaded channels with memory are both Gilbert channels, then the desired probability of receiving i packets at the output of the second channel given that n packets are transmitted at the input of the first channel can be expressed as

$$\begin{aligned}\phi(n, i) = \sum_{i_1=i}^{n} \big(&\pi(G_1)\big(\phi_{1,G_0G_{i_1}}(n) + \phi_{1,G_0B_{i_1}}(n)\big) \\ &+ \pi(B_1)\big(\phi_{1,B_0G_{i_1}}(n) + \phi_{1,B_0B_{i_1}}(n)\big)\big) \\ &\times \big(\pi(G_2)\big(\phi_{2,G_0G_i}(i_1) + \phi_{2,G_0B_i}(i_1)\big) \\ &+ \pi(B_2)\big(\phi_{2,B_0G_i}(i_1) + \phi_{2,B_0B_i}(i_1)\big)\big).\end{aligned}$$

Here, $\phi_{j,G_0G_i}(n)$ $(\phi_{j,B_0B_i}(n))$ is the probability that the transmitter sends n packets and the receiver receives i packets over the jth channel given that the channel begins and ends in a good (bad) state. Also, $\pi(G_j)$ $(\pi(B_j))$ is the probability that the jth channel is in a good (bad) state.

The aforementioned expressions for $\phi(n,i)$ over two-cascaded channels can be generalized to N channels with memory. In particular, one can infer the following probability of receiving i packets at the output of N cascaded channels with memory given that the transmitter sends n packets:

$$\phi(n,i) = \sum_{i_1=i}^{n} \sum_{i_2=i_1}^{n} \cdots \sum_{i_{N-1}=i_{N-2}}^{n} \phi_1(n,i_1)\phi_2(i_1,i_2)\cdots\phi_{N-1}(i_{N-2},i_{N-1}) \times \phi_N(i_{N-1},i).$$

A more compact representation of this probability is

$$\phi(n,i) = \sum_{i_1=i}^{n} \sum_{i_2=i_1}^{n} \cdots \sum_{i_{N-1}=i_{N-2}}^{n} \phi_1(n,i_1)\left(\prod_{j=2}^{N-1} \phi_j(i_{j-1},i_j)\right)\phi_N(i_{N-1},i).$$

8.4 WIDE-SCALE INTERNET STREAMING STUDY

8.4.1 Overview

The Internet is a complex interconnection of computer networks whose behavior and structure are usually challenging to measure. Numerous studies have attempted to shed light on the performance of the Internet; however, they traditionally examined backbone and campus-network characteristics and paid little attention to the conditions experienced by average home users during their daily activities. Among several traditional approaches, the Internet has been studied from the perspective of TCP connections by Paxson [27], Bolliger et al. [10], Caceres *et al.* [16], Mogul [25], and several others (e.g., [9]). Paxson's study included 35 geographically distributed sites in nine countries; Bolliger and colleagues employed 11 sites in seven countries and compared the throughput performance of various implementations of TCP during a 6-month experiment; whereas the majority of other researchers monitored transit TCP traffic at a single backbone router [8, 25] or inside several campus networks [16] for the duration ranging from several hours to several days. The methodology used in both large-scale TCP experiments [10,27] was similar and involved a topology where each participating site was paired with every other participating site for an FTP-like transfer. Although this setup approximates well the current use of TCP in the Internet, future entertainment-oriented streaming services, however, are more likely to involve a small number of backbone video servers and a large number of home users.

We should further mention that the Internet has been studied extensively by various researchers using ICMP `ping` and `traceroute` packets [8,17–19,26,27], UDP echo packets [11,14,15], and multicast backbone (MBone) audio packets

[35,36]. With the exception of the last one, similar observations apply to these studies—neither the setup nor the type of probe traffic represented realistic real-time streaming scenarios. Among the studies that specifically sent audio/video traffic over the Internet [12,13,20,21,32–34], the majority of experiments involved only a few Internet paths, lasted for a short period of time, and focused on analyzing the features of the proposed scheme rather than the impact of Internet conditions on real-time streaming.

In this work, we argue that studying network conditions observed by *regular users* is an important research topic and take a fundamentally different measurement approach that looks at Internet dynamics from the angle of Internet users rather than network operators. In our experiments, video streaming clients connect to the Internet through several dial-up ISPs in the United States and emulate the behavior of an average end user in the late 1990s and early 2000s.[1] In addition to choosing a different topological setup for the experiment, our work is different from the previous studies in the following three aspects. First, recall that the sending rate of a TCP connection is driven by its congestion control, which can sometimes cause increased packet loss and higher end-to-end delays in the path along which it operates (e.g., during slow start or after timeouts). In our experiment, we aimed to measure the *true* end-to-end path dynamics without the bias of congestion control applied to slow modem links. Our decision not to use congestion control was additionally influenced by the evidence that the majority of streaming traffic in the current Internet employs constant bit rate (CBR) video streams [30], where users explicitly select the desired streaming rate from content providers' Web pages. Second, TCP uses a positive ACK retransmission scheme, whereas current real-time applications (such as [30]) employ NACK-based retransmission to reduce the amount of traffic from users to streaming servers. As a consequence, end-to-end path dynamics perceived by a NACK-based protocol could differ from those sampled by TCP along the same path: real-time applications acquire samples of the round-trip delay (RTT) at rare intervals, send significantly less data along the path from the receiver to the sender, and bypass certain aspects of TCP's retransmission scheme (such as exponential timer backoff). Finally, TCP relies on window-based flow control, whereas real-time applications usually utilize rate-based flow control. In many video coding schemes, a real-time streaming server must maintain a certain target streaming rate for the decoder to avoid *underflow events*, which are caused by packets arriving after their decoding deadlines. As a result, a real-time sender may operate at different levels of packet

[1] Market research reports (e.g., [22,23,28]) show that in Q2 of 2001 approximately 89% of Internet-enabled U.S. households used dial-up access to connect to the Internet. As of March 2006, 34% of polled Americans used dial-up, many of whom had no plans or desire to switch to broadband [37]. Furthermore, countries with less developed network infrastructure are expected to experience dial-up-like (including high-latency satellite and cellular) Internet access for the foreseeable future.

burstiness and instantaneous sending rate than a TCP sender, as the sending rate of a TCP connection is governed by the arrival of positive ACKs from the receiver rather than by the application.

In what follows in the rest of this chapter, we present the methodology and analyze the results of a 7-month, large-scale, real-time streaming experiment that involved three nationwide dial-up ISPs, each with several million active subscribers in the United States. The topology of the experiment consisted of a backbone video server streaming MPEG-4 video sequences to unicast home users located in more than 600 major U.S. cities. The streaming was performed in real time (i.e., with a real-time decoder), utilized UDP for the transport of all messages, and relied on simple NACK-based retransmission to attempt recovery of lost packets before their decoding deadlines.

8.4.2 Methodology

8.4.2.1 Setup for the Experiment

We started our work by attaching a Unix video server to the UUNET backbone via a T1 link as shown in Figure 8.7. To support the clients' connectivity to the Internet, we selected three major nationwide dial-up ISPs: AT&T WorldNet, Earthlink, and IBM Global Network (which we call ISP_a, ISP_b, and ISP_c, respectively), each with at least 500 V.90 (i.e., 56 kb/s) dial-up numbers in the United States. Our experiment emulated the activity of hypothetical Internet users who dialed local access numbers to reach the Internet and streamed video sequences from a backbone server. Although the clients were physically located in our laboratory in the state of New York, they dialed long-distance phone numbers and connected to the Internet through ISPs' access points in each of the 50 states. Our database of phone numbers included 1813 different V.90 access numbers in 1188 major U.S. cities.

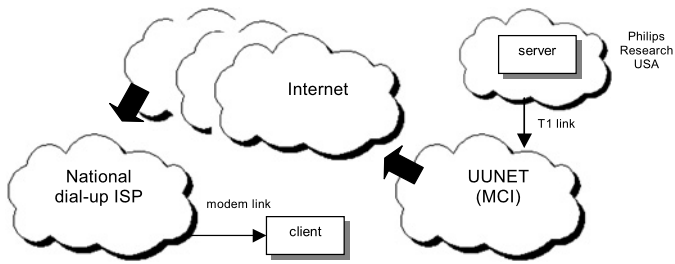


FIGURE 8.7: Setup of the experiment.

After the phone database was in place, we designed and implemented special software, which we call the *dialer*, that dialed phone numbers from the database, connected to the ISPs using the point-to-point protocol (PPP), issued a *parallel traceroute* to the server, and, upon success, started the video client with the instructions to stream a 10-min video sequence from the server. Our implementation of traceroute (built into the dialer) used ICMP instead of the more traditional UDP, sent all probes in parallel instead of sequentially (hence the name "parallel"), and recorded the IP time-to-live (TTL) field of each returned "TTL-expired" message. The use of ICMP packets and parallel traceroute facilitated much quicker discovery of routers, and the analysis of the TTL field in the returned packets allowed the dialer to compute the number of hops in the reverse path from each intermediate router to the client machine using a simple fact that each router reset the TTL field of each generated "TTL-expired" packet to some default value. The majority of routers used the default TTL equal to 255, while some initialized the field to 30, 64, or 128. Subtracting the received TTL from the default TTL produced the number of hops along the reverse path. Using the information about the number of *forward* and *reverse* hops for each router, the dialer was able to detect asymmetric end-to-end paths, which is studied in Section 8.4.8.

In our analysis of data, we attempted to isolate clearly modem-related pathologies (such as packet loss caused by a poor connection over the modem link and large RTTs due to data-link retransmission) from those caused by congested routers of the Internet. Thus, connections that were unable to complete a traceroute to the server, those with high bit-error rates (BER), and those during which the modem could not sustain our streaming rates were all considered useless for our study and were excluded from the analysis in this section. In particular, we utilized the following methodology. We defined a streaming attempt through a given access number to be *successful* if the access point of the ISP was able to sustain the transmission of our video stream for its entire length at the stream's target IP bit rate r. Success was declared if the video client finished streaming while the *aggregate* (i.e., counting from the very beginning of a session) packet loss at all times t was below a certain threshold β_p and the *aggregate* incoming bit rate was above another threshold β_r. The experiments reported in this section used β_p equal to 15% and β_r equal to $0.9r$, whose combination was experimentally found to quite effectively filter out modem-related failures. The packet-loss threshold was activated after 1 min of streaming and the bit rate threshold after 2 min to make sure that slight fluctuations in packet loss and incoming bit rate at the beginning of a session were not mistaken for poor connection quality. After a session was over, the success or failure of the session was communicated from the video client to the dialer, the latter of which kept track of the time of day and the phone number that either passed or failed the streaming test.

In order to make the experiment reasonably short, we considered all phone numbers from the same state to be equivalent; consequently, we assumed that

a successful streaming attempt through any phone number of a state indicated a successful coverage of the state regardless of which phone number was used. Furthermore, we divided each 7-day week into 56 three-hour timeslots (i.e., 8 time slots per day) and designed the dialer to select phone numbers from the database such that each state would be successfully covered within each of the 56 time slots at least once. In other words, each ISP needed to sustain exactly $50 \times 56 = 2800$ successful sessions before the experiment was allowed to end.

8.4.2.2 Real-Time Streaming

For the purpose of the experiment, we used an MPEG-4 encoder to create two 10-min QCIF (176×144) video streams coded at five frames per second (fps). The first stream, which we call S_1, was coded at the video bit rate of 14 kb/s, and the second steam, which we call S_2, was coded at 25 kb/s. The experiment with stream S_1 lasted during November–December 1999 and the one with stream S_2 was an immediate follow-up during January–May 2000.

During the transmission of each video stream, the server split it into 576-byte IP packets. Video frames always started on a packet boundary; consequently, the last packet in each frame was allowed to be smaller than others (in fact, many P [prediction-coded] frames were smaller than the maximum payload size and were carried in a single UDP packet). As a consequence of packetization overhead, the *IP bit rates* (i.e., including IP, UDP, and our special 8-byte headers) for streams S_1 and S_2 were 16.0 and 27.4 kb/s, respectively. The statistics of each stream are summarized in Table 8.1.

In our streaming experiment, the term *real time* refers to the fact that the video decoder was running in real time. Recall that each compressed video frame has a specific *decoding deadline*, which is usually based on the time of the frame's encoding. If a compressed video frame is not fully received by the decoder buffer at the time of its deadline, the video frame is discarded and an underflow event is registered. Moreover, to simplify the analysis of the results, we implemented a *strict* real-time decoder model, in which the playback of the arriving frames continued at the encoder-specified deadlines regardless of the number of underflow events (i.e., the decoding deadlines were not adjusted based on network conditions). Note that in practice, better results can be achieved by allowing the decoder to freeze the display and rebuffer a certain number of frames when underflow events become frequent (e.g., as done in [30]).

Table 8.1: Summary of streams statistics.

Stream	Size, MB	Packets	Video bit rate, kb/s	Average frame size, bytes
S_1	1.05	4188	14.0	350
S_2	1.87	5016	25.0	623

In addition, many CBR video coding schemes include the notion of *ideal start-up delay* [29,30] (the delay is called "ideal" because it assumes a network with no packet loss and a constant end-to-end delay). This ideal delay must always be applied to the decoder buffer before the decoding process may begin. The ideal start-up delay is independent of the network conditions and solely depends on the decisions made by the encoder during the encoding process.[2] On top of this ideal start-up delay, the client in a streaming session must usually apply an *additional* start-up delay in order to compensate for delay jitter (i.e., variation in the one-way delay) and permit the recovery of lost packets via retransmission. This additional start-up delay is called the *delay budget* (D_{budget}) and reflects the values of the *expected* delay jitter and round-trip delay during the length of the session. Note that in the context of Internet streaming, it is common to call D_{budget} simply "start-up delay" and to completely ignore the ideal start-up delay (e.g., [21]). From this point on, we will use the same convention. In all our experiments, we used D_{budget} equal to 2700 ms, which was manually selected based on preliminary testing. Consequently, the total start-up delay (observed by an end user) at the beginning of each session was approximately 4 s.

8.4.2.3 Client–Server Architecture

For the purpose of our experiment, we implemented a client–server architecture for MPEG-4 streaming over the Internet. The server was fully multithreaded to ensure that the transmission of packetized video was performed at the target IP bit rate of each streaming session and to provide a quick response to clients' NACK requests. The streaming was implemented in bursts of packets (with the burst duration D_b varying between 340 and 500 ms depending on the bit rate) for the purposes of making the server as low overhead as possible (e.g., RealAudio servers have been reported to use $D_b = 1800$ ms [24]).

The second and the more involved part of our architecture, the client, was designed to recover lost packets through NACK-based retransmission and to collect extensive statistics about each received packet and each decoded frame. Furthermore, as it is often done in NACK-based protocols, the client was in charge of collecting round-trip delay samples. The measurement of RTTs involved the following two methods. In the first method, each successfully recovered packet provided a sample of the RTT, which was the duration between sending a NACK and receiving the corresponding retransmission. In our experiment, in order to avoid the ambiguity of which retransmission of the same packet actually returned to the client, the header of each NACK request and each retransmitted packet contained an extra field specifying the retransmission sequence number of the packet.

[2] We will not elaborate further on the ideal start-up delay, except to mention that it was approximately 1300 ms for each stream.

The second method of measuring the RTT was used by the client to obtain *additional* samples of the round-trip delay in cases where network packet loss was too low. The method involved periodically sending *simulated* retransmission requests to the server if packet loss was below a certain threshold. In response to these simulated NACKs, the server included the usual overhead of fetching the needed packets from the storage and sending them to the client.[3] In our experiment, the client activated simulated NACKs, spaced 30 seconds apart, if packet loss was below 1%.

We tested the software and the concept of a wide-scale experiment of this sort for 9 months before we felt comfortable with the setup, the reliability of the software, and the exhaustiveness of the collected statistics. In addition to extensive testing of the prototype, we monitored various statistics reported by the clients in real time (i.e., on the screen) during the experiments for sanity and consistency with previous tests. Overall, the work reported in this section took us 16 months to complete.

Our traces consist of six datasets, each collected by a different machine. Throughout this section, we use notation D_n^x to refer to the dataset collected by the client assigned to ISP_x $(x = a, b, c)$ during the experiment with stream S_n $(n = 1, 2)$. Furthermore, we use notation D_n to refer to the combined set $\{D_n^a \cup D_n^b \cup D_n^c\}$.

8.4.3 Overview of Experimental Results

In dataset D_1, the three clients performed 16,783 long-distance connections to the ISPs' remote modems and successfully completed 8429 streaming sessions. Typical reasons for failing a session were PPP-layer connection problems, inability to reach the server (i.e., failed traceroute), high bit-error rates, and low (14.4–19.2 kb/s) modem connection rates. In D_2, the clients performed 17,465 connections and sustained 8423 successful sessions. In dataset D_1, the clients traced the arrival of 37.7 million packets, and in D_2, the arrival of an additional 47.3 million (for a total of 85 million). In terms of bytes, the first experiment transported 9.4 GB of video data and the second one transported another 17.7 GB (for a total of 27.1 GB).

Recall that each experiment lasted as long as it was needed to cover the entire United States. Depending on the success rate within each state, the access points used in the experiment comprised a subset of our database. In D_1, the experiment covered 962 dial-up points in 637 U.S. cities, and in D_2, it covered 880 dial-up points in 575 U.S. cities. Figure 8.8 shows the per-state distribution of the number

[3] Server overhead was below 10 ms for all retransmitted packets and did not have a major impact on our characterization of the RTT process later in this section.

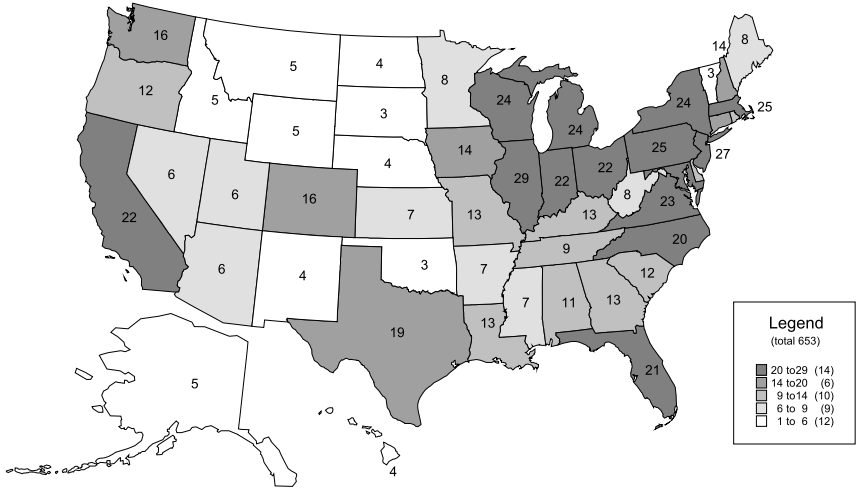


FIGURE 8.8: The number of unique cities per state that participated in both experiments (i.e., in $\{D_1 \cup D_2\}$).

of distinct cities in each state covered by both experiments, which represents 1003 access points in 653 cities.

Analysis of the success rates observed during the experiment suggests that in order to receive real-time streaming material at 16 to 27.4 kb/s, an average U.S. end user equipped with a V.90 modem needs to make approximately two dialing attempts to his/her local ISPs. The success rate of streaming sessions during the different times of the day is illustrated in Figure 8.9 (top). Note the dip by a factor of two between the best (i.e., 12–3 a.m.) and the worst (i.e., 9 p.m.–12 a.m.) times of the day.

During this measurement study, each session was preceded by a parallel traceroute that recorded the IP addresses of all discovered routers (DNS and WHOIS lookups were done off-line after the experiments were over). The average time needed to trace an end-to-end path was 1731 ms, with 90% of the paths traced under 2.5 s and 98% under 5 s. Dataset D_1 recorded 3822 distinct Internet routers, D_2 recorded 4449 distinct routers, and both experiments combined produced the IP addresses of 5266 unique router interfaces. The majority of the discovered routers belonged to the ISPs' networks (51%) and UUNET (45%), which confirmed our intuition that all three ISPs had direct peering connections with UUNET. Interestingly, the traces showed approximately 200 routers that belonged to five additional Autonomous Systems (AS), indicating that certain end-to-end paths were routed across additional ISPs.

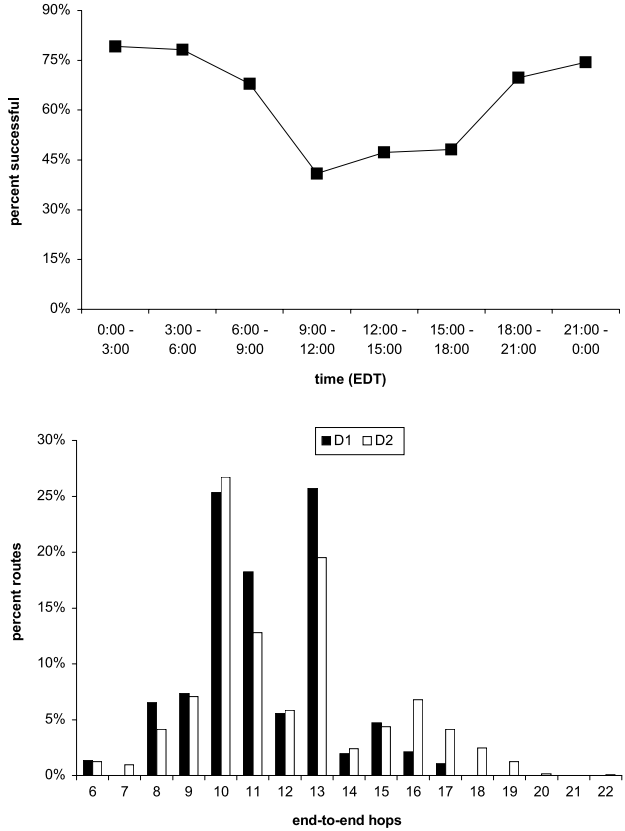


FIGURE 8.9: Success of streaming attempts during the day (top). Distribution of the number of end-to-end hops (bottom).

The average end-to-end hop count was 11.3 in D_1 (6 minimum and 17 maximum) and 11.9 in D_2 (6 minimum and 22 maximum). Figure 8.9 (bottom) shows the distribution of the number of hops in the encountered end-to-end paths in each of D_1 and D_2. As Figure 8.9 shows, the majority of paths (75% in D_1 and 65% in D_2) contained between 10 and 13 hops.

Throughout the rest of the section, we restrict ourselves to studying only *successful* (as defined earlier) sessions in both D_1 and D_2. We call these new *purged* datasets with only successful sessions D_{1p} and D_{2p}, respectively (purged datasets D_{np}^{x} are defined similarly for $n = 1, 2$ and $x = a, b, c$). Recall that $\{D_{1p} \cup D_{2p}\}$ contains 16,852 successful sessions, which are responsible for 90% of the bytes and packets, 73% of the routers, and 74% of the U.S. cities recorded in $\{D_1 \cup D_2\}$.

8.4.4 Packet Loss

8.4.4.1 Overview

Numerous studies have focused on Internet packet loss; however, due to the enormous diversity of the Internet, only a few of them agree on the average packet loss rate or the average loss-burst length (i.e., the number of packets lost in a row). Among prior conclusions, the average Internet packet loss was reported to vary between 11 and 23% by Bolot [11] depending on the inter-packet transmission spacing, between 0.36 and 3.54% by Borella *et al*. [14,15] depending on the studied path, between 1.38 and 11% by Yajnik *et al*. [36] depending on the location of the MBone receiver, and between 2.7 and 5.2% by Paxson [27] depending on the year of the experiment. In addition, 0.49% average packet loss rate was reported by Balakrishnan *et al*. [9], who analyzed the dynamics of a large number of TCP Web sessions at a busy Internet server.

In dataset D_{1p}, the average recorded packet loss rate was 0.53% and in D_{2p}, it was 0.58%. Even though these rates are much lower than those traditionally reported by Internet researchers during the last decade due to the much lower transmission rates used in our study, they are still much higher than those advertised by backbone ISPs (i.e., 0.01–0.1%). We thus speculate that the majority of loss occurred at the "edges" of the Internet rather than at its core. Approximately 38% of the sessions in $\{D_{1p} \cup D_{2p}\}$ did not experience any packet loss, 75% experienced loss rates below 0.3%, and 91% experienced loss rates below 2%. However, 2% of the sessions suffered packet loss rates 6% or higher.

As expected, average packet loss rates exhibited a wide variation during the day. Figure 8.10 (top) shows the evolution of loss rates as a function of the time slot (i.e., the time of day), where each point represents the average of approximately 1000 sessions. As Figure 8.10 shows, the variation in loss rates between the best (3–6 a.m.) and the worst (3–6 p.m.) times of the day was by a factor of two in D_{1p} and by a factor of three in D_{2p}. The apparent discontinuity between time slots 7 (21:00–0:00) and 0 (0:00–3:00) is due to a coarse timescale in Figure 8.10 (top). On finer timescales (e.g., minutes), loss rates converge to a common value near midnight. A similar discontinuity in packet loss rates was reported by Paxson [27] for North American sites, where packet loss during time slot 7 was approximately twice as high as that during time slot 0.

The average *per-state* packet loss shown in Figure 8.10 (bottom) varied quite substantially from 0.2% in Idaho to 1.4% in Oklahoma, but virtually did not depend on the state's average number of end-to-end hops to the server (correlation coefficient ρ was -0.04) or the state's average RTT (correlation -0.16). However, as discussed later, the average per-state RTT and the number of end-to-end hops were, in fact, positively correlated.

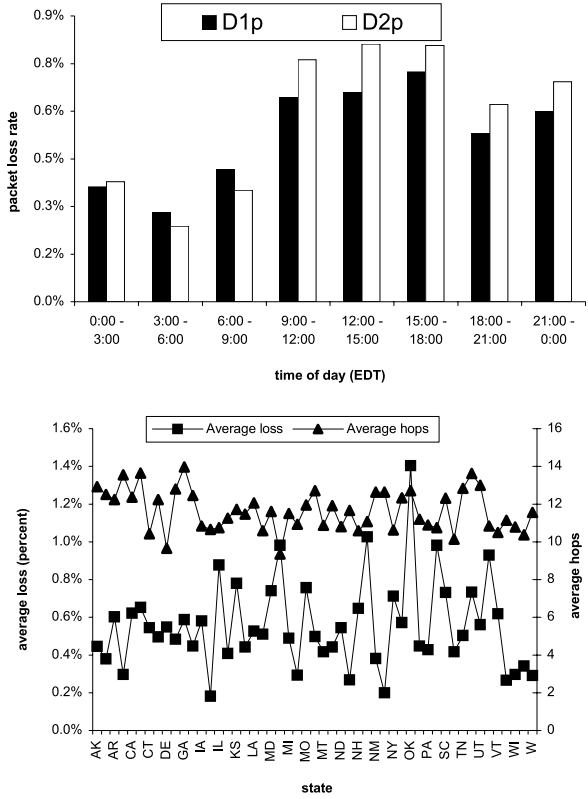


FIGURE 8.10: Average packet loss rates during the day (top). Average per-state packet loss rates (bottom).

8.4.4.2 Loss Burst Lengths

We next attempt to answer the question of how bursty Internet packet loss was during the experiment. Figure 8.11 (top) shows the distribution (both the histogram and the CDF) of loss-burst lengths in $\{D_{1p} \cup D_{2p}\}$. Note that Figure 8.11 stops at burst length 20, which covers more than 99% of the bursts. Even though the upper tail of the distribution had very few samples, it was fairly long and reached burst lengths of over 100 packets.

Figure 8.11 (top) is based on 207,384 loss bursts and 431,501 lost packets. The prevalence of single-packet losses, given the fact that the traffic in our experiment was injected into the Internet in bursts at the T1 speed, leads to one possibility that router queues sampled in our experiment predominantly overflowed

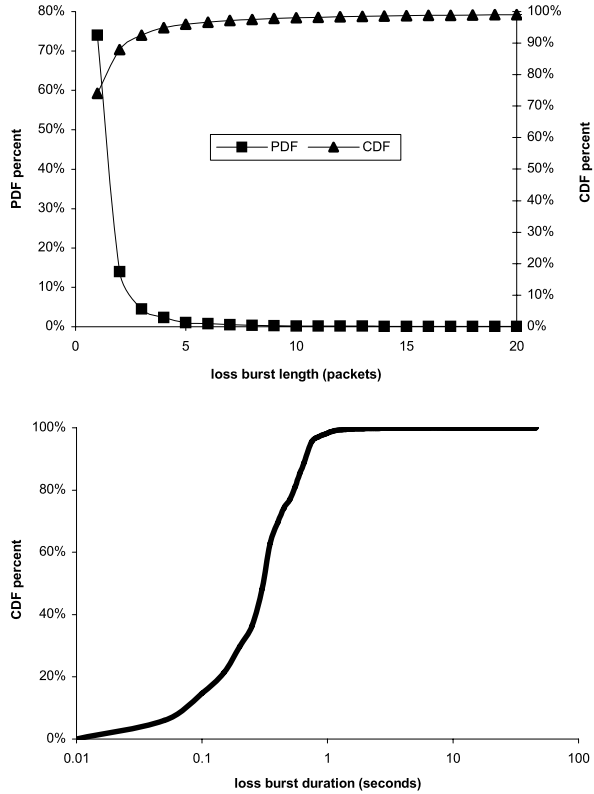


FIGURE 8.11: Histogram (PDF) and CDF functions of loss-burst lengths in $\{D_{1p} \cup D_{2p}\}$ (top). The CDF function of loss-burst durations in $\{D_{1p} \cup D_{2p}\}$ (bottom).

on timescales smaller than the time needed to transmit a single IP packet over a T1 link (i.e., 3 ms for the largest packets and 1.3 ms for the average-size packets). However, interference of cross-traffic between video packets at prebottleneck routers (i.e., which causes expansion of interpacket dispersion) or usage of RED makes it much more difficult to accurately assess the duration of loss events inside routers. To investigate the presence of RED in the Internet, we contacted several backbone and dial-up ISPs whose routers were recorded in our trace and asked them to comment on the deployment of RED in their backbones. Among the ISPs that responded to our request, the majority had purposely disabled RED and the others were running RED only for select customers at border routers, but not on the public backbone. Ruling out RED, another difficulty of computing the

duration of congestion-related loss at routers is the fact that single-packet losses were underrepresented in our traces, as packets that were lost in bursts longer than one packet could have been dropped by *different* routers along the path from the server to the client. Therefore, using end-to-end measurements, an application cannot distinguish between n ($n \geq 2$) single-packet losses at n different routers from an n-packet bursty loss at a single router. Both types of loss events appear identical to an end-to-end application even though the underlying cause is quite different. Consequently, we conclude that even though the analysis of our datasets points toward transient (i.e., 1–3 ms) buffer overflows in the Internet routers sampled by our experiment, a more detailed study is needed to verify this finding and sample packet-loss durations at individual routers.

As previously pointed out by many researchers, the upper tail of loss-burst lengths usually contains a substantial percentage of all lost packets. In each of D_{1p} and D_{2p}, single-packet bursts contained only 36% of all lost packets, bursts 2 packets or shorter contained 49%, bursts 10 packets or shorter contained 68%, and bursts 30 packets or shorter contained 82%. At the same time, 13% of all lost packets were dropped in bursts at least 50 packets long.

Traditionally, the burstiness of packet loss is measured by the average loss-burst length. In dataset D_{1p}, the average burst length was 2.04 packets. In dataset D_{2p}, the average burst length was slightly higher (2.10), but not high enough to conclude that the higher bit rate of stream S_2 was clearly responsible for burstier packet loss. Furthermore, the *conditional probability* of packet loss, given that the previous packet was also lost, was 51% in D_{1p} and 53% in D_{2p}. These numbers are consistent with those previously reported in the literature. Bolot [11] observed the conditional probability of packet loss to range from 18 to 60% depending on interpacket spacing during transmission, Borella *et al.* [15] from 10 to 35% depending on the time of day, and Paxson [27] reported 50% conditional probability for *loaded* (i.e., queued behind the previous) TCP packets and 25% for *unloaded* packets. Using Paxson's terminology, the majority of our packets were *loaded* since the server sent packets in bursts at a rate higher than the bottleneck link's capacity.

8.4.4.3 Loss Burst Durations

To a large degree, the average loss-burst length depends on how closely the packets are spaced during transmission. Assuming that bursty packet loss comes from buffer overflows in drop-tail queues rather than from consecutive hits by RED or from bit-level corruption, it is clear that all packets of a flow passing through an overflown router queue will be dropped for the duration of the instantaneous congestion. Hence, the closer together the flow's packets arrive to the router, the more packets will be dropped during each queue overflow. This fact was clearly demonstrated in Bolot's [11] experiments, where UDP packets spaced 8 ms apart

suffered larger loss-burst lengths (mean 2.5 packets) than packets spaced 500 ms apart (mean 1.1 packets). Yajnik *et al.* [36] reported a similar correlation between loss-burst lengths and the distance between packets. Consequently, instead of analyzing burst lengths, one might consider measuring burst durations in time units since the latter does not depend on interpacket spacing during transmission.

Using our traces, we can only infer an approximate duration of each loss burst because we do not know the *exact* time when the lost packets were supposed to arrive to the client. Hence, for each loss event, we define the *loss-burst duration* as the time elapsed between the receipt of the packet immediately preceding the loss burst and the packet immediately following it. Figure 8.11 (bottom) shows the distribution (CDF) of loss-burst durations in seconds. Although the distribution tail is quite long (up to 36 s), the majority (more than 98%) of loss-burst durations in both datasets D_{1p} and D_{2p} fall under 1 s. We speculate that some of this effect was caused by data-link retransmission on the modem link, which may also be responsible for large delays in modern wireless and satellite networks. Paxson's [27] study similarly observed large loss-burst durations (up to 50 s); however, only 60% of loss bursts studied by Paxson were contained below 1 s. In addition, our traces showed that the average distance between lost packets in the experiment was 172–188 good packets, or 21–27 s, depending on the streaming rate.

8.4.4.4 Heavy Tails

In conclusion of this section, it is important to note that packet losses sometimes cannot be modeled as independent events due to buffer overflows that last long enough to affect multiple adjacent packets. Consequently, future real-time protocols should expect to deal with bursty packet losses (Figure 8.11) and possibly heavy-tailed distributions of loss-burst lengths (see later). Several researchers reported a heavy-tailed nature of loss-burst lengths with shape parameter α of the Pareto distribution fitted to the length (or duration) of loss bursts ranging from 1.06 [8] to 2.75 [15]. However, Yajnik *et al.* [36] partitioned the collected data into empirically chosen stationary segments and reported that loss-burst lengths could be modeled as exponential (i.e., not heavy-tailed) within each stationary segment.

Using intuition, it is clear that packet loss and RTT random processes in both D_{1p} and D_{2p} are expected to be nonstationary. For example, the nonstationarity can be attributed to the time of day or the location of the client. In either case, we see three approaches to modeling such nonstationary data. In the first approach, one would analyze 16,852 CDF functions (one for each session) for stationarity and heavy tails. Unfortunately, an average session contained only 24 loss bursts, which was insufficient to build a good distribution function for statistical analysis.

The second approach would be to combine all sessions into groups that are intuitively perceived to be stationary (e.g., according to the access point or the time slot) and then perform similar tests for stationarity and heavy tails within each group. We might consider this direction for future work. The third approach is to assume that all data samples belong to some stationary process and are drawn from a single distribution, which is commonly performed by researchers for simplicity of analysis. Using the last approach, Figure 8.12 (top) shows a log–log plot of the complementary CDF function from Figure 8.11 (top) with a least-squares

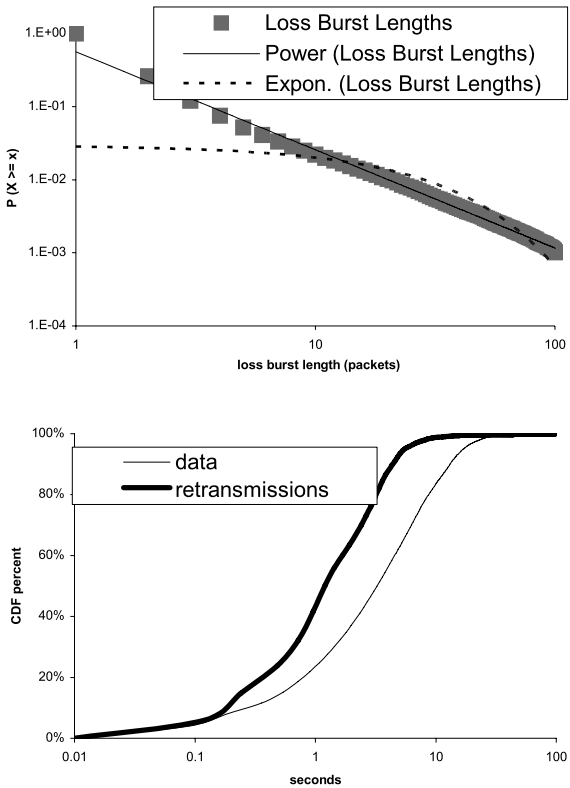


FIGURE 8.12: The complementary CDF of loss-burst lengths in $\{D_{1p} \cup D_{2p}\}$ on a log–log scale fitted with hyperbolic (straight line) and exponential (dotted curve) distributions (top). CDF functions of the amount of time by which retransmitted and data packets were late for decoding (bottom).

fit of a straight line representing a heavy-tailed distribution (the dotted curve is the exponential distribution fitted to data). The fit of a straight line is quite good (with correlation $\rho = 0.99$) and provides a strong indication that the distribution of loss-burst lengths in the combined dataset $\{D_{1p} \cup D_{2p}\}$ is heavy tailed. However, the exponential distribution in Figure 8.12 (top) decays too quickly to even remotely fit the data.

Finally, consider a Pareto distribution with CDF $F(x) = 1 - (\beta/x)^{\alpha}$ and PDF $f(x) = \alpha\beta^{\alpha}x^{-\alpha-1}$, where α is the shape parameter and β is the location parameter. Using Figure 8.12 (top), we establish that a Pareto distribution with $\alpha = 1.34$ (finite mean, but infinite variance) and $\beta = 0.65$ fits our data very well.

8.4.5 Underflow Events

The impact of packet losses on real-time applications is understood fairly well. Each lost packet that is not recovered before its deadline causes an underflow event. In addition to packet loss, real-time applications suffer from large end-to-end delays. However, not all types of delay are equally important to real-time applications. As shown later, one-way delay jitter was responsible for 90 times more underflow events in our experiment than packet loss combined with large RTTs.

Delays are important for two reasons. First, large round-trip delays make retransmissions late for their decoding deadlines. However, the RTT is important only to the extent of recovering lost packets and, in the worst case, can cause only *lost* packets to be late for decoding. However, delay jitter (i.e., one-way delay variation) can potentially cause each *data* (i.e., nonretransmitted) packet to be late for decoding. In $\{D_{1p} \cup D_{2p}\}$, packet loss affected 431,501 packets, out of which 159,713 (37%) were discovered to be missing *after* their decoding deadlines had passed. As a result, NACKs were not sent for these packets. Out of 271,788 remaining lost packets, 257,065 (94.6%) were recovered before their deadlines, 9013 (3.3%) arrived late, and 5710 (2.1%) were never recovered. The fact that more than 94% of "recoverable" lost packets were actually received before their deadlines indicates that retransmission is a very effective method of overcoming packet loss in real-time applications. Clearly, the success rate will be even higher in networks with smaller RTTs or applications with larger start-up delays. In fact, these results can be used to properly select D_{budget} for applications operating in similar network conditions to ensure the desired level of lost-packet recovery.

Before studying underflow events caused by delay jitter, we introduce two types of late retransmissions. The first type consists of packets that arrived after the decoding deadline of the last frame of the corresponding GoP. These packets were *completely* useless and were discarded. The second type of late packets, which we call *partially late*, consists of those packets that missed their *own* decoding

deadline, but arrived before the deadline of the last frame of the same GOP. Since the video decoder in our experiment could decompress frames at a substantially higher rate than the target fps, the client was able to use partially late packets for motion-compensated reconstruction of the remaining frames from the same GOP before *their* corresponding decoding deadlines. Out of 9013 late retransmissions, 4042 (49%) were partially late. Using each partially late packet, the client was able to save on average 4.98 frames from the same 10-frame GOP in D_{1p} and 4.89 frames in D_{2p} by employing the catch-up technique described earlier (for more discussion, see [31]).

In contrast to 174,436 underflows caused by packet loss, one-way delay jitter was responsible for 1,167,979 underflows in *data* (i.e., nonretransmitted) packets. Hence, the total number of packets missing at the time of decoding was $174{,}436 + 1{,}167{,}979 = 1{,}342{,}415$ (1.7% of the number of sent packets), which means that 87% of underflow packets were produced by large one-way delay jitter rather than by packet loss. Even if the clients had not attempted to recover any of the 431,501 lost packets, 73% of the missing packets at the time of decoding would have been caused by large delay jitter. In terms of user-perceived metrics, 1.3 million underflow packets caused a freeze-frame effect on average for 10.5 s per 10-min session in D_{1p} and 8.6 s in D_{2p}, which can be considered excellent given the small amount of delay budget used in the experiments.

To further understand the phenomenon of late packets, Figure 8.12 (bottom) plots the CDFs of the amount of time by which late packets missed their deadlines (i.e., the amount of time that was needed to add to delay budget $D_{\text{budget}} = 2700$ ms in order to avoid a certain percentage of underflow events). As Figure 8.12 shows, 25% of late retransmissions missed their deadlines by more than 2.6 s, 10% by more than 5 s, and 1% by more than 10 s (the tail of the CDF extends up to 98 s). At the same time, one-way delay jitter had a more adverse impact on data packets: 25% of late data packets missed their deadlines by more than 7 s, 10% by more than 13 s, and 1% by more than 27 s (the CDF tail extends up to 56 s).

One common way of reducing the number of late packets caused by large RTTs and delay jitter is to apply a higher start-up delay D_{budget} at the beginning of a session. An additional approach is to freeze the display and effectively increase the start-up delay during the session as need arises. The final approach works for streaming prerecorded content and allows the receiver to request that the server transmit video traffic at a faster-than-ideal bit rate at certain times to compensate for delayed packets and to increase the amount of buffered frames at the decoder (available bandwidth permitting). Using a strict no-freeze model of this work, only the first approach was viable, which would require a 13-s total delay budget to save 99% of late retransmissions and 84% of late data packets under similar streaming conditions.

8.4.6 Round-Trip Delay

8.4.6.1 Overview

We should mention that circuit-switched long-distance links through PSTN between our clients and remote access points did not significantly influence the measured end-to-end delays because the additional delay on each long-distance link was essentially the propagation delay between New York and the location of the access point. Since the propagation delay is determined by the speed of light and geographic distance, most links experienced bias of no more than approximately 32 ms, which is the round-trip delay of a 3000-mile link. Clearly, this delay is negligible compared to the queuing and transmission delays experienced by our packets along the entire end-to-end path. Figure 8.13 (top) shows

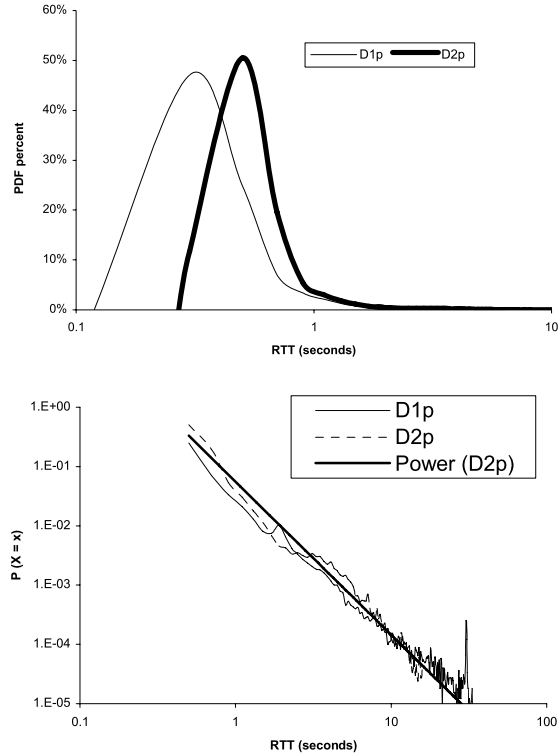


FIGURE 8.13: Histograms (PDF) of RTT samples in each of D_{1p} and D_{2p} (top). Log–log plot of the upper tails of the RTT histogram. The straight line is fitted to D_{2p} (bottom).

the histogram of round-trip delays in each of D_{1p} and D_{2p} (660,439 RTT samples in both datasets). Although the tail of the combined distribution reached the enormous values of 126 s for *simulated* and 102 s for *real* retransmissions, the majority (75%) of the samples were below 600 ms, 90% below 1 s, and 99.5% below 10 s. The average RTT was 698 ms in D_{1p} and 839 ms in D_{2p}. The minimum RTT was 119 and 172 ms, respectively. Although very rare, extremely high RTTs were found in all six datasets $D^a_{1p} - D^c_{2p}$. Out of more than 660,000 RTT samples in $\{D_{1p} \cup D_{2p}\}$, 437 were at least 30 s, 32 at least 50 s, and 20 at least 75 s.

Although pathologically high RTTs may seem puzzling at first, there is a simple explanation. Modem error correction protocols (i.e., the commonly used V.42) implement retransmission for corrupted blocks of data at the physical or data-link layer.[4] Error correction is often necessary if modems negotiated data compression (i.e., V.42bis) over the link and is desirable if the PPP Compression Control Protocol is enabled on the data-link layer. In all our experiments, both types of compression were enabled, imitating the typical setup of a home user. Therefore, if a client established a connection to a remote modem at a low bit rate (which was sometimes accompanied by a significant amount of noise in the phone line), each retransmission at the physical layer took a long time to complete before data were delivered to the upper layers. In addition, large IP-level buffers on either side of the modem link further delayed packets arriving to or originating from the client host.

Note that the purpose of classifying sessions into failed and successful in Section 8.4.2.1 was to avoid reporting pathological conditions caused by modem links. Since less than 0.5% of RTTs in $\{D_{1p} \cup D_{2p}\}$ were seriously affected by modem-level retransmission and bit errors (i.e., had RTTs higher than 10 s), we conclude that our heuristic was successful in filtering out the majority of pathological connections and that future application-layer protocols running over a modem link should be prepared to experience RTTs on the order of several seconds.

8.4.6.2 Heavy Tails

Mukherjee [26] reported that RTTs along certain Internet paths could be modeled by a shifted gamma distribution. Even though the shape of the PDF in Figure 8.13 (top) resembles that of a gamma function, the distribution tails in Figure 8.13 decay much slower than those of an exponential distribution (see later). Using our approach from Section 8.4.4.4 (i.e., assuming that each studied Internet random

[4] Since the telephone network beyond the local loop in the United States is mostly digital, we believe that dialing long-distance numbers had no significant effect on the number of bit errors during the experiment.

process is stationary), we extracted the upper tails of the PDF functions in Figure 8.13 (top) and plotted the results on a log–log scale in Figure 8.13 (bottom). Figure 8.13 shows that a straight line (without loss of generality fitted to the PDF of D_{2p} in the figure) provides a good fit to data (correlation 0.96) and allows us to model the upper tails of both PDF functions as a Pareto distribution with PDF $f(x) = \alpha\beta^{\alpha}x^{-\alpha-1}$, where shape parameter α equals 1.16 in dataset D_{1p} and 1.58 in D_{2p} (as before, the distribution has a finite mean, but an infinite variance).

8.4.6.3 Variation of the RTT

We conclude the discussion of the RTT by showing that it exhibited a variation during the day similar to that of packet loss shown previously in Figure 8.10 (top) and that the average RTT was correlated positively with the length of the corresponding end-to-end path. Figure 8.14 (top) shows the average round-trip

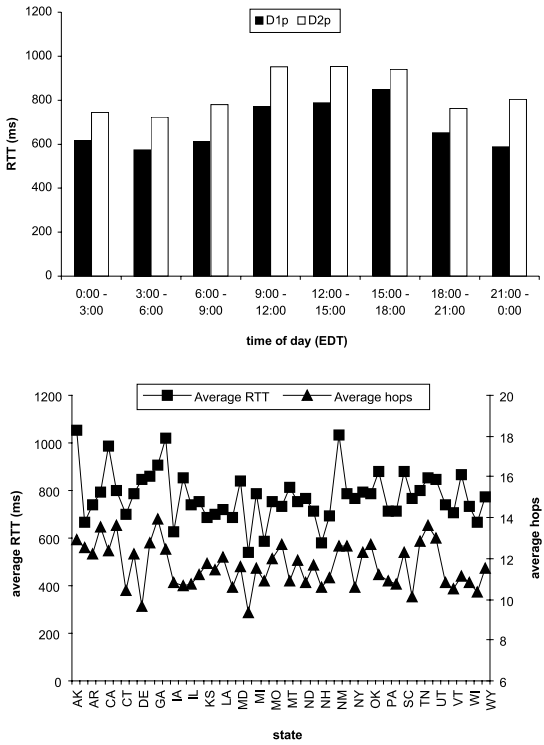


FIGURE 8.14: Average RTT as a function of the time of day (top). Average RTT and average hop count in each of the states in $\{D_{1p} \cup D_{2p}\}$ (bottom).

delay during each of the eight time slots of the day (as before, each point in Figure 8.14 represents the average of approximately 1000 sessions). Figure 8.14 confirms that the worst time for sending traffic over the Internet is between 9 a.m. and 6 p.m. EDT and shows that the increase in the delay during peak hours is relatively small (i.e., by 30–40%).

Figure 8.14 (bottom) shows the average RTT sampled by the clients in each of the 50 U.S. states. The average round-trip delay was consistently high (i.e., above 1 s) for three states: Alaska, New Mexico, and Hawaii. However, the RTT was consistently low (below 600 ms) also for three states: Maine, New Hampshire, and Minnesota. These results, except Minnesota, can be directly correlated with the distance from New York; however, in general, we found that the geographical distance of the access point from the East Coast had little effect on the average RTT. For example, certain states in the Midwest had small (600–800 ms) average round-trip delays and certain states on the East Coast had large (800–1000 ms) average RTTs. A more substantial link can be established between the number of end-to-end hops and the average RTT, as shown in Figure 8.14 (bottom). Even though the average RTT of many states did not exhibit a clear dependency on the average length of the path, the correlation between the RTT and the number of hops in Figure 8.14 (bottom) was reasonably high with $\rho = 0.52$. This result was intuitively expected, as the RTT is essentially the sum of queuing and transmission delays at intermediate routers.

8.4.7 Delay Jitter

As discussed earlier, in certain streaming situations round-trip delays are much less important to real-time applications than one-way delay jitter because the latter can potentially cause significantly more underflow events. In addition, due to asymmetric path conditions (i.e., uneven congestion in the upstream and downstream directions), large RTTs are not necessarily an indication of bad network conditions for a NACK-based application. In many sessions with high RTTs during the experiment, the outage was caused by the upstream path, while the downstream path did not suffer from extreme one-way delay variation and data packets were arriving to the client throughout the entire duration of the outage. Hence, we conclude that the RTT is not necessarily a good indicator of a session's quality during streaming and that one-way delay jitter should be used instead.

Assuming that delay jitter is defined as the difference between one-way delays of each two consecutively sent packets, an application can sample both positive and negative values of delay jitter. Negative values are produced by two types of packets—those that suffered a *packet compression event* (i.e., the packets' arrival spacing was smaller than their transmission spacing) and those that became reordered. The former case is of great interest in packet-pair bandwidth estimation studies and otherwise remains relatively unimportant. The latter case is studied

in Section 8.4.8 under packet reordering. However, positive values of delay jitter represent *packet expansion events*, which are responsible for late packets. Consequently, we analyzed the distribution of only *positive* delay jitter samples and found that although the highest sample was 45 s, 97.5% of the samples were less than 140 ms and 99.9% under 1 s. As the aforementioned results show, large values of delay jitter were not frequent, but once a packet was significantly delayed by the network, a substantial number of the following packets were delayed as well, creating a "snowball" of underflows. This fact explains the large number of underflow events reported in previous sections, even though the overall delay jitter was relatively low.

8.4.8 Packet Reordering

8.4.8.1 Overview

Real-time protocols often rely on the assumption that packet reordering in the Internet is a rare and insignificant event for all practical purposes (e.g., [21]). Although this assumption simplifies the design of a protocol, it also makes the protocol poorly suited for use over the Internet. Certainly, there are Internet paths along which reordering is either nonexistent or extremely rare. At the same time, there are paths that are dominated by multipath routing effects and often experience reordering (e.g., Paxson [27] reported a session with 36% of packets arriving out of order).

Unfortunately, there is not much data documenting reordering rates experienced by IP traffic over modem links. Using intuition, we expected reordering in our experiments to be extremely rare given the low bit rates of streams S_1 and S_2. However, we were surprised to find out that certain paths experienced consistent reordering with a relatively large number of packets arriving out of order, although the average reordering rates in our experiments were substantially lower than those reported by Paxson [27].

For example, in dataset D^a_{1p}, we observed that out of every three missing[5] packets one was reordered. Hence, if users of ISP_a employed a streaming protocol that used a gap-based detection of lost packets [21] (i.e., the first out-of-order packet triggered an NACK), 33% of NACKs would be redundant and a large number of retransmissions would be unnecessary, causing a noticeable fraction of ISP's bandwidth to be wasted.

Since each missing packet is potentially reordered, the true frequency of reordering can be captured by computing the percentage of reordered packets relative to the total number of *missing* packets. The average reordering rate in our experiment was 6.5% of the number of *missing* packets, or 0.04% of the number

[5]Missing packets are defined as gaps in sequence numbers.

of *sent* packets. These numbers show that our reordering rates were at least by a factor of 10 lower than those reported by Paxson [27], whose average reordering rates varied between 0.3 and 2% of sent packets depending on the dataset. This difference can be explained by the fact that our experiment was conducted at substantially lower end-to-end bit rates, as well as by the fact that Paxson's experiment involved several paths with extremely high reordering rates.

Out of 16,852 sessions in $\{D_{1p} \cup D_{2p}\}$, 1599 (9.5%) experienced at least one reordering. Interestingly, the average session reordering rates in our datasets were very close to those in Paxson's 1995 data [27] (12% sessions with at least one reordering), despite the fundamental differences in sending rates. The highest reordering rate *per ISP* in our experiment occurred in D^a_{1p}, where 35% of missing packets (0.2% of sent packets) turned out to be reordered. In the same D^a_{1p}, almost half of the sessions (47%) experienced at least one reordering event. Furthermore, the maximum number of reordered packets in a single session occurred in D^b_{1p} and was 315 packets (7.5% of sent packets).

Interestingly, the reordering probability did not show any dependence on the time of day (i.e., the time slot) and was virtually the same for all states.

8.4.8.2 Reordering Delay

To further study packet reordering, we define two metrics that allow us to measure the extent of packet reordering. First, let *packet reordering delay* D_r be the delay from the time when a reordered packet was declared as *missing* to the time when the reordered packet arrived to the client. Second, let *packet reordering distance* d_r be the number of packets (including the very first out-of-sequence packet, but not the reordered packet itself) received by the client during reordering delay D_r. These definitions are illustrated in Figure 8.15, where reordering distance d_r is two packets and reordering delay D_r is the delay between receiving packets 3 and 2.

Figure 8.16 (top) shows the histogram of D_r in $\{D_{1p} \cup D_{2p}\}$. The largest reordering distance d_r in the combined dataset was 10 packets, and the largest reordering delay D_r was 20 s (however, in the latter case, d_r was only 1 packet). Although quite large, the maximum value of D_r is consistent with previously reported numbers (e.g., 12 s in Paxson's data [27]). The majority (90%) of samples

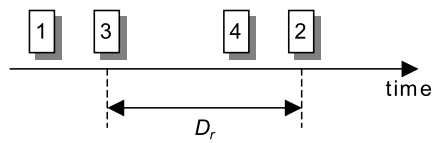


FIGURE 8.15: The meaning of reordering delay D_r.

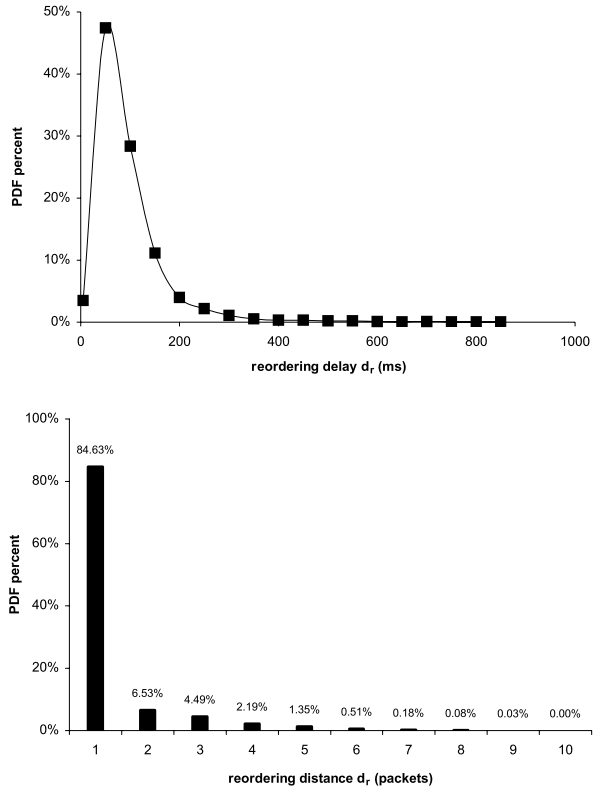


FIGURE 8.16: Histogram of reordering delay D_r in $\{D_{1p} \cup D_{2p}\}$ (top). Histogram of reordering distance d_r in $\{D_{1p} \cup D_{2p}\}$ (bottom).

in Figure 8.16 (top) were below 150 ms, 97% below 300 ms, and 99% below 500 ms.

8.4.8.3 Reordering Distance

We next analyze the suitability of TCP's triple-ACK scheme in helping NACK-based protocols detect reordering. TCP's *fast retransmit* relies on *three* consecutive duplicate ACKs (hence the name "triple-ACK") from the receiver to detect packet loss and avoid unnecessary retransmissions. Therefore, if reordering distance d_r is either 1 or 2, the triple-ACK scheme successfully avoids duplicate packets; if d_r is greater than or equal to 3, it generates a duplicate packet. Figure 8.16 (bottom) shows the PDF of reordering distance d_r in both datasets.

Using Figure 8.16, we can infer that TCP's triple-ACK would be successful for 91.1% of the reordering events in our experiment, double-ACK for 84.6%, and quadruple-ACK for 95.7%. Note that Paxson's TCP-based data [27] show similar, but slightly better, detection rates, specifically 95.5% for triple-ACK, 86.5% for double-ACK, and 98.2% for quadruple-ACK.

8.4.9 Asymmetric Paths

Recall that during the initial executions of the traceroute, our dialer recorded the TTL field of each received "TTL-expired" packet. These fields allowed the dialer to compute the number of hops between the router that generated a particular "TTL-expired" message and the client. Suppose some router i was found at hop f_i in the *upstream* (i.e., forward) direction and at hop r_i in the *downstream* (i.e., reverse) direction. Hence, we can conclusively establish that an n-hop end-to-end path is *asymmetric* if a router exists for which the number of downstream hops is different from the number of upstream hops (i.e., $\exists i,\ 1 \leq i \leq n\colon\ f_i \neq r_i$). However, the opposite is not always true—if each router has the same number of downstream and upstream hops, we cannot conclude that the path is symmetric (i.e., it could be asymmetric as well). Hence, we call such paths *possibly symmetric*.

In $\{D_{1p} \cup D_{2p}\}$, 72% of the sessions sent their packets over *definitely* asymmetric paths. In that regard, two questions prompt for an answer. First, does path asymmetry depend on the number of end-to-end hops? To answer this question, we extracted path information from $\{D_{1p} \cup D_{2p}\}$ and counted each end-to-end path through a particular access point exactly once. Figure 8.17 shows the percentage of asymmetric paths as a function of the number of end-to-end hops in the path. As Figure 8.17 shows, almost all paths with 14 hops or more were asymmetric, as well as that even the shortest paths (with only 6 hops) were prone to asymmetry. This result can be explained by the fact that longer paths are more likely to cross over AS boundaries or intra-AS administrative domains. In both cases, "hot-potato" routing policies may cause path asymmetry.

The second question we attempt to answer is whether path asymmetry has anything to do with reordering. In $\{D_{1p} \cup D_{2p}\}$, 95% of all sessions with at least one reordered packet were running over an *asymmetric* path. Consequently, we can conclude that if a session in our datasets experiences a reordering event along a path, then the path is most likely asymmetric. However, a new question that arises is whether the opposite is true as well: if a path is asymmetric, will a session be more likely to experience a reordering? To answer the last question, we have the following numbers. Out of 12,057 sessions running over a definitely asymmetric path, 1522 experienced a reordering event, which translates into 12.6% reordering rate. However, out of 4795 sessions running over a possibly symmetric path, only 77 (1.6%) experienced a reordering event. Hence, an asymmetric

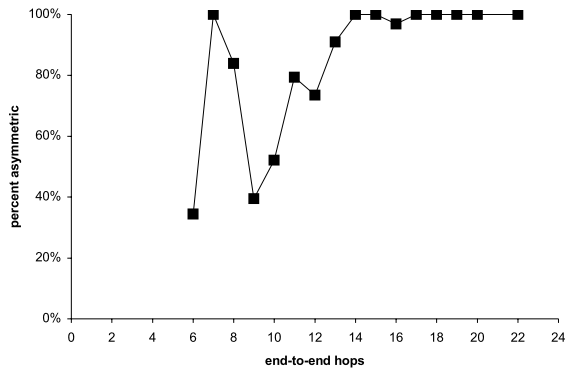


FIGURE 8.17: Percentage of asymmetric routes in $\{D_{1p} \cup D_{2p}\}$ as a function of the number of end-to-end hops.

path is eight times more likely to experience a reordering event than a possibly symmetric path.

Even though there is a clear link between reordering and asymmetry in our datasets, we speculate that the two could be related through the length of each end-to-end path. In other words, longer paths are more likely to experience reordering as well as be asymmetric. Hence, rather than saying that reordering causes asymmetry or vice versa, we can explain the result by noting that longer paths are more likely to cross AS-level routing boundaries during which both "hot-potato" routing (which causes asymmetry) and IP-level load balancing (which causes reordering) are apparently quite frequent.

Clearly, the findings in this section depend on the particular ISP employed by the end user and the autonomous systems that user traffic traverses. Large ISPs (such as the ones studied in this work) often employ numerous peering points (hundreds in our case), and path asymmetry rates found in this section may not hold for smaller ISPs. Nevertheless, our data allow us to conclude that home users in the United States experience asymmetric end-to-end paths with a much higher frequency than symmetric ones.

8.5 SUMMARY AND FURTHER READING

In this chapter, introductory information theory concepts that are related to characterizing packet losses over the Internet were presented. The most basic channel model that can be used to capture packet losses over network routes is the Binary Erasure Channel or its simple extension the Packet Erasure Channel. For a given probability δ of symbol loss (bit loss for a BEC and a packet loss for a

PEC), these channels have a capacity of $C = 1 - \delta$ in symbols (bits or packets) per channel use. Extensions of this basic and fundamental result to cascaded links and to routes with feedback were outlined. For excellent treatment of information theory concepts, the reader is referred to [1–3].

All of the results related to the BEC/PEC channels and their extensions are based on the assumption that the losses are memoryless. Meanwhile, deriving the capacity of channels with memory is beyond the scope of this introductory material. Methods for measuring information theory parameters, such as mutual information and capacity, for channels with memory can be found in [5]. An important measure for Internet multimedia applications that employ some form of channel coding is the probability of recovering a desired number of message (or data) packets when transmitting an FEC block of n packets [that include both k message and $(n - k)$ parity packets]. We outlined a set of closed form solutions of such measure for a basic channel with memory, namely the two-state Markov channel (i.e., the Gilbert channel). Some details for deriving these closed form solutions can be found elsewhere [6,7].

The second part of this chapter described a comprehensive Internet video study conducted for gaining insight into a variety of end-to-end performance parameters crucial for real-time multimedia applications. These performance parameters included packet loss, loss-burst lengths and durations, roundtrip delay, delay jitter, packet reordering and related delays due to reordering, and video underflow events. A great deal of work and research efforts has collected very valuable data regarding Internet performance. Some of these efforts focused on real-time and multimedia applications, including studies that specifically sent audio/video traffic over the Internet, such as the ones reported in [12,13,20,21,32–34]. The majority of these studies, however, involved only a few Internet paths, lasted for a short period of time, and focused on analyzing the features of the proposed scheme rather than the impact of Internet conditions on real-time streaming. However, the Internet video study reported in this chapter is quite comprehensive and covered a broad range of performance parameters that are important for Internet multimedia applications.

BIBLIOGRAPHY

[1] T. M. Cover and J. A. Thomas. *Elements of Information Theory*, Wiley, 1991.

[2] R. W. Yeung. *A First Course in Information Theory*, Kluwer Academic/Plenum Publishers, 2002.

[3] R. G. Gallager. *Information Theory and Reliable Communication*, Wiley, 1968.

[4] M. Mushkin and I. Bar-David. "Capacity and coding for the Gilbert–Elliot channels," *IEEE Trans. Inform. Theory*, vol. IT-35, no. 6, pp. 1277–1290, November 1989.

[5] A. J. Goldsmith and P. P. Varaiya. "Capacity, mutual information, and coding for finite-state Markov channels," *IEEE Trans. Inform. Theory*, vol. IT-42, no. 3, pp. 868–886, May 1996.

[6] M. Wu and H. Radha. "Network-Embedded FEC (NEF) Performance for Multi-Hop Wireless Channels with Memory," IEEE International Conference on Communications (ICC), May 2005.

[7] M. Wu and H. Radha. "Network Embedded FEC for Overlay and P2P Multicast over Channels with Memory," Conference on Information Sciences and Systems (CISS), Johns Hopkins University, March 2005.

[8] A. Acharya and J. Saltz. "A Study of Internet Round-trip Delay," *Technical Report CS-TR-3736*, University of Maryland, December 1996.

[9] H. Balakrishnan, V. M. Padmanablah, S. Seshan, M. Stemm, and R. H. Katz. "TCP Behavior of a Busy Internet Server: Analysis and Improvements," *IEEE INFOCOM*, March 1998.

[10] J. Bolliger, T. Gross, and U. Hengartner. "Bandwidth Modeling for Network-Aware Applications," *IEEE INFOCOM*, 1999.

[11] J. Bolot. "Characterizing End-to-End Packet Delay and Loss in the Internet," *Journal of High Speed Networks*, vol. 2, no. 3, pp. 289–298, September 1993.

[12] J. Bolot and T. Turletti. "A Rate Control Mechanism for Packet Video in the Internet," *IEEE INFOCOM*, pp. 1216–1223, June 1994.

[13] J. Bolot and T. Turletti. "Experience with Rate Control Mechanisms for Packet Video in the Internet," *ACM Computer Communication Review*, pp. 4–15, January 1998.

[14] M. S. Borella, D. Swider, S. Uludag, and G. B. Brewster. "Internet Packet Loss: Measurement and Implications for End-to-End QoS," *International Conference on Parallel Processing*, August 1998.

[15] M. S. Borella, S. Uludag, G. B. Brewster, and I. Sidhu. "Self-similarity of Internet Packet Delay," *IEEE ICC*, August 1997.

[16] R. Caceres, P. B. Danzig, S. Jamin, and D. Mitzel. "Characteristics of Wide-Area TCP/IP Conversations," *ACM SIGCOMM*, 1991.

[17] K. C. Claffy, G. C. Polyzos, and H.-W. Braun. "Measurement Considerations for Assessing Unidirectional Latencies," *Internetworking: Research and Experience*, vol. 4, pp. 121–132, 1993.

[18] K. C. Claffy, G. C. Polyzos, and H.-W. Braun. "Application of Sampling Methodologies to Network Traffic Characterization," *ACM SIGCOMM*, 1993.

[19] K. C. Claffy, G. C. Polyzos, and H.-W. Braun. "Traffic Characteristics of the T1 NSFNET Backbone," *IEEE INFOCOM*, 1993.

[20] B. Dempsey, J. Liebeherr, and A. Weaver. "A New Error Control Scheme for Packetized Voice over High-Speed Local Area Networks," *18th IEEE Local Computer Networks Conference*, September 1993.

[21] B. Dempsey, J. Liebeherr, and A. Weaver. "On retransmission-based error control for continuous media traffic in packet-switching networks," *Computer Networks and ISDN Systems*, vol. 28, no. 5, pp. 719–736, March 1996.

[22] ISP Planet and Cahners In-Stat Group. "Dial-Up Remains ISPs' Bread and Butter," http://isp-planet.com/research/2001/dialup_butter.html, 2001.

[23] ISP Planet and Telecommunications Research International. "U.S. Residential Internet Market Grows in Second Quarter," http://isp-planet.com/research/2001/us_q2.html, 2001.

[24] A. Mena and J. Heidemann. "An Empirical Study of Real Audio Traffic," *IEEE INFOCOM*, March 2000.

[25] J. C. Mogul. "Observing TCP Dynamics in Real Networks," *ACM SIGCOMM*, 1992.
[26] A. Mukherjee. "On the Dynamics and Significance of Low Frequency Components of Internet Load," *Internetworking: Research and Experience*, vol. 5, pp. 163–205, 1994.
[27] V. Paxson. "Measurements and Analysis of End-to-End Internet Dynamics," *Ph.D. dissertation*, Computer Science Department, University of California at Berkeley, 1997.
[28] S. P. Pizzo. "Why Is Broadband So Narrow?" *Forbes ASAP*, p. 50, September 2001.
[29] H. Radha, Y. Chen, K. Parthasarathy, and R. Cohen. "Scalable Internet Video Using MPEG-4," *Signal Processing: Image Communications Journal*, 1999.
[30] Real Player G2. Real Networks, http://www.real.com.
[31] I. Rhee. "Error Control Techniques for Interactive Low Bitrate Video Transmission over the Internet," *ACM SIGCOMM*, September 1998.
[32] S. Servetto and K. Nahrstedt. "Broadcast Quality Video over IP," *IEEE Transactions on Multimedia*, vol. 3, no. 1, March 2001.
[33] W. Tan and A. Zakhor. "Real-Time Internet Video Using Error Resilient Scalable Compression and TCP-Friendly Transport Protocol," *IEEE Transactions on Multimedia*, vol. 1, no. 2, June 1999.
[34] T. Turletti and G. Huitema. "Videoconferencing on the Internet," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, June 1996.
[35] M. Yajnik, J. Kurose, and D. Towsley. "Packet Loss Correlation in the MBone Multicast Network," *IEEE GLOBECOM*, November 1996.
[36] M. Yajnik, S. Moon, J. Kurose, and D. Townsley. "Measurement and Modelling of the Temporal Dependence in Packet Loss," *IEEE INFOCOM*, 1999.
[37] Pew Internet & American Life Project. "Home Broadband Adoption 2006," http://www.pewinternet.org/PPF/r/184/report_display.asp, May 2006.

9

Forward Error Control for Packet Loss and Corruption

Raouf Hamzaoui, Vladimir Stanković, and Zixiang Xiong

9.1 INTRODUCTION

Many techniques have been proposed to protect media data against channel errors. One possible approach is error-resilient source coding, which includes packetization of the information bit stream into independently decodable packets, exploitation of synchronization markers to control error propagation, reversible variable-length coding, and multiple description coding. Another approach is based on error concealment, where the lost or corrupted data is estimated at the receiver side with, for example, interpolation. Error control for media data may also exploit error detection and retransmission (ARQ, see Chapter 7). One further approach is forward error correction (FEC) with error correcting codes. Finally, one may combine any of the aforementioned methods. The choice of an appropriate error control method is not easy because it requires a deep understanding of both the source and the channel. In this respect, many important questions have to be answered: What is the type of the data? Is the data compressed? If yes, what is the compression scheme used? Is the data being transmitted over a wireline or a wireless network? Is there a feedback channel? What are the channel conditions? Moreover, the user requirements must also be taken into consideration. What is more important: reconstruction fidelity or transmission speed?

In this chapter, we present error control systems that rely on forward error correction only. While ARQ techniques have traditionally been the error control method of choice, there are many situations in which they are not suitable. For example, ARQ is not possible when there is no feedback channel. Also, in some

applications, such as video multicasting or broadcasting, ARQ can overwhelm the sender with retransmission requests.

This chapter focuses on error control systems that were designed for embedded (or scalable) video bit streams (e.g., bit streams produced by MPEG-4 FGS, H.264/AVC MCTF, or some of the three-dimensional wavelet-based video coders described in Chapter 5). An overview of error control techniques for nonscalable video coders can be found in [33], in [32], and in Chapter 2. We describe several error protection systems and discuss their rate–distortion performance. When possible, we also provide efficient algorithms for optimizing this performance by adequately allocating the total transmission bit budget between the source coder and the channel coder.

This chapter is organized as follows. In Section 9.2, we present a class of transmission systems that are particularly well suited to the packet erasure channel, which as explained in Chapter 8, is a good model for the Internet. In Section 9.3, we describe a transmission system that was successfully used for bit error protection over a memoryless channel. Finally, Section 9.4 handles the more difficult case of channels with memory. Details about the channel codes mentioned in this chapter can be found in Chapter 7.[1]

9.2 PRIORITY ENCODING TRANSMISSION: CROSS-PACKET ERASURE CODING

In a packet network, the transmitted packets can be dropped, delayed, or corrupted (see Chapter 8). By ignoring delayed packets and discarding corrupted ones, one can model the channel as a packet erasure channel, which assumes that a transmitted packet is either correctly received or lost. In this section, we present transmission systems for this channel model. They share the feature that systematic maximum-distance separable (MDS) codes are applied across blocks of packets. Such codes could be Reed–Solomon (RS) codes, punctured RS codes, or shortened RS codes. For simplicity, we call them RS codes in this chapter. RS codes are used for two main reasons. First, as MDS codes, they are optimal in the sense that the smallest possible number of received symbols is used for full recovery of all information symbols. Second, both the encoding and the decoding are very fast when the length of the channel code word is not too large [18].

In Priority Encoding Transmission [1], the information bit stream is partitioned into segments with different priorities. Each segment is protected with a systematic RS code (Table 9.1). Since the packet number is indicated in the packet header, the receiver knows the location of the erased symbols in each codeword. Thus, if the RS code used for a given segment is known, the receiver is able to

[1] Parts of this work were previously published in [12].

Table 9.1: Priority encoding transmission with eight transmitted packets. The information bit stream is partioned into three segments with different priorities. Numbers denote information symbols and x denotes a redundant symbol. The first segment consists of the first 6 symbols of the message and is protected with an $(8, 2)$ systematic RS code. The second segment consists of the next 6 symbols of the message and is protected with an $(8, 3)$ systematic RS code. The third segment consists of the next 10 symbols of the message and is protected with an $(8, 5)$ systematic RS code.

Packet 1	1	2	3	7	8	13	14
Packet 2	4	5	6	9	10	15	16
Packet 3	x	x	x	11	12	17	18
Packet 4	x	x	x	x	x	19	20
Packet 5	x	x	x	x	x	21	22
Packet 6	x	x	x	x	x	x	x
Packet 7	x	x	x	x	x	x	x
Packet 8	x	x	x	x	x	x	x

reconstruct the segment when the number of packets lost does not exceed the number of parity symbols for this code. For example, if for the system of Table 9.1 two packets are lost, then the receiver can recover all information symbols. Given the length of the packet payload, the length of the information bit stream, the number of priority levels, the length of the bit stream in each priority level, and the RS code rate for each segment, the authors [1] provide an algorithm that computes the number of packets sent and the number of information packets in each segment.

In [15], priority encoding transmission was used for embedded information bit streams and extended by allowing the number of segments to be equal to the number of symbols in the packet payload. This is done as follows. Suppose that the encoded bit stream is to be sent as N packets of payload size L symbols each. Then the system builds L segments $S_1, \ldots, S_L$, each of which consists of $m_i \in \{1, \ldots, N\}$ information symbols and protects each segment S_i with an (N, m_i) systematic RS code (Table 9.2).

For each $i \in \{1, \ldots, L\}$, let $f_i = N - m_i$ denote the number of parity symbols that protect segment S_i. If n packets of N are lost during transmission, then the RS codes ensure that all segments that contain at most $N - n$ source symbols can be recovered. Thus, by adding the monotonicity constraint $f_1 \geq f_2 \geq \cdots \geq f_L$, if at most f_i packets are lost, then the receiver can recover at least the first i segments. This monotonicity constraint is justified by the fact that if a segment cannot be recovered, then all the next segments are useless. In the example of Table 9.2, suppose that any three packets are lost. Then the receiver can reconstruct the first four segments and thus decode the first eight information symbols.

Table 9.2: Block of packets. There are $N = 6$ packets of $L = 5$ symbols each. Numbers denote information symbols of an embedded bit stream and x denotes a parity symbol.

Packet 1	1	2	3	6	9
Packet 2	x	x	4	7	10
Packet 3	x	x	5	8	11
Packet 4	x	x	x	x	12
Packet 5	x	x	x	x	13
Packet 6	x	x	x	x	14

Denote by $\mathcal{F}_L$ the set of L-tuples $(f_1, \ldots, f_L)$ such that $f_i \in \{0, \ldots, N-1\}$ for $i = 1, \ldots, L$ and $f_1 \geq f_2 \geq \cdots \geq f_L$. Let ϕ be the operational distortion–rate function of the source coder and X be the random variable whose value is the number of packets lost when N packets are sent. For a given L-segment protection $F = (f_1, \ldots, f_L) \in \mathcal{F}_L$, the expected distortion is

$$E[d](F) = \sum_{i=0}^{L} P_i(F)\phi\big(V_i(F)\big), \tag{9.1}$$

where $V_0(F) = 0$, and for $i = 1, \ldots, L$, $V_i(F)$ is the number of information symbols in the first i segments, that is, $V_i(F) = \sum_{k=1}^{i} m_k = iN - \sum_{k=1}^{i} f_k$, and

$$P_i(F) = \begin{cases} Prob(X > f_1) & \text{for } i = 0; \\ Prob(f_{i+1} < X \leq f_i) & \text{for } i = 1, \ldots, L-1; \\ Prob(X \leq f_L) & \text{for } i = L. \end{cases}$$

Let $p_N(n)$ denote the probability that exactly n packets of N are lost. Then for $i = 1, \ldots, L-1$, we have

$$P_i(F) = \begin{cases} 0 & \text{if } f_i = f_{i+1}; \\ \sum_{n=f_{i+1}+1}^{f_i} p_N(n) & \text{otherwise.} \end{cases}$$

An L-segment protection that minimizes (9.1) over $\mathcal{F}_L$ is called distortion optimal. We point out that the expected distortion (9.1) can also be expressed [9] in the equivalent form

$$c_N(N)\phi\big(V_0(F)\big) + \sum_{i=1}^{L} c_N(f_i)\big(\phi\big(V_i(F)\big) - \phi\big(V_{i-1}(F)\big)\big),$$

where $c_N(k) = \sum_{n=0}^{k} p_N(n)$, $k = 0, \ldots, N$. Thus, $c_N(f_i)$ is the probability that the receiver correctly recovers segment S_i.

Another way to look at the expected distortion was given in [17]. For any L-segment protection $F = (f_1, \ldots, f_L) \in \mathcal{F}_L$, there exists a unique N-tuple $\mathbf{R} = (R_1, \ldots, R_N)$ with $R_1 \leq \cdots \leq R_N$ such that for $i = 1, \ldots, N$, R_i denotes the number of successfully decoded information symbols if exactly i packets of N are received. For example, in Table 9.2, we have $R_1 = 2$, $R_2 = 2$, $R_3 = 8$, $R_4 = 8$, $R_5 = 8$, and $R_6 = 14$. The expected distortion for $\mathbf{R} = (R_1, \ldots, R_N)$ is $E[d](\mathbf{R}) = \sum_{j=0}^{N} q_N(j)\phi(R_j)$, where $R_0 = 0$ and $q_N(i) = 1 - p_N(i)$ is the probability that exactly i of N packets are received. It is easy to show that the total transmission rate (in symbols) is $R_t(\mathbf{R}) = \sum_{j=1}^{N} \alpha_j R_j$, where $\alpha_j = \frac{N}{j(j+1)}$, $j = 1, \ldots, N-1$, and $\alpha_N = 1$.

9.2.1 Optimization

We now discuss algorithms for minimizing the expected distortion (9.1). Since the number of possible candidates $F \in \mathcal{F}_L$ is $\binom{L+N-1}{L}$, finding an optimal solution by brute force is infeasible in practice.

Puri and Ramchandran [17] noted that an optimal solution can be determined by finding $\mathbf{R} = (R_1, \ldots, R_N)$ that minimizes $E[d](\mathbf{R})$ subject to (1) $R_t(\mathbf{R}) \leq NL$, (2) $R_1 \leq R_2 \leq \cdots \leq R_N$, and (3) $R_i - R_{i-1} = k_i i$, $k_i \in \mathbf{N}$, $i = 2, \ldots, N$. Instead of this constrained discrete optimization problem, they first consider the relaxed problem of minimizing $E[d](\mathbf{R})$ for real variables $R_1, \ldots, R_N$ subject only to the first constraint $R_t(\mathbf{R}) \leq NL$. Assuming convexity and differentiability of the operational distortion–rate function, this is done by minimizing the Lagrangian

$$J(\mathbf{R}, \lambda) = \sum_{j=0}^{N} q_N(j)\phi(R_j) + \lambda\left(\sum_{j=1}^{N} \alpha_j R_j - NL\right) \tag{9.2}$$

giving the slopes of the distortion–rate function at the extremal points

$$\frac{d\phi(R_i)}{dR_i} = -\lambda \frac{\alpha_i}{q_N(i)}, \quad i = 1, \ldots, N. \tag{9.3}$$

After the Lagrange multiplier λ is eliminated via a bisectional search, further steps are carried out to handle constraints 2 and 3 [17]. The algorithm finds near-optimal solutions in practice and optimal ones subject to the convexity of ϕ and fractional bit allocation assignments. Its time complexity is $O(kN)$, where k is the number of iterations needed to determine the Lagrange multiplier that corresponds to the given transmission bit budget. Moreover, a preprocessing step that computes the vertices of the lower convex hull of LN distortion–rate points of the source coder is needed.

Mohr *et al.* [14] also determined a solution that is optimal if the operational distortion–rate function is convex and fractional bit allocation assignments are acceptable. The algorithm first computes the h vertices of the lower convex hull of LN distortion–rate points of the source coder and then finds a solution in $O(hN \log N)$ time.

Stockhammer and Buchner [29] presented an $O(N^2L^2)$ dynamic programming algorithm that is almost exact in the general case and exact if the operational distortion–rate function is convex and the packet loss probability $p_N(n)$ is a monotonically decreasing function of the number of lost packets n. Dumitrescu *et al.* [9] independently found the same algorithm. However, they showed that its complexity can be reduced to $O(NL^2)$. Moreover, they gave an $O(N^2L^2)$ algorithm that finds an optimal solution in the general case.

A fast local search algorithm that computes a near-optimal solution in practice was presented in [27]. The idea is to start from a *rate-optimal* solution and then iteratively improve it by searching for a better solution in its neighborhood (Figure 9.1). Here a rate-optimal solution is defined as an L-segment protection that maximizes over $\mathcal{F}_L$, the expected number of correctly reconstructed source symbols given by

$$E[r](F) = \sum_{i=0}^{L} P_i(F) V_i(F). \tag{9.4}$$

A rate-optimal L-segment protection is shown [27] to be the equal loss protection strategy $(f_r, \ldots, f_r)$, with

$$f_r = \arg \max_{i=0,\ldots,N-1} (N-i) \sum_{n=0}^{i} p_N(n).$$

Thus its computation is straightforward and can always be done in $O(N)$ steps. The local search (or iterative improvement) part needs at most $L(N-1)+1$ computations and $L(N-1)$ comparisons of cost function (9.1), bringing the overall worst-case time complexity to $O(NL)$.

9.2.2 Multiple Blocks of Packets (BOPs)

When the transmission rate budget R_T (in symbols) is large and the packet payload size L is small, the channel codeword length N has to be very large to guarantee that $LN = R_T$, making channel encoding and decoding very complex in software. A solution is to keep N small and use more than one block of packets (Table 9.3).

This problem is considered in [31], where a heuristic algorithm for efficiently determining the packet erasure protection in each block of packets is given. The

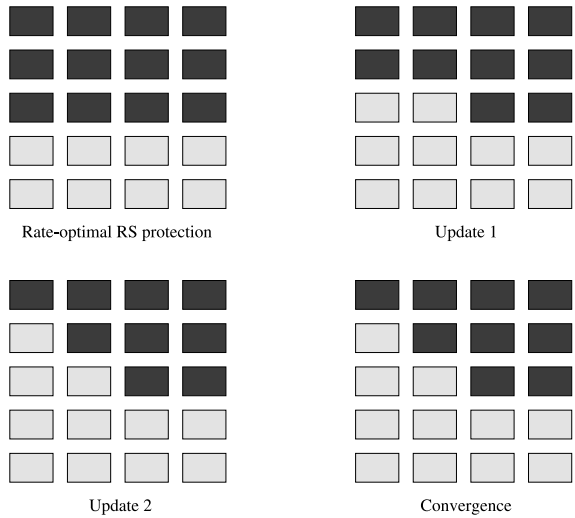


FIGURE 9.1: Illustration of the local search algorithm for $N = 5$ and $L = 4$. The neighbors of a solution (f_1, f_2, f_3, f_4) are $(f_1 + 1, f_2, f_3, f_4)$, $(f_1 + 1, f_2 + 1, f_3, f_4)$, $(f_1 + 1, f_2 + 1, f_3 + 1, f_4)$, and $(f_1 + 1, f_2 + 1, f_3 + 1, f_4 + 1)$. Either the current solution is updated with the best neighbor or the algorithm stops. Dark areas correspond to information symbols and light areas to parity symbols.

algorithm is iterative and converges to a local minimum. Dumitrescu and Wu [8] proposed a dynamic programming algorithm that finds an optimal solution to the problem. The time complexity of the algorithm is $O(K^2L^2N^2)$, where K is the number of transmitted BOPs.

9.2.3 Ensuring Quality of Service

One limitation of basing the protection strategy on minimizing the expected distortion (or maximizing the expected PSNR) is that no quality of service (QoS) is

Table 9.3: Multiple BOPs. The transmission rate budget is $R_T = 30$, the packet payload size is $L = 5$, the codeword length of the channel codes is $N = 3$, and $K = 2$ BOPs are sent.

Packet 1	1	2	4	6	9	Packet 4	12	13	14	16	19
Packet 2	x	3	5	7	10	Packet 5	x	x	15	17	20
Packet 3	x	x	x	8	11	Packet 6	x	x	x	18	21

Table 9.4: Packet loss protection with QoS. Here $k' = 2$ and $L_1 = 3$. The first six symbols provide a distortion smaller than $d_{\min}$.

Packet 1	1	3	5	7	10
Packet 2	2	4	6	8	11
Packet 3	x	x	x	9	12
Packet 4	x	x	x	x	13
Packet 5	x	x	x	x	14

guaranteed. Indeed, since the distortion is minimized on average, there may be transmissions where the distortion is too high for meaningful applications. One way to alleviate the problem is to add a constraint on the probability of such occurrences [10]. More precisely, for $F \in \mathcal{F}_L$, define $p(F)$ as the probability that the distortion is above a quality threshold $d_{\min}$. Then one looks for a protection F that minimizes the expected distortion (9.1) subject to the constraint $p(F) < p_{\max}$, where $p_{\max}$ is a probability threshold.

To reduce the complexity of the problem, a suboptimal algorithm is proposed in [10]. First one determines the largest integer k' such that the probability of receiving fewer than k' packets is smaller than $p_{\max}$, that is, k' is the largest integer such that $\sum_{n=0}^{k'-1} q_N(n) < p_{\max}$. Second, one determines the smallest integer L_1 such that $\phi(k'L_1) < d_{\min}$. Then the first L_1 columns are protected with an (N, k') systematic RS code. The choice of k' ensures that the probability that the distortion is larger than $d_{\min}$ will be smaller than $p_{\max}$. Finally, an optimal unequal error protection for the remaining $L - L_1$ columns is computed in the usual way (Table 9.4).

9.2.4 Layered Multiple Description Coding

In multicast applications, clients usually have differing transmission bandwidths. Instead of sending a separate block of packets to each client, Chou and colleagues [6] proposed to design a single block of packets consisting of a base layer and additional refinement layers. For example, when two clients are present, the low-bandwidth client receives only the base layer, while the high-bandwidth client additionally receives an enhancement layer (Table 9.5).

Unfortunately, this construction cannot offer to both clients the same quality performance as two separate, optimal, nonlayered multiple description schemes. A naive method to solve the problem by optimizing the protection for only one client usually leads to very high distortions for the nonoptimized client [6]. A better approach is to optimize the protection for the low-bandwidth client and reallocate a number of parity packets from the enhancement layer of the high-bandwidth client to the base layer to strengthen its protection [6]. However, the solution was

Table 9.5: The first three packets (base layer) are sent to both low-bandwidth and high-bandwidth clients. The remaining four packets are sent to the high-bandwidth client only. Numbers denote information symbols, x denotes a parity symbol. Packets 4 and 5 provide supplementary protection to the base layer.

Packet 1	1	2	4	6
Packet 2	x	3	5	7
Packet 3	x	x	x	8
Packet 4	x	x	x	x
Packet 5	x	x	x	x
Packet 6	9	10	11	12
Packet 7	x	x	x	13

optimized only for the low-bandwidth client, and the high-bandwidth client potentially suffered a significant performance loss. For example, for the Foreman video sequence encoded with MPEG-4 FGS, the expected distortion for the high-bandwidth client was 1.4 dB worse than the smallest possible distortion for this client [6].

A better trade-off between the distortions seen by the clients can be obtained by minimizing the largest performance loss experienced by any client [28]. Such a code tends to average the quality loss among the clients, ensuring that none of the clients suffers a significantly higher quality degradation than the others. Two fast heuristic algorithms for the setup with two clients were proposed in [28]. Experimental results show that the algorithms provide significant improvements in the quality trade-off over the results of [6]. Finding an optimal solution in polynomial time for the two-client case is still an open problem. Another open problem is to compute a fast optimal or near-optimal solution when there are more than two clients.

9.3 CRC+RCPC: WITHIN-PACKET ERROR DETECTION AND CORRECTION CODING

In this section, we present a transmission system based on a concatenation of an error detection code as an outer code and an error correction code as an inner code. Although this system was already known in the communications literature [20], it was Sherwood and Zeger [22] who first combined it with an embedded source coder for progressively transmitting images. The motivation for using the concatenated code is that it allows the receiver to stop the decoding as soon as the first noncorrectable error is detected. In this way, error propagation, which can

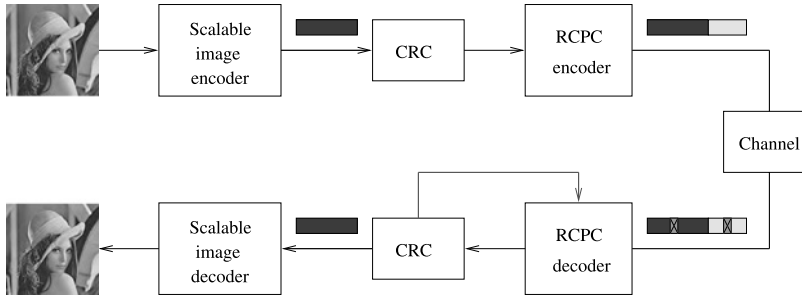


FIGURE 9.2: Schematic representation of the system of Sherwood and Zeger [22]. The dark areas correspond to information bits and the light areas to parity bits.

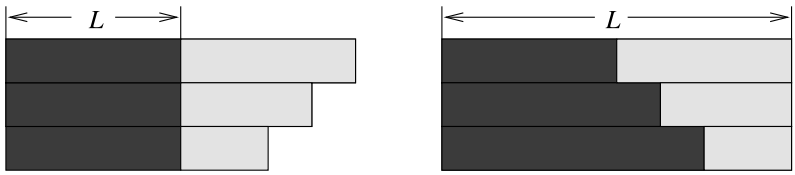
be catastrophic for variable-length source codes, is avoided. For illustration purposes, we will assume that the source coder is an embedded image coder; however, the reader should keep in mind that any embedded or scalable source coder can be used as well.

Figure 9.2 shows a schematic representation of the transmission system proposed in [22]. After an embedded coder is used to compress the source, the compressed bit stream is divided into consecutive information blocks. To each information block, CRC parity bits are appended, and the resulting block is encoded with an RCPC coder. The receiver uses a List Viterbi algorithm (LVA) to find a best maximum-likelihood decoding solution for a received packet. If an error is detected by the CRC decoder, the LVA is used again to find a next best solution, which is also checked for errors. This process is repeated until the CRC test is passed or a maximum number of solutions is reached. In the first case, the next received packet is considered, whereas in the second case, decoding is stopped and the image is reconstructed using only the correctly decoded packets.

In the following, we explain how to optimally allocate a transmission bit budget between the source coder and the channel coder for the system. Three different performance criteria are considered: the expected distortion, the expected PSNR (defined in decibels as $10\log_{10}\frac{255^2}{MSE}$), and the expected number of correctly decoded information bits. In Section 9.3.1, we assume as in [22] that the lengths of all information blocks are fixed. In Section 9.3.2, we treat the case where the size of the channel codewords is fixed [2,25] (Figure 9.3).

9.3.1 Fixed-Length Information Blocks

Let N_p be the number of pixels in the image and L be the length of the information blocks. Suppose that the channel coder is given by a family $\mathcal{C} = \{c_1, \ldots, c_m\}$



(a) (b)

FIGURE 9.3: (a) Fixed-length information blocks and variable-length channel codewords. (b) Fixed-length channel codewords with variable-length information blocks. The dark areas correspond to information bits and the light areas to parity bits.

of error correction-detection codes. For $i = 1, \ldots, m$, let $r(c_i)$ and $p(c_i)$ denote the code rate and the probability of incomplete decoding for channel code c_i, respectively. It is assumed that all decoding errors can be detected. The protection of the information blocks is given by an error protection strategy (EPS) $\pi = (\pi_1, \ldots, \pi_{N(\pi)})$, where $N(\pi)$ is the number of transmitted information blocks, and for $i = 1, \ldots, N(\pi)$, $\pi_i \in \mathcal{C}$ is the channel code used to protect the ith information block (Figure 9.3a).

For $k = 1, \ldots, N(\pi)$ and $i = k-1, \ldots, N(\pi)$, let $P_{i|k-1}(\pi)$ denote the conditional probability that exactly the first i packets are decoded correctly given that the first $k-1$ packets are decoded correctly. Since the channel is memoryless, we have

$$P_{i|k-1}(\pi) = \begin{cases} p(\pi_k) & \text{for } i = k-1; \\ p(\pi_{i+1}) \prod_{j=k}^{i} \big(1 - p(\pi_j)\big) & \text{for } i = k, \ldots, N(\pi) - 1; \\ \prod_{j=k}^{N(\pi)} \big(1 - p(\pi_j)\big) & \text{for } i = N(\pi). \end{cases}$$

In particular, when $k = 1$, we have $P_{0|0}(\pi) = p(\pi_1)$ is the probability that the first packet is not correctly decoded, for $i = 1, \ldots, N(\pi) - 1$, $P_{i|0}(\pi) = p(\pi_{i+1}) \prod_{j=1}^{i} (1 - p(\pi_j))$ is the probability that the first i packets are correctly decoded while the next one is not, and $P_{N(\pi)|0}(\pi) = \prod_{j=1}^{N(\pi)} (1 - p(\pi_j))$ is the probability that all $N(\pi)$ packets are correctly decoded.

Let $\phi(r)$ be the operational distortion–rate function of the source coder with the rate r given in bpp. Then the expected distortion for an EPS π is

$$E[d](\pi) = \sum_{i=0}^{N(\pi)} P_{i|0}(\pi) \phi(iL/N_p). \tag{9.5}$$

The expected PSNR for the EPS π is

$$E[PSNR](\pi) = \sum_{i=0}^{N(\pi)} P_{i|0}(\pi) PSNR(iL/N_p). \tag{9.6}$$

Finally, the expected reconstructed source coding rate is

$$E[r](\pi) = \sum_{i=0}^{N(\pi)} P_{i|0}(\pi)(iL/N_p). \tag{9.7}$$

Ideally, one would like to minimize (9.5) or maximize (9.6). However, maximizing (9.7) is also reasonable for an efficient embedded coder because the expected distortion will generally decrease when the expected number of correctly decoded source bits increases. Note, however, that the optimization of (9.5), (9.6), and (9.7) does not necessarily yield the same solution, with one exception being the trivial case where ϕ is a linear function. One nice feature of maximizing (9.7) is that the solution is not dependent on the source contents or the source coder and thus can be computed off-line. Moreover, this solution can also be determined by the receiver and need not be transmitted over the channel.

For $k = 1, \ldots, N(\pi)$ let $\Delta(k, \pi) = \sum_{i=k}^{N(\pi)} (\sum_{j=k}^{i} \delta_j) P_{i|k-1}(\pi)$, where

$$\delta_i = \begin{cases} \phi\big((i-1)L/N_p\big) - \phi(iL/N_p) & \text{for the distortion;} \\ PSNR(iL/N_p) - PSNR\big((i-1)L/N_p\big) & \text{for the PSNR;} \\ 1, & \text{for the expected source rate.} \end{cases}$$

Then one can show [3] that an EPS that optimizes the performance of the system for a given total transmission bit rate r_T is a strategy $\pi^* = (\pi_1^*, \ldots, \pi_{N(\pi)}^*)$ that maximizes $\Delta(1, \pi)$ subject to $\sum_{i=1}^{N(\pi)} \frac{L}{N_p r(\pi_i)} \le r_T$. This can be done with dynamic programming with complexity $O(r_T^2)$ if either the distortion or the PSNR is optimized, or $O(r_T)$ if the expected reconstructed source rate is maximized [3]. One can also show that for an EPS $\pi^* = (\pi_1^*, \ldots, \pi_{N(\pi)}^*)$ that maximizes the expected reconstructed source rate, we have $r(\pi_k^*) \le r(\pi_{k+1}^*)$ for $k = 1, \ldots, N(\pi) - 1$ [3]. That is, the information blocks should be protected with increasingly weaker channel codes. However, this property does not necessarily hold if either the distortion or the PSNR is optimized. However, the property is satisfied if one assumes that the logarithm of the block decoding error probability is an affine function of the channel packet length [16].

Experimental results show that the solutions to maximizing the expected PSNR and maximizing the expected source rate yield a similar performance for the SPIHT coder and a CRC/RCPC channel coder (the difference in PSNR is less than 0.2 dB for the 512×512 gray-scale Lenna image [3]). This can be analytically confirmed under some theoretical assumptions, including an independent,

identically distributed Gaussian source and a perfect progressive source coder that achieves the distortion–rate function [13].

A further nice feature of rate-based optimization is that if $\pi^* = (\pi_1^*, \ldots, \pi_N^*)$ is rate optimal for target transmission rate r_T, then the EPSs $(\pi_j^*, \ldots, \pi_N^*)$, $j = 2, \ldots, N$ are also rate optimal for target transmission rates $r_j = r_T - \frac{1}{N_p} \sum_{i=1}^{j-1} \frac{L}{r(\pi_i^*)}$ [3]. This result has an important application if rate-compatible channel codes are used. Indeed, in this case, the transmission of the sequence of bits can be organized such that rate-based optimality is guaranteed at the intermediate rates r_j, $j = 2, \ldots, N$ [3].

9.3.2 Fixed-Length Channel Codewords

We now consider the slightly different system of Figure 9.3b [2,25], where the size of the channel codewords is fixed, but the blocks of information bits have variable lengths. Compared to the system using information blocks of fixed length, the obvious advantage of having channel codewords of fixed length is that cross-layer design will be easier because other layers (e.g., the physical layer) do not have to deal with the issue of different channel codeword lengths. Denote the set of channel codes by $\mathcal{C} = \{c_1, \ldots, c_m\}$ and the length of the channel codewords by L. Given a transmission rate budget B (in symbols), the system transforms $N = \lfloor B/L \rfloor$ successive blocks of the source coder output bit stream into a sequence of N channel codewords of fixed length L. Here we use an N-packet EPS $\pi = (\pi_1, \ldots, \pi_N)$, which encodes the kth information block with channel code $\pi_k \in \mathcal{C}$. Like the system in Figure 9.3a with fixed-length information blocks, if the decoder detects an error, then the decoding is stopped, and the message is reconstructed from the correctly decoded packets.

For $i = 1, \ldots, m$, let $p(c_i)$ denote the probability of a decoding error of channel code c_i. We may assume without loss of generality that $p(c_1) < \cdots < p(c_m) < 1$. For the N-packet strategy π, the expected number of correctly decoded source bits is

$$E_N[r](\pi) = \sum_{i=0}^{N} P_{i|0}(\pi) V_i(\pi), \tag{9.8}$$

where $V_0(\pi) = 0$ and for $i \geq 1$, $V_i(\pi) = \sum_{j=1}^{i} v(\pi_j)$ with $v(\pi_j) = \lfloor L r(\pi_j) \rfloor$ being the number of source bits in the jth packet. The expected distortion is

$$E_N[d](\pi) = \sum_{i=0}^{N} P_{i|0}(\pi) \phi\big(V_i(\pi)\big), \tag{9.9}$$

where $\phi(R)$ is the distortion associated to rate R (in bits).

As in Section 9.2.1, an EPS π^r that maximizes (9.8) is called rate optimal. A rate-optimal EPS can be computed in $O(mN)$ time [25]. This is essentially due to the property that if the N-packet EPS $(\pi_1, \ldots, \pi_N)$ is rate optimal, then for $1 \leq i \leq N-1$ the $(N-i)$-packet EPS $(\pi_{i+1}, \ldots, \pi_N)$ must also be rate optimal [25]. An EPS π^d that minimizes (9.9) is called distortion optimal. However, in contrast to the fixed-information block setting, there is no known polynomial-time algorithm to compute a distortion-optimal EPS π^d. Assuming that the operational distortion–rate function $\phi(R)$ of the source coder is nonincreasing and convex, one can give a tight lower bound of $E_N[d](\pi^d)$ as $\phi(E_N[r](\pi^r))$ [11], which is computable with complexity $O(mN)$. Then if π^r is used as an approximation of π^d, a tight upper bound on the quality loss $E_N[d](\pi^r) - E_N[d](\pi^d)$ is $E_N[d](\pi^r) - \phi(E_N[r](\pi^r))$. In addition, it is conjectured in [11] that under the same assumptions for $\phi(R)$, the total number of information bits for π^d is smaller than or equal to that for π^r.

In [2], an approximation for π^d based on the Viterbi algorithm was proposed. It has a quadratic time complexity in the number of transmitted channel code words N. However, this result is guaranteed only for channel code rates that are a subset of $\{\frac{p}{q}, \frac{p+1}{q}, \ldots, \frac{q-1}{q}\}$, where p and q are positive integers with $p < q$. For channel codes that do not fulfill this condition, including rate-compatible punctured codes, the worst-case time complexity is exponential in N.

A fast local search algorithm was proposed in [11] to compute an approximation of π^d. The algorithm works by iterative improvement and has an $O(mN)$ worst-case complexity [11]. It starts from a rate-optimal solution π^r and then considers the first neighbor of π^r. If the expected distortion of this neighbor is smaller than that of the current solution, it updates the current solution and repeats the procedure; otherwise it considers the next neighbor and repeats the procedure. If there is no neighbor that is better than the current solution, the algorithm returns the current solution. Here a neighbor of an N-packet EPS π with nondecreasing rates is any N-packet EPS with nondecreasing rates, fewer information symbols than π, and differs in code rates from π on only one packet. For example, suppose $\mathcal{C} = \{c_1, c_2, c_3, c_4\}$, if $\pi = (c_1, c_2, c_4, c_4)$, then π has three neighbors, (c_1, c_2, c_3, c_4) being the first one, (c_1, c_2, c_2, c_4) the second, and (c_1, c_1, c_4, c_4) the third. Note how in observance of the conjecture in [11], all solutions tested by the local search algorithm have fewer information bits than π^r.

9.4 ERROR PROTECTION FOR WIRELESS NETWORKS

In fading channels, the transmitted packets experience different bit error rates: packets transmitted when the channel is in the bad state are exposed to much higher bit error rates than those transferred during the good state of the channel. Thus, to avoid decoding failures with the CRC/RCPC system of [22], the RCPC

codes should be designed for the bit error rate in the channel's bad state. This causes overprotection during the good state of the channel (which usually lasts much longer) and bounds the achievable performance from the theoretical limits. In this section, we present three successful extensions of the CRC/RCPC system of [22] for fading channels. The first system [30] introduces interleaving. The two other systems [19,23] are based on product channel codes.

9.4.1 Using Interleaving

Interleaving tends to spread deep fade and to transform a memory channel into a memoryless one. It improves the performance during transmission over fading channels at the expense of increased complexity and time delay. A system that exploits block interleaving to alleviate the problems of channel burst errors during a deep fade is that of Stockhammer and Weiss [30]. As in the system of [22], an embedded bit stream is encoded with a punctured convolutional coder, and the decoding of the received bit stream is stopped when the first decoding error is detected. The convolutional coder has a strong systematic mother code of memory 96 and code rate $1/7$. Channel decoding is done with the Fano algorithm. A distortion-optimal unequal error protection solution is determined using dynamic programming as in [3].

9.4.2 Product Code System

Sherwood and Zeger [23] proposed a transmission system based on a product channel code to protect the embedded information bit stream. The product code uses the concatenated CRC/RCPC code of [22] as the row code and a systematic RS code as the column code (Table 9.6). The main idea is to strengthen the protection of the CRC/RCPC code by using channel coding across the packets.

Table 9.6: EEP with the product code of Sherwood and Zeger [23]. There are $N = 6$ RCPC codewords, each of which has a payload of 11 symbols. Cells labeled by numbers contain successive information symbols of an embedded bit stream, x denotes a parity symbol of a $(3, 2)$ RS code, $+$ a CRC parity symbol, and o an RCPC parity symbol. Each column contains two different RS codewords. The RCPC code need not be systematic.

Packet 1	1	2	3	4	5	+	+	o	o	o	o
Packet 2	6	7	8	9	10	+	+	o	o	o	o
Packet 3	x	x	x	x	x	+	+	o	o	o	o
Packet 4	11	12	13	14	15	+	+	o	o	o	o
Packet 5	16	17	18	19	20	+	+	o	o	o	o
Packet 6	x	x	x	x	x	+	+	o	o	o	o

Table 9.7: Unequal error protection with the product code system of Sherwood and Zeger [23]. The information symbols of the two blocks are protected with a $(3, 2)$ RS code. Additionally, the information symbols of the first block are protected with a $(4, 2)$ RS code, whose parity symbols are denoted by y.

Packet 1	1	2	3	4	5	+	+	o	o	o	o
Packet 2	6	7	8	9	10	+	+	o	o	o	o
Packet 3	x	x	x	x	x	+	+	o	o	o	o
Packet 4	11	12	13	14	15	+	+	o	o	o	o
Packet 5	16	17	18	19	20	+	+	o	o	o	o
Packet 6	x	x	x	x	x	+	+	o	o	o	o
Packet 7	y	y	y	y	y	+	+	o	o	o	o
Packet 8	y	y	y	y	y	+	+	o	o	o	o

The information bit stream is first partitioned into packets of equal length L, which are then grouped into K blocks of k packets each. All k information packets within a block are protected columnwise by an (n, k) systematic RS code. In this way, $(n - k)$ parity packets are associated to each block, resulting in $N = K \times n$ packets. Finally, each packet (information or parity) is fed to a CRC/RCPC encoder. The transmitter first sends $k \times K$ information packets (more precisely, RCPC codewords obtained by protecting information packets). Then, the parity packets, which may be arbitrarily interleaved to improve performance during a deep fade, are transmitted. Interleaving of the parity packets provides the desired trade-off between performance quality and delay. In the example of Table 9.6, $N = 6$, $L = 5$, $K = 2$, $k = 2$, and $n = 3$.

Finding an optimal RCPC code rate, an optimal RS code rate, and an optimal interleaver for the system is a very difficult problem. Moreover, no efficient method that computes a near-optimal solution is known. Sherwood and Zeger [23] suggest selecting RCPC code so that it can efficiently protect the transmitted data while the channel is in the good state.

In addition to equal error protection, several ways of implementing unequal error protection were proposed in [23]. The most successful one protects the earliest symbols of the embedded bit stream by additional RS codes (Table 9.7).

9.4.3 Another Product Code System

The system proposed by Sachs *et al.* [19] is also based on a product channel code where each row code is a concatenated CRC/RCPC code and each column code is a systematic RS code (Table 9.8).

The embedded information bit stream is first protected with RS codes of length N as in the system of [15] (Table 9.2). Then the CRC parity symbols are added to each row. Finally, each row is encoded with the same RCPC code of length L. The resulting product code is sent as N packets of L symbols each.

Table 9.8: Product code of Sachs *et al.* [19]. There are $N = 6$ packets of $L = 11$ symbols each. Cells labeled by numbers contain successive information symbols of an embedded bit stream, x denotes an RS parity symbol, + a CRC parity symbol, and o an RCPC parity symbol. The RCPC code need not be systematic.

Packet 1	1	3	6	10	15	+	+	o	o	o	o
Packet 2	2	4	7	11	16	+	+	o	o	o	o
Packet 3	x	5	8	12	17	+	+	o	o	o	o
Packet 4	x	x	9	13	18	+	+	o	o	o	o
Packet 5	x	x	x	14	19	+	+	o	o	o	o
Packet 6	x	x	x	x	20	+	+	o	o	o	o

Let $\mathcal{C} = \{c_1, \ldots, c_m\}$ be the set of available RCPC codes. For $c \in \mathcal{C}$, we denote by $L(c)$ the sum of the number of information symbols and RS parity symbols used in a packet protected with c. Thus, we have $L(c)$ information segments $S_1, \ldots, S_{L(c)}$, where segment S_j, $1 \le j \le L(c)$, consists of $m_j \in \{1, \ldots, N\}$ information symbols that are protected by $f_j = N - m_j$ RS symbols.

Packets are sent over the channel in the order in which they are generated. Since all packets are of equal importance, packet interleaving cannot improve the performance. Each received packet is decoded with the RCPC decoder. If the CRC code detects an error, then the packet is considered to be lost (we suppose that all errors can be detected). Suppose now that n packets of N are erased (i.e., either lost during transmission or discarded due to RCPC decoding failure), then the RS codes ensure that all segments that contain at most $N - n$ information symbols can be recovered. By adding the constraint $f_1 \ge \cdots \ge f_{L(c)}$, one guarantees that the receiver can decode at least the first j segments whenever at most f_j packets are erased.

In contrast to the system of [23], which puts the earliest symbols in the first rows, the product code system of [19] puts these symbols in the first columns. Consequently, the first system has a better progressive ability and a shorter decoding delay. However, experimental results [19] for a flat-fading Rayleigh channel and BPSK modulation indicate that the performance of the system of [19] is better than that of [23]. Finally, the performance of the first system depends on which packets are received, whereas in the second system, all packets are of equal importance. For example, suppose that the first two packets in Tables 9.6 and 9.8 are erased. Then, the first system cannot recover any information symbol, while the second system can successfully recover the first nine information symbols. However, if the second and fourth transmission packets are erased, the system in Table 9.6 reconstructs all 20 information symbols, while the system of Table 9.8 will be able to reconstruct only the first 10 symbols.

We now consider strategies for minimizing the expected distortion of the product code system of [19]. As in Section 9.1, we denote by $\mathcal{F}_{L(c)}$, $c \in \mathcal{C}$, the set

of $L(c)$-tuples $(f_1, \ldots, f_{L(c)})$ such that $f_1 \geq \cdots \geq f_{L(c)}$ and $f_j \in \{0, \ldots, N-1\}$ for $j = 1, \ldots, L(c)$. Then a protection (c, F) for the product code is given by an RCPC code $c \in \mathcal{C}$ and an $L(c)$-segment RS protection $F \in \mathcal{F}_{L(c)}$.

A distortion-optimal product code solution (c^*, F^*) is given by an RCPC code $c^* \in \mathcal{C}$ and an $L(c^*)$-segment RS protection $F^* \in \mathcal{F}_{L(c^*)}$ that solve the minimization problem

$$\min_{c \in \mathcal{C}, F \in \mathcal{F}_{L(c)}} \sum_{k=0}^{L(c)} P_k(F) \phi\big(V_k(F)\big), \tag{9.10}$$

where P_k, ϕ, and $V_k(F)$ are defined as in Section 9.1. Solving (9.10) by brute force is impractical because the number of possible product codes is

$$\sum_{c \in \mathcal{C}} \binom{L(c) + N - 1}{L(c)}.$$

In [19], the authors use the Lagrange-based optimization algorithm of [17] to determine a near-optimal $L(c)$ segment RS protection for each $c \in \mathcal{C}$. Then the protection that yields the smallest expected distortion is selected. Even though the Lagrange-based optimization algorithm is fast, the overall optimization can be too expensive when the number of candidate RCPC codes is large.

A fast heuristic method for solving problem (9.10) was proposed in [27]. In contrast to [19], the method of [27] does not try to minimize (9.1) for each RCPC code. Instead, it starts from a rate-optimal product code solution, that is, one that solves the maximization problem

$$\max_{c \in \mathcal{C}, F \in \mathcal{F}_{L(c)}} \sum_{k=0}^{L(c)} P_k(F) V_k(F), \tag{9.11}$$

and then tries to improve this solution by progressively increasing the total number of parity symbols. This is done by alternately applying the local search algorithm of Section 9.2.1 and decreasing the RCPC code rate. The procedure is illustrated in Table 9.9, which is obtained by decreasing the RCPC code rate in Table 9.8. Note how the total number of parity symbols increases.

The method of [27] also exploits the fact that if F is the current RS protection, then one can exclude all RCPC code rates for which the expected distortion is greater than $E[d](F)$. This is because a distortion-optimal RS protection corresponding to one such code rate cannot be better than F. In the worst case, the algorithm computes $(N-1)L(c) + 1$ times the cost function (9.1) for each $c \in \mathcal{C}$.

Table 9.9: Product code obtained from Table 9.8 by decreasing the RCPC code rate. The new RS protection is $(4, 3, 2, 1)$ whereas the old one is $(4, 3, 2, 1, 0)$.

Packet 1	1	3	6	10	+	+	o	o	o	o	o
Packet 2	2	4	7	11	+	+	o	o	o	o	o
Packet 3	x	5	8	12	+	+	o	o	o	o	o
Packet 4	x	x	9	13	+	+	o	o	o	o	o
Packet 5	x	x	x	14	+	+	o	o	o	o	o
Packet 6	x	x	x	x	+	+	o	o	o	o	o

9.5 SUMMARY AND FURTHER READING

We presented FEC-based transmission systems for embedded media bit streams. We emphasized two optimization approaches for these systems. The first one maximizes the expected number of correctly decoded information bits. This approach has two desirable features: the optimization is independent of the source and it can be done very quickly. The second approach minimizes the expected distortion. While its optimization is more complex than that of the first approach, it can provide a significantly better rate–distortion performance in many situations.

We conclude this chapter with suggestions for further reading. FEC can be combined with other error-resilient techniques for robust media transmission. For example, Cosman *et al.* [7] combined FEC with error-resilient source coding. This was done by packetizing the compressed media bit stream into independently decodable packets before applying the CRC/RCPC system of Section 9.3. The system of [7] was later improved in [26]. Chande *et al.* [4] proposed a media transmission system for the binary symmetric channel that combines FEC with ARQ. The media data is first compressed with an embedded coder and encoded with the CRC/RCPC coder of Section 9.3. The receiver decodes the RCPC code and uses the CRC code to check for errors. If no errors are detected, the receiver sends an acknowledgment bit to the sender, which then proceeds with the next packet. If errors are detected, a no acknowledgment bit is sent to the sender, which then transmits additional parity bits of a stronger RCPC code. The procedure is repeated until the decoding of the packet is successful or no stronger RCPC code is available. In the latter case, packet decoding is stopped, and the media data is reconstructed from the correctly decoded packets only. The system of [4] was extended in [5] to the Gilbert–Elliot channel.

REFERENCES

[1] A. Albanese, J. Blömer, J. Edmonds, M. Luby, and M. Sudan. "Priority encoding transmission," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1737–1744, November 1996.

[2] B. A. Banister, B. Belzer, and T. R. Fischer. "Robust image transmission using JPEG2000 and turbo-codes," *IEEE Signal Processing Letters*, vol. 9, pp. 117–119, April 2002.

[3] V. Chande and N. Farvardin. "Progressive transmission of images over memoryless noisy channels," *IEEE JSAC*, vol. 18, pp. 850–860, June 2000.

[4] V. Chande, H. Jafarkhani, and N. Farvardin. "Joint source-channel coding of images for channels with feedback," *Proc. IEEE Workshop Information Theory*, San Diego, February 1998.

[5] V. Chande, N. Farvardin, and H. Jafarkhani. "Image communication over noisy channels with feedback," *IEEE ICIP-99*, Kobe, October 1999.

[6] P. A. Chou, H. J. Wang, and V. N. Padmanabhan. "Layered multiple description coding," *Proc. 13th Intl. Packet Video Workshop*, Nantes, France, April 2003.

[7] P. Cosman, J. Rogers, P. G. Sherwood, and K. Zeger. "Combined forward error control and packetized zerotree wavelet encoding for transmission of images over varying channels," *IEEE Trans. Image Processing*, vol. 9, pp. 982–993, June 2000.

[8] S. Dumitrescu and X. Wu. "Globally optimal uneven erasure-protected multi-group packetization of scalable codes," *Proc. IEEE ICME 2005*, pp. 900–903, Amsterdam, July 2005.

[9] S. Dumitrescu, X. Wu, and Z. Wang. "Globally optimal uneven error-protected packetization of scalable code streams," *IEEE Trans. Multimedia*, vol. 6, pp. 230–239, April 2004.

[10] M. Grangetto, E. Magli, and G. Olmo. "Ensuring quality of service for image transmission: Hybrid loss protection," *IEEE Trans. Image Processing*, vol. 13, pp. 751–757, June 2004.

[11] R. Hamzaoui, V. Stanković, and Z. Xiong. "Fast algorithm for distortion-based error protection of embedded image codes," *IEEE Trans. Image Processing*, vol. 14, pp. 1417–1421, October 2005.

[12] R. Hamzaoui, V. Stanković, and Z. Xiong. "Optimized error protection of scalable image bitstreams," *IEEE Signal Processing Magazine*, vol. 22, pp. 91–107, November 2005.

[13] A. Hedayat and A. Nosratinia. "Rate allocation criteria in source-channel coding of images," *Proc. IEEE ICIP'01*, vol. 1, pp. 189–192, Thessaloniki, Greece, October 2001.

[14] A. E. Mohr, R. E. Ladner, and E. A. Riskin. "Approximately optimal assignment for unequal loss protection," *Proc. IEEE ICIP'00*, vol. 1, pp. 367–370, Vancouver, BC, September 2000.

[15] A. E. Mohr, E. A. Riskin, and R. E. Ladner. "Unequal loss protection: Graceful degradation of image quality over packet erasure channels through forward error correction," *IEEE JSAC*, vol. 18, pp. 819–828, June 2000.

[16] A. Nosratinia, J. Lu, and B. Aazhang. "Source-channel rate allocation for progressive transmission of images," *IEEE Trans. Commun.*, vol. 51, pp. 186–196, February 2003.

[17] R. Puri and K. Ramchandran. "Multiple description source coding using forward error correction codes," *Proc. 33rd Asilomar Conference on Signal, Systems, and Computers*, vol. 1, pp. 342–346, Pacific Grove, CA, October 1999.

[18] L. Rizzo, "On the feasibility of software FEC," DEIT Technical Report LR-970131.

[19] D. G. Sachs, R. Anand, and K. Ramchandran. "Wireless image transmission using multiple-description based concatenated code," *Proc. VCIP'00*, vol. 3974, pp. 300–311, San Jose, CA, January 2000.

[20] N. Seshadri and C.-E. W. Sundberg. "List Viterbi decoding algorithms with applications," *IEEE Trans. Commun.*, vol. 42, pp. 313–323, February–April 1994.

[21] P. G. Sherwood, X. Tian, and K. Zeger. "Channel code blocklength and rate optimization for progressive image transmission," *Proc. IEEE WCNC'99*, vol. 2, pp. 978–982, New Orleans, LA, 1999.

[22] P. G. Sherwood and K. Zeger. "Progressive image coding for noisy channels," *IEEE Signal Processing Letters*, vol. 4, pp. 189–191, July 1997.

[23] P. G. Sherwood and K. Zeger, "Error protection for progressive image transmission over memoryless and fading channels," *IEEE Trans. Commun*, vol. 46, pp. 1555–1559, December 1998.

[24] V. Stanković, R. Hamzaoui, Y. Charfi, and Z. Xiong. "Real-time unequal error protection algorithms for progressive image transmission," *IEEE JSAC*, vol. 21, pp. 1526–1535, December 2003.

[25] V. Stanković, R. Hamzaoui, and D. Saupe. "Fast algorithm for rate-based optimal error protection of embedded codes," *IEEE Trans. Commun.*, vol. 51, pp. 1788–1795, November 2003.

[26] V. Stanković, R. Hamzaoui, and Z. Xiong. "Efficient channel code rate selection algorithms for forward error correction of packetized multimedia bitstreams in varying chanels," *IEEE Trans. Multimedia*, vol. 6, pp. 240–248, April 2004.

[27] V. Stanković, R. Hamzaoui, and Z. Xiong. "Real-time error protection of embedded codes for packet erasure and fading channels," *IEEE Trans. Circuits and Systems for Video Tech.*, vol. 14, pp. 1064–1072, August 2004.

[28] V. Stanković, R. Hamzaoui, and Z. Xiong. "Robust layered multiple description coding of scalable media data for multicast," *IEEE Signal Proc. Letters*, vol. 12, pp. 154–157, February 2005.

[29] T. Stockhammer and C. Buchner. "Progressive texture video streaming for lossy packet networks," *Proc. 11th Intl. Packet Video Workshop*, Kyongju, May 2001.

[30] T. Stockhammer and C. Weiss. "Channel and complexity scalable image transmission," *Proc. IEEE ICIP'01*, vol. 1, pp. 102–105, Thessaloniki, Greece, October 2001.

[31] J. Thie and D. Taubman. "Optimal erasure protection assignment for scalable compressed data with small channel packets and short channel codewords," *EURASIP Journal on Applied Signal Processing*, no. 2, pp. 207–219, February 2004.

[32] Y. Wang, S. Wenger, J. Wen, and A. Katsaggelos. "Error resilient video coding techniques," *IEEE Signal Proc. Magazine*, pp. 61–82, July 2000.

[33] Y. Wang and Q.-F. Zhu. "Error control and concealment for video communication: A review," *Proc. IEEE*, vol. 86, pp. 974–997, May 1998.

This page intentionally left blank

10

Network-Adaptive Media Transport

Mark Kalman and Bernd Girod

10.1 INTRODUCTION

Internet packet delivery is characterized by variations in throughput, delay, and loss, which can severely affect the quality of real-time media. The challenge is to maximize the quality of audio or video at the receiver, while simultaneously meeting bit-rate limitations and satisfying latency constraints. For the best end-to-end performance, Internet media transmission must adapt to changing network characteristics; it must be *network adaptive*. It should also be *media aware*, so that adaptation to changing network conditions can be performed intelligently.

A typical streaming media system comprises four major components that should be designed and optimized in concert:

1. The *encoder application* compresses video and audio signals and uploads them to the media server.
2. The *media server* stores the compressed media streams and transmits them on demand, often serving hundreds of clients simultaneously.
3. The *transport mechanism* delivers media packets from the server to the client for the best possible user experience, while sharing network resources fairly with other users.
4. The *client application* decompresses and renders the video and audio packets and implements the interactive user controls.

The streaming media client typically employs error detection and concealment to mitigate the effects of lost packets. These techniques have been discussed in Chapter 2 for video and Chapter 3 for audio. To adapt to network conditions, the server receives feedback from the client, for example, as positive or negative

acknowledgments. More sophisticated client feedback might inform about packet delay and jitter, link speeds, or congestion.

Unless firewalls force them to, streaming media systems do not rely on TCP but implement their own, application-layer transport mechanisms. This allows for protocols that are both network adaptive and media aware. A transport protocol may determine, for example, when to retransmit packets for error control and when to drop packets to avoid network congestion. If the protocol takes into consideration the relative importance of packets and their mutual dependencies, audio or video quality can be greatly improved.

The media server can implement intelligent transport by sending the right packets at the right time, but the computational resources available for each media stream are often limited because a large number of streams must be served simultaneously. Much of the burden of an efficient and robust system is therefore on the encoder application, which, however, cannot adapt to the varying channel conditions and must rely on the media server for this task. Rate scalable representations are therefore desirable to facilitate adaptation to varying network throughput without requiring computation at the media server. Switching among bit streams encoded at different rates is an easy way to achieve this task, and this method is widely used in commercial systems. Embedded scalable representations, as discussed in Chapter 5 for video and Chapter 6 for audio, are more elegant and are preferable, if the rate–distortion penalty often associated with scalable coding can be kept small.

This chapter begins in Section 10.2 with a review of the framework for rate–distortion optimized media streaming initially proposed by Chou and Miao [6]. In the sections that follow, we discuss extensions to the framework. Section 10.3 shows how rich acknowledgments can be incorporated. In Section 10.4, we discuss the importance of multiple deadlines for video packets. Section 10.5 discusses how the framework can be extended to include a more accurate statistical model for characterizing packet loss and delay. In Section 10.6, finally, we discuss an alternative to rate–distortion optimized streaming that directly minimizes congestion instead of rate.

10.2 RATE–DISTORTION OPTIMIZED STREAMING

We start by reviewing the seminal work by Chou and Miao on rate–distortion optimized (RaDiO) streaming [6]. They considered streaming as a stochastic control problem, with the goal of determining which packets to send and when to minimize reconstruction distortion at the client for a given average transmission rate. Our discussion serves as the starting point for the extensions and variations described in the later sections.

10.2.1 Basic RaDiO Framework

Let us assume that a media server has stored a compressed audio or video stream that has been packetized into data units. Each data unit l has a size in bytes B_l and a deadline by which it must arrive at the client in order to be useful for decoding. The importance of each data unit is captured by its *distortion reduction* ΔD_l, a value representing the decrease in distortion that results if the data unit is decoded. Often, distortion is expressed as mean-squared error, but other distortion measures might be used as well.

Whether a data unit can be decoded often depends on which other data units are available. In the RaDiO framework, these interdependencies are expressed in a directed acyclic graph. An example dependency graph is shown for SNR-scalable video encoding with Intra (I), Predicted (P), and Bidirectionally predicted (B) frames (Figure 10.1). Each square represents a data unit and the arrows indicate the order in which data units can be decoded.

The RaDiO framework can be used to choose an optimal set of data units to transmit at successive transmission opportunities. These transmission opportunities are assumed to occur at regular time intervals. Because of decoding dependencies among data units, the importance of transmitting a packet at a given transmission opportunity often depends on which packets will be transmitted in the near future. The scheduler therefore makes transmission decisions based on an entire optimized plan that includes anticipated later transmissions. Of course, to keep the system practical, only a finite time horizon can be considered.

The plan governing packet transmissions that will occur within a time horizon is called a *transmission policy*, π. Assuming a time horizon of N transmission opportunities, π is a set of length-N binary vectors π_l, with one such vector for each data unit l under consideration for transmission. In this representation, the N binary elements of π_l indicate whether, under the policy, the data unit l will

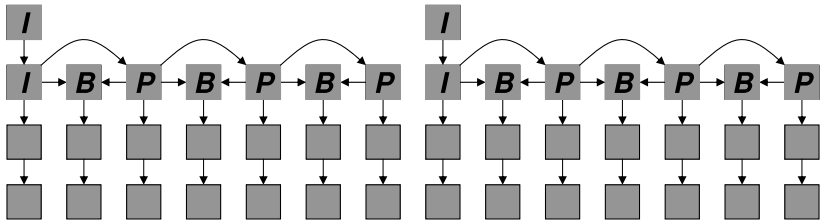


FIGURE 10.1: A directed acyclic graph captures the decoding dependencies for an SNR-scalable encoding of video with I-frames, P-frames, and B-frames. Squares represent data units and arrows indicate decoding order.

be transmitted at each of the next N transmission opportunities. The policy is understood to be contingent upon future acknowledgments that might arrive from the client to indicate that the packet has been received. No further transmissions of an acknowledged data unit l are attempted, even if π_l specifies a transmission for a future time slot.

Each transmission policy leads to its own *error probability,* $\varepsilon(\pi_l)$, defined as the probability that data unit l arrives at the client late, or not at all. Each policy is also associated with an expected number of times that the packet is transmitted under the policy, $\rho(\pi_l)$. The goal of the packet scheduler is to find a transmission policy π with the best trade-off between expected transmission rate and expected reconstruction distortion. At any transmission opportunity the optimal π minimizes the Lagrangian cost function

$$J(\pi) = D(\pi) + \lambda R(\pi), \tag{10.1}$$

where the expected transmission rate

$$R(\pi) = \sum_l \rho(\pi_l) B_l, \tag{10.2}$$

and the expected reconstruction distortion

$$D(\pi) = D_0 - \sum_l \Delta D_l \prod_{l' \preceq l} \big(1 - \varepsilon(\pi_{l'})\big). \tag{10.3}$$

The Lagrange multiplier λ controls the trade-off between rate and distortion. In (10.3) D_0 is the distortion if no data units arrive, ΔD_l is the distortion reduction if data unit l arrives on time and can be decoded, and the product term $\prod_{l' \preceq l} \big(1 - \varepsilon(\pi'_l)\big)$ is the probability for this to occur. The notation $l' \preceq l$ is shorthand for the set of data units that must be present to decode data unit l.

In the aforementioned formulation, delays and losses experienced by packets transmitted over the network are assumed to be statistically independent. Packet loss is typically modeled as Bernoulli with some probability, and the delay of arriving packets is often assumed to be a shifted-Γ distribution. Expressions for $\varepsilon(\pi_l)$ and $\rho(\pi_l)$ can be derived in terms of the Bernoulli loss probabilities, the cumulative distribution functions for the Γ-distributed delays, the transmission policies and transmission histories, and the data units' arrival deadlines. These derivations are straightforward, but because the resulting expressions are cumbersome, they are omitted here.

The scheduler reoptimizes the entire policy π at each transmission opportunity to take into account new information since the previous transmission opportunity and then executes the optimal π for the current time. An exhaustive search to

find the optimal π is generally not tractable; the search space grows exponentially with the number of considered data units, M, and the length of the policy vector, N [14]. Even though rates and distortion reductions are assumed to be additive, the graph of packet dependencies leads to interactions, and an exhaustive search would have to consider all 2^{MN} possible policies. Chou and Miao's RaDiO framework [6] overcomes this problem by using conjugate direction search. Their *Iterative Sensitivity Adjustment (ISA)* algorithm minimizes (10.1) with respect to the policy π_l of one data unit while the transmission policies of other data units are held fixed. Data units' policies are optimized in round-robin order until the Lagrangian cost converges to a (local) minimum.

Rewritten in terms of the transmission policy of one data unit, (10.1), (10.2) and (10.3) become

$$J_l(\pi_l) = \varepsilon(\pi_l) + \lambda' \rho(\pi_l), \tag{10.4}$$

where $\lambda' = \frac{\lambda B_l}{S_l}$ incorporates the rate–distortion trade-off multiplier λ from (10.1), the data unit size B_l, and S_l, a term that expresses the sensitivity of the overall expected distortion to the error probability $\varepsilon(\pi_l)$ of data unit l. The sensitivity term represents the relative importance of a particular data unit and incorporates the error probabilities of the other data units that l depends on. The sensitivity S_l changes with each iteration of the ISA algorithm to take into account the optimized policy for the other data units.

Figure 10.2 demonstrates improved video streaming performance achieved with RaDiO. Luminance PSNR versus transmitted bit rate is plotted for streaming simulations using an H.263+ two-layer SNR scalable encoding of the *Foreman* sequence. The frame rate is 10 fps; a Group of Pictures (GOP) consists of one I-frame followed by nine P-frames. The encoded source rate is 32 kbps for the base layer alone and 69 kbps when the enhancement layer is added. The results are for a simulated channel in which packet losses occur independently with a loss rate of 20%, and packet delays are drawn as independent, shifted-Γ random variables with a mean delay of 50 ms and a standard deviation of 25 ms. Figure 10.2 plots PSNR versus transmitted bit rate for a heuristic, prioritized ARQ system and for R–D optimized system. In the ARQ system, the client requests retransmissions for packets that do not arrive by a time interval after they are expected, and the server transmits these requested packets with priority as long as the requested packet may still reach the client in time for playout. When the capacity of the channel falls below the source rate for the enhanced stream, the ARQ system sends only the base layer packets. Both the ARQ and the R–D optimized system use an initial preroll delay of 400 ms. By continuously optimizing its packet transmission choices, the optimized system makes use of the SNR and temporal scalability of the source encoding to finely tune the source rate to the available channel capacity, yielding substantial gains.

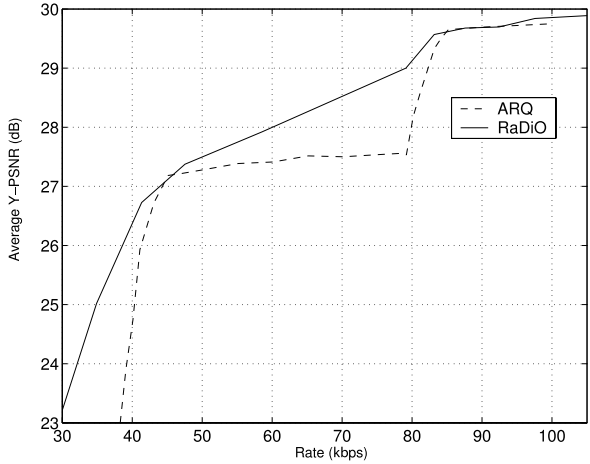


FIGURE 10.2: PSNR vs. transmitted bit rate for a video streaming system that uses heuristic deadline-constrained prioritized ARQ and for a system that uses RaDiO transmission scheduling. The results are for an H.263+ SNR scalable encoding [9] of the *Foreman* sequence.

Several techniques have been proposed to further reduce the complexity of the basic RaDiO algorithm. Chou and Sehgal [7] have presented simplified methods to compute approximately optimized policies. The framework appears to be robust against simplifications of the algorithm and approximations to ΔD_l, the information characterizing the value of individual packets with respect to reconstruction distortion. An attractive alternative to ISA is a randomized algorithm recently developed by Setton in which heuristically and randomly generated candidate policies are compared at each transmission opportunity [15,17]. The best policy from the previous transmission opportunity is one of the candidates and thus past computations are efficiently reused. With a performance similar to ISA, the randomized algorithm usually requires much less computation.

10.2.2 Receiver-Driven Streaming

When transmitting many audio and video streams simultaneously, a media server might become computation limited rather than bandwidth limited. It is therefore desirable to shift the computation required for network adaptive media transport from the server to the client to the extent possible. Fortunately, rate–distortion optimized streaming can be performed with the algorithm running at the client so that very little computation is required at the server [7].

For *receiver-driven* streaming, the client is provided information about the sizes, distortion reduction values, and interdependencies of the data units available at the server ahead of time. The size of this *hint track* or *rate–distortion preamble* is small relative to the media stream. The receiver uses this information to compute a sequence of requests that specify the data units that the server should transmit. It is straightforward to adapt the algorithm discussed in Section 10.2.1 to compute a sequence of requests that yield an optimal trade-off between the expected transmission rate of the media packets that the server will send and the expected reconstruction distortion that will result [7]. Figure 10.3 illustrates the differences between sender-driven and receiver-driven streaming.

By combining sender-driven and receiver-driven techniques, the RaDiO framework can be applied to diverse network topologies. For example, RaDiO might be implemented in a proxy server placed between the backbone network and a last hop link (Figure 10.4) [3]. The proxy coordinates the communication between the media server and the client using a hybrid of receiver- and sender-driven stream-

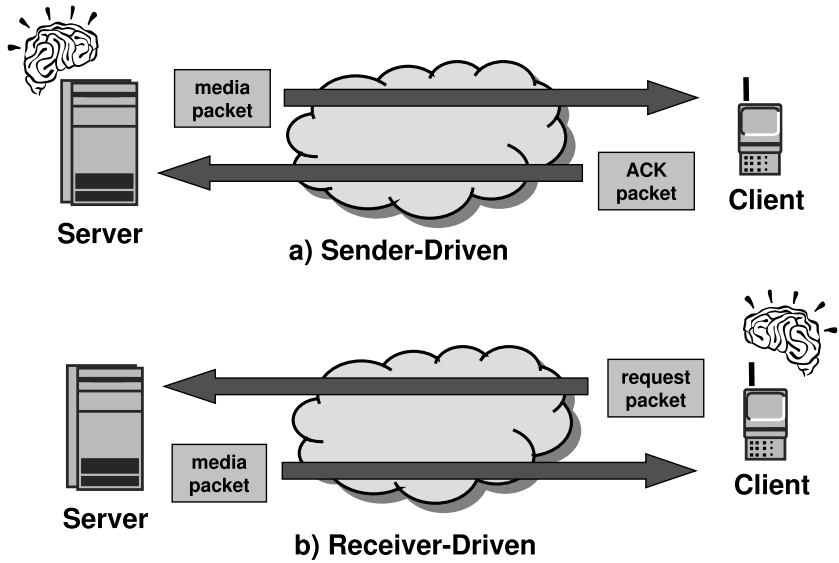


FIGURE 10.3: In sender-driven streaming (a), the server computes an optimal sequence of media packet transmissions and the client acknowledges packets upon receipt. In receiver-driven streaming (b), the complexity is shifted to the client. The client computes an R–D optimized sequence of requests to send to the server and the server only needs to respond to the client's requests.

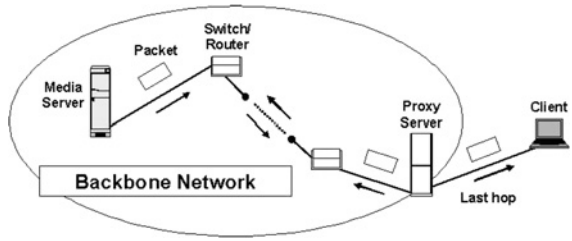


FIGURE 10.4: Proxy-driven RaDiO streaming. A proxy server located between the backbone network and a last hop link uses a hybrid of receiver- and sender-driven RaDiO streaming to jointly optimize requests to send to the server and media packets to forward to the client.

ing. End-to-end performance is improved compared to a sender- or receiver-driven RaDiO system because traffic created by retransmissions of media packets lost in the last hop to the client does not need to traverse and congest the backbone link.

10.3 RICH ACKNOWLEDGMENTS

In one extension to the RaDiO framework, streaming performance is improved through the use of *rich acknowledgments* [4]. In sender-driven RaDiO streaming using conventional acknowledgments, when a client receives a media packet, the client sends an acknowledgment packet (ACK) to the server. If the ACK packet is lost, the server may decide to unnecessarily retransmit the packet at the expense of other packets.

With rich acknowledgments, the client does not acknowledge each data unit separately. Instead, it periodically transmits a packet that positively acknowledges all packets that have arrived so far and negatively acknowledges (NACK) packets that have not yet arrived. A rich acknowledgment packet hence provides a snapshot of the state of the receiver buffer.

Rich acknowledgments require some changes to the basic RaDiO framework described in Section 10.2. As shown in [6], a transmission policy π_l for a data unit can be understood in terms of a Markov decision process. At discrete times t_i the server makes an observation o_i and then takes a transmission action a_i specifying *send* or *don't send*. Sequences of observation and action pairs (o_i, a_i) in time can be enumerated in a Markov decision tree. Each node q_i in the tree specifies a particular history of observations and actions $(a_0, o_0), (a_1, o_1), \ldots, (a_i, o_i)$. A transmission policy specifies what transmission action will be taken as a function of what state q_i is reached in the tree.

A Markov decision tree is shown in Figure 10.5. The tree enumerates the possible sequences of observation–action pairs for the transmission of a data unit using

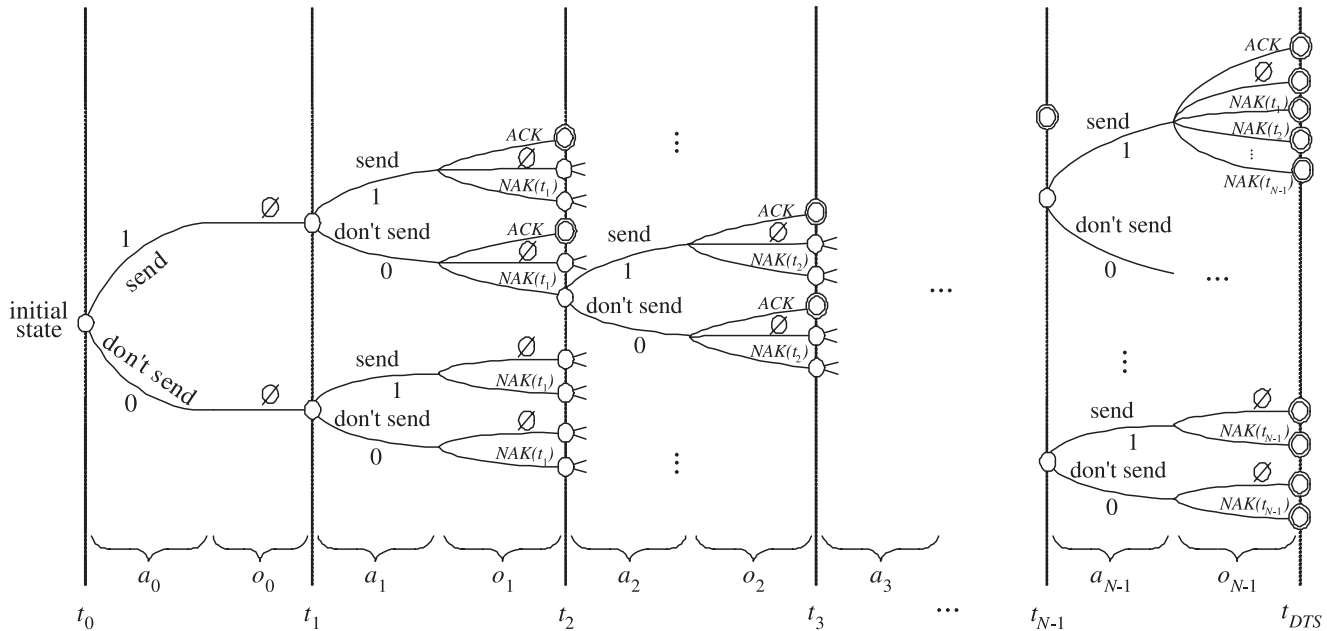


FIGURE 10.5: State space for the Markov decision process when *rich acknowledgments* are used.

the rich acknowledgment scheme. In the tree, possible actions a are *send* or *don't send*. Possible observations o are ($\emptyset$), no relevant feedback has arrived, *ACK*, a feedback packet has acknowledged the reception of the data unit, or *NACK*, a feedback packet has indicated that the packet has not been received by the feedback packet's send time. NACKs with different time stamps are distinct observations. In contrast, in the conventional feedback scheme in which each packet is acknowledged individually upon receipt, there are only two possible observations, *ACK* and $\emptyset$. Regardless of the scheme, the optimization algorithm calculates the probabilities of each path through the tree given a policy and then chooses the policy that yields the best trade-off between expected number of transmissions $\rho(\pi_l)$ and loss probability $\varepsilon(\pi_l)$.

Figure 10.6 compares average PSNR versus transmitted bit rate for the 13-s *Foreman* sequence streamed using the rich feedback scheme and using the conventional acknowledgment scheme. Two-layer SNR-scalable H.263+ is used for the encoding [9]. The bit rate of the base layer alone is 32 kbps with an average PSNR of 27 dB. When the enhancement layer is added, the encoded rate becomes 69 kbps with an average PSNR of 30.5 dB. The results are for simulation experiments with a 10% loss rate for both media packets and feedback packets. Delays for packets not lost were distributed according to independent shifted-Γ distributions with shift $\kappa = 50$ ms, mean $\mu = 25$ ms, and standard deviation $\sigma = 35$ ms.

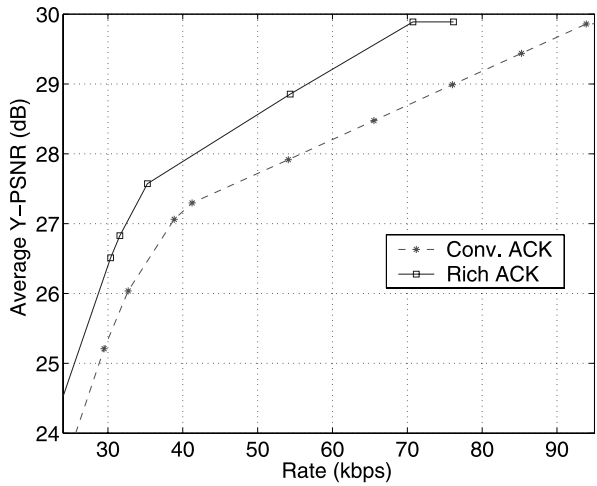


FIGURE 10.6: Rich vs. conventional acknowledgments for rate–distortion optimized streaming of QCIF Foreman.

In Figure 10.6, the rich acknowledgment scheme outperforms the RaDiO scheme with conventional ACKs for all transmission rates. The maximum PSNR improvement is 1.3 dB at a transmitted bit rate of 70 kbps. The improved performance of the rich acknowledgment scheme is due to the robust transmission of the feedback information. With rich acknowledgments, the effect of a lost feedback packet is mitigated because subsequent feedback packets contain the same (updated) information. In addition, because rich acknowledgment packets also provide NACKs, there is less ambiguity for the server to resolve. In the case of conventional feedback, a nonacknowledged transmission may be due to a lost media packet or to a lost acknowledgment packet.

10.4 MULTIPLE DEADLINES

In the RaDiO framework described in Section 10.2, ΔD_l is the expected distortion reduction if data unit l is decodable by its deadline. It was assumed that a data unit l must arrive by its specific deadline in order for its distortion reduction ΔD_l to be realized and in order for data units dependent on l to be decoded. Late data units are discarded. Often, however, a data unit arriving after its deadline is still useful for decoding.

As an example, consider the case of bidirectional prediction with a sequence of frames I-B-B-B-P. In the RaDiO framework, the deadline for the P-frame would be determined by the decoding time of the first B-frame. If the P-frame arrives later, however, it should not be discarded. It may still be useful for decoding subsequent B-frames or at least for decoding and displaying the P-frame itself. Thus there are several deadlines associated with the P-frame, each with its own associated distortion reduction [18].

Another example where a data unit may be associated with multiple deadlines is the case of decoders that allow *Accelerated Retroactive Decoding* (ARD). This idea was initially proposed in the context of MPEG-2 transmission over ATM [8]. ARD makes use of the ability of many streaming clients to decode video faster than real time. With ARD, when late-arriving data units finally do arrive, the decoder goes back to the frames corresponding to the late-arriving packets and quickly again decodes the dependency chain up to the current playout time, but now without error. In this way the remaining pictures in the GOP can be decoded and displayed without degradation.

As shown in [12], the introduction of multiple deadlines results in changes to expressions used to calculate expected distortion $D(\pi)$ for R–D optimized streaming. For each data unit, there is no longer a single error probability, but a set of them, one for each of the frame deadlines associated with that data unit. This results in changes to (10.4) that express the Lagrangian cost (10.1) as a function of only the transmission policy of one data unit π_l while other policies are

held fixed. With multiple deadlines, the expression in (10.4) becomes

$$J_l(\pi_l) = \rho(\pi_l) + \sum_{i \in \mathcal{W}_l} \nu_{t_i} \varepsilon(\pi_l, t_i), \tag{10.5}$$

where $\mathcal{W}_l$ is the set of frames that require data unit l for decoding, i is the frame index, and t_i is the decoding deadline for frame i. The quantity $\varepsilon(\pi_l, t_i)$ is the probability that data unit l does not arrive by deadline t_i. As before, $\rho(\pi_l)$ is the expected number of times data unit l is transmitted under policy π_l.

In (10.5) the quantity ν_{t_i}, given by $\nu_{t_i} = \frac{S_{l,t_i}}{\lambda B_l}$, is analogous to the reciprocal of λ' in (10.4). Note that the sensitivity term S_{l,t_i} is also indexed by the deadline. It is the sensitivity of the overall distortion to the arrival of data unit l by deadline t_i.

Figure 10.7 shows performance gains due to the multiple deadline formulation in the case when ARD is implemented in a streaming video client. PSNR-versus-rate results are shown for the *Foreman* sequence streamed in a low-latency application in which the preroll delay is 100 ms. In the simulation experiments, frames of video were due for decoding 100 ms after they became available for

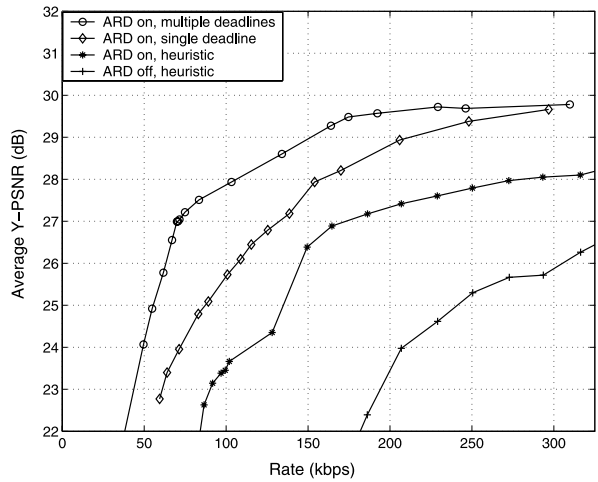


FIGURE 10.7: Rate–distortion performance of schedulers for the Foreman sequence streamed over a simulated channel with iid shifted-Γ-distributed packet delays and 20% Bernoulli loss. End-to-end latency $d = 100$ ms. A PSNR improvement of up to 3.15 dB is observed for the optimizing scheduler that considers multiple deadlines compared to the one that considers a single deadline.

transmission at the server. The packet loss rate was 20% in both directions, and delays for packets not lost were independent, identically distributed (iid) shifted-Γ with shift $\kappa = 10$ ms, mean $\mu = 40$ ms, and standard deviation $\sigma = 23$ ms. The sequence was encoded using a two-layer SNR-scalable H.263+ [9], at 10 fps with prediction structure I-P-P-P ... and GOP length of 20 frames. The base and enhancement layer bit rates and PSNRs were similar to those of the sequence in Section 10.3.

In Figure 10.7, PSNR-versus-rate curves compare the multiple-deadline schemes and the single-deadline scheme, as well as a heuristic scheme. The heuristic scheme uses prioritized, deadline-limited ARQ in which base layer retransmissions had highest priority, followed by base layer transmissions, enhancement layer transmission, and enhancement layer retransmissions. Retransmissions were triggered when packets were not acknowledged within the 0.90 point of the cumulative distribution function of the round-trip time. Figure 10.7 shows that the multiple-deadline formulation yields up to a 3-dB improvement over a single deadline. The single-deadline scheme does not recognize the utility of late packets and often misses opportunities to schedule valuable data units close to, or after, their original deadlines. The R–D optimizing schemes outperform the heuristic schemes regardless of whether the heuristic schemes are used with ARD-enabled clients.

10.5 DEPENDENT PACKET DELAYS

In the R–D optimized streaming algorithms discussed in Sections 10.2, 10.3, and 10.4, the delays of successive packets have been modeled as iid shifted-Γ random variables with loss also occurring independently as described in [6]. The iid model simplifies calculations for $\varepsilon(\pi_l)$, the error probability due to a transmission policy, and for $\rho(\pi_l)$, the expected number of transmissions that will result from a transmission policy. It fails to capture the dependence among delays, however. In the Internet, successive packets usually travel along the same path, might experience a similar backlog while waiting in the same queues, and rarely arrive out of order. This results in strongly dependent delays of successive packets.

In streaming simulations that employ measured Internet delay traces, we have observed that the iid model can lead to suboptimal scheduling performance. For example, Figure 10.8 shows simulation results when packets were delayed according to a delay trace measured over a 14-hop Internet path with a cable modem last hop, as described in [10]. At transmission rates above 80 kbps, the multideadline R–D optimizing formulation described in Section 10.4 is outperformed by the simple heuristic ARQ scheme (also described in Section 10.4). The suboptimal performance at high rates is due to the iid delay model assumed by the R–D optimization algorithm. With the iid model, policies that specify repeated

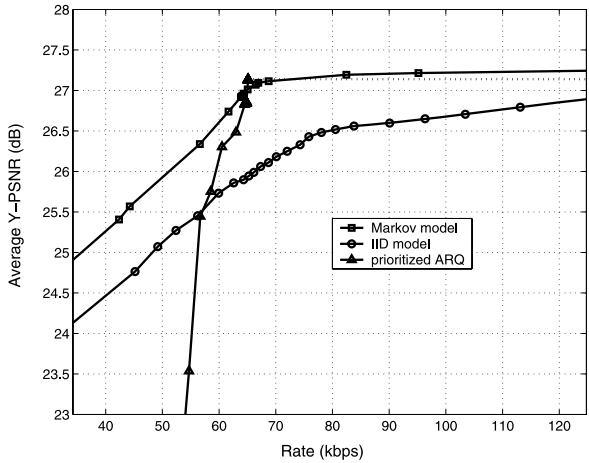


FIGURE 10.8: Rate–distortion performance of schedulers for the *Foreman* sequence streamed over a measured Internet delay trace. End-to-end latency $d = 150$ ms. The RaDiO scheduler that models delays as iid is suboptimal at high rates where it is outperformed by a heuristic-prioritized ARQ scheduler. The scheduler that models delays as a first-order Markov random process yields PSNR improvement of up to 1.1 dB over the iid scheduler.

transmission of a data unit at successive opportunities yield lower calculated error probabilities for errors due to late loss. The algorithm mistakenly believes that if the data unit is delayed the first time it is transmitted, subsequent transmissions may arrive earlier and on time. Thus at higher rates, the algorithm sends packets multiple times even though in our measured trace the loss probability is very low (0.014%) and packets always arrive in the order they are transmitted.

Rate–distortion performance can be improved by modeling packet delays at successive transmission time slots as a first-order Markov random process [10]. In [11] we have presented an R–D optimization scheme that uses this model. In the scheme, feedback packets inform the server about the delay over the channel in the recent past. Using this feedback and a family of conditional delay distributions, the scheme can more accurately calculate the expected distortion $D(\pi)$ and the expected transmission rate $R(\pi)$ resulting from a transmission policy π.

Figure 10.8 shows that the RaDiO scheme using the Markov channel model outperforms the RaDiO scheduler that uses iid delay modeling by up to 1.1 dB and is not outperformed by the heuristic scheduler at low rates. We note that the mean PSNR for all experiments is limited because the delays in the 14-hop cable

modem trace are often greater than the 150-ms preroll. Because the client uses the ARD scheme discussed in Section 10.4 and because the packet loss rate is nearly zero, the heuristic scheme, which uses time-out triggered retransmissions with the time-out set to $2 \cdot$ (estimated RTT), performs nearly optimally at high transmission bit rates.

10.6 CONGESTION–DISTORTION OPTIMIZED SCHEDULING

RaDiO streaming and its various extensions described do not consider the effect that transmitted media packets may have on the delay of subsequently transmitted packets. Delay is modeled as a random variable with a parameterized distribution; parameters are adapted slowly according to feedback information. In the case when the media stream is transmitted at a rate that is negligible compared to the minimum link speed on the path from server to client, this may be an acceptable model. In the case where there is a bottleneck link on the path from server to client, however, packet delays can be strongly affected by *self-congestion* resulting from previous transmissions.

In [16] a congestion–distortion optimized (CoDiO) algorithm is proposed, which takes into account the effect of transmitted packets on delay. The scheme is intended to achieve an R–D performance similar to RaDiO streaming but specifically schedules packet transmissions in a way that yields an optimal trade-off between reconstruction distortion and congestion, measured as average delay, on the bottleneck link. As with RaDiO, transmission actions are chosen at discrete transmission opportunities by finding an optimal policy over a time horizon. However, in CoDiO, the optimal policy minimizes the Lagrangian cost $D + \lambda\Delta$, where D is the expected distortion due to the policy and Δ is the expected end-to-end delay, which measures congestion.

CoDiO's channel model assumes a succession of high-bandwidth links shared by many users, followed by a bottleneck last hop used only by the media stream under consideration. CoDiO needs to know the capacity of the bottleneck, which can be estimated, for example, by transmitting back-to-back packets [13]. The overall delay experienced by packets is captured by a gamma pdf that is dynamically shifted by an extra delay that models the self-inflicted backlog at the bottleneck. Since the bottleneck is not shared and its capacity is known, the backlog can be accurately estimated. This channel model is used to calculate the expected distortion D due to packet loss and the expected end-to-end delay Δ.

The performance of the CoDiO scheme is illustrated using ns-2 simulation experiments [1,16]. The first hop is a high-bandwidth 45-Mbps link with 22-Mbps exponential cross-traffic. The second hop is a 50-kbps link that carries only the video traffic to be scheduled. The video encoding used is the same as that described in Section 10.4. The preroll delay for the experiments is 600 ms. Figure 10.9 plots luminance PSNR versus average end-to-end delay for the CoDiO

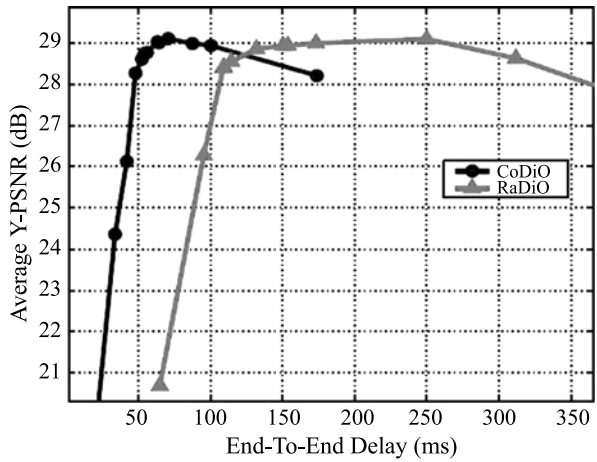


FIGURE 10.9: Performance comparison of RaDiO and CoDiO for video streaming over a bottleneck link. The horizontal axis shows the expected end-to-end delay due to the congestion caused by the video traffic. CoDiO causes much less congestion than RaDiO at the same PSNR. From [16].

and the RaDiO schemes. The various points on the curves were generated by varying λ, which trades-off congestion–distortion in the case of CoDiO and rate distortion in the case of RaDiO. The graphs show that the CoDiO scheme resulted in end-to-end delays that were approximately half of those measured for the RaDiO scheme at the same PSNR. Transmission rates versus PSNR for both schemes are almost identical (Figure 10.10).

CoDiO outperforms RaDiO because it distributes transmissions in time and attempts to send packets as late as safely possible. This reduces the backlog in the bottleneck queue and hence the average end-to-end delay. Other applications sharing the network experience less congestion. RaDiO, however, is less network-friendly. As the scheduler only considers average rate, its traffic tends to be more burst, relying more on the buffering in the network itself.

10.7 SUMMARY AND FURTHER READING

In this chapter we have discussed network adaptive media transport through the RaDiO framework for rate distortion optimized media streaming. After reviewing the basic framework as initially presented by Chou and Miao, we considered extensions and enhancements that have been proposed. The framework can be implemented in a media server or, alternatively, at the client. Rich acknowledgments

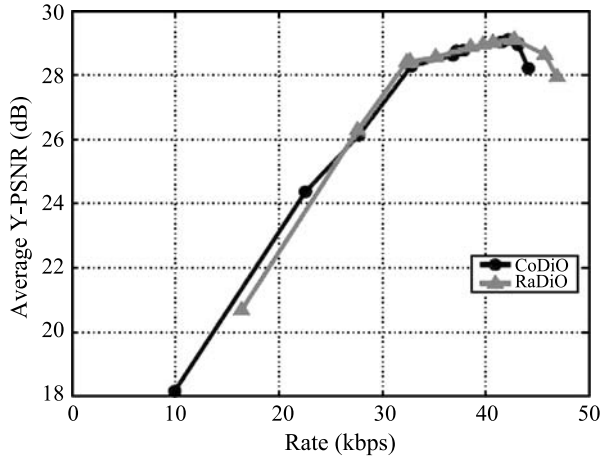


FIGURE 10.10: Rate–distortion performance of RaDiO and CoDiO for video streaming over a bottleneck link. From [16].

are an easy way to improve resilience against losses in the feedback channel. For video streaming, it is useful to incorporate multiple deadlines for packets. Considerable gains are possible by accelerated retroactive decoding of packets, particularly if a multiple-deadline scheduler knows about this client capability and schedules accordingly. RaDiO typically assumes independent packet delays, but, in fact, Internet packet delays are highly dependent. In Section 10.5, an extension of RaDiO streaming that utilizes a Markov model of successive packet delays has been shown to rectify the poor performance that arises due to its simple iid channel model. Finally, we have considered self-congestion that might arise with streaming over a bottleneck link. Congestion–distortion optimized streaming, CoDiO, yields the same PSNR performance as RaDiO, but reduces the congestion, measured in terms of end-to-end delay.

Readers with further interest should first study Chou and Miao's seminal paper [6] in depth. The paper is based on a longer technical report [5], so readers might want to consult this document as well. Interestingly, numerous papers appeared during the review period of [6], inspired by this work, many of which are now referenced in [6] itself. Various extensions and the most comprehensive experiments applying RaDiO to streaming of H.264/AVC encoded video can be found in Chakareski's dissertation [2]. The best reference for CoDiO and low-complexity randomized scheduling algorithms so far is Setton's dissertation [15]. There are numerous research groups active in the area, and their publications might be of interest to those following the evolving state-of-the-art in network adaptive media transport.

REFERENCES

[1] The Network Simulator – ns-2. *www.isi.edu/nsnam/ns/*.

[2] J. Chakareski. *Rate-Distortion Optimized Packet Scheduling for Video Streaming*. Ph.D. thesis, Rice University, Houston, TX, 2005.

[3] J. Chakareski, P. A. Chou, and B. Girod. Rate-distortion optimized streaming from the edge of the network. In *IEEE Workshop on Multimedia Signal Processing*, St. Thomas, U.S. Virgin Islands, December 2002.

[4] J. Chakareski and B. Girod. Rate-distortion optimized video streaming with rich acknowledgments. In *Proceedings SPIE Visual Communications and Image Processing VCIP-2004*, Santa Clara, CA, January 2004.

[5] P. A. Chou and Z. Miao. Rate-distortion optimized streaming of packetized media. Technical report MSR-TR-2001-35, Microsoft Research, Redmond, WA, 2001.

[6] P. A. Chou and Z. Miao. Rate-distortion optimized streaming of packetized media. *IEEE Transactions on Multimedia*, 8(2):390–404, April 2006.

[7] P. A. Chou and A. Sehgal. Rate-distortion optimized receiver-driven streaming over best-effort networks. In *Packet Video Workshop*, Pittsburgh, PA, April 2002.

[8] M. Ghanbari. Postprocessing of late cells for packet video. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(6):669–678, December 1996.

[9] ITU-T. Video coding for low bitrate communication: Recommendation H.263, Version 2, 1998.

[10] M. Kalman and B. Girod. Modeling the delays of successively-transmitted internet packets. In *IEEE Conference on Multimedia and Expo*, Taipei, Taiwan, June 2004.

[11] M. Kalman and B. Girod. Rate-distortion optimized video streaming using conditional packet delay distributions. In *IEEE Workshop on Multimedia Signal Processing*, Siena, Italy, September 2004.

[12] M. Kalman, P. Ramanathan, and B. Girod. Rate distortion optimized streaming with multiple deadlines. In *IEEE International Conference on Image Processing*, Barcelona, Spain, September 2003.

[13] V. Paxson. *Measurement and Analysis of End-to-End Internet Dynamics*. Ph.D. dissertation, UC Berkeley, Berkeley, CA, 1997.

[14] M. Podolsky, S. McCanne, and M. Vetterli. Soft ARQ for layered streaming media. Technical Report UCB/CSD-98-1024, University of California, Computer Science Department, Berkeley, CA, November 1998.

[15] E. Setton. *Congestion-Aware Video Streaming over Peer-to-Peer Networks*. Ph.D. dissertation, Stanford University, Electrical Engineering, 2006.

[16] E. Setton and B. Girod. Congestion-distortion optimized scheduling of video over a bottleneck link. In *IEEE Workshop on Multimedia Signal Processing*, Siena, Italy, September 2004.

[17] E. Setton, J. Noh, and B. Girod. Congestion-distortion optimized peer-to-peer video streaming. In *Proc. IEEE International Conference on Image Processing, ICIP-2006*, Atlanta, GA, October 2006.

[18] S. Wee, W. Tan, J. Apostolopoulos, and M. Etoh. Optimized video streaming for networks with varying delay. In *Proceedings of the IEEE International Conference on Multimedia and Expo 2002*, Lausanne, Switzerland, August 2002.

PART **D**

WIRELESS NETWORKING

This page intentionally left blank

11

Performance Modeling and Analysis over Medium Access Control Layer Wireless Channels

Syed Ali Khayam and Hayder Radha

11.1 INTRODUCTION

Wireless networks suffer from frequent bit errors due to their vulnerability to interference and transmission medium degradation. Errors not corrected by a wireless physical layer propagate to the medium access control (MAC) layer. Corrupted packets with bit errors are generally detected and dropped using a MAC layer checksum at a wireless receiver. Because users have little or no control over the hardware-based wireless physical layer, MAC layer bit errors constitute the higher layers' view of the wireless channel. Moreover, wireless standards generally adapt the physical layer to cater for new requirements, but the MAC and higher layers remain unchanged [1,2]. There has been significant research interest in analysis of wireless MAC layer packet losses and bit errors [3–9].

An accurate model of the MAC layer channel can render important insights into the underlying characteristics of an impairment (e.g., bit errors, packet losses) random process. This insight is essential for the design, performance evaluation, and parameter tuning of a wide range of wireless communication protocols, applications, and services. For instance:

- *Wireless congestion control protocols*, instead of relying on MAC layer retransmissions, can use accurate MAC layer error models to differentiate between losses due to congestion, medium degradation, or mobility. The

inability of wired congestion control algorithms to differentiate between different types of losses (and the consequent bandwidth underutilization) has been repeatedly highlighted by wireless studies. Knowledge of losses due to channel errors is assumed in many wireless congestion control solutions and such knowledge can only be rendered by real-time MAC layer channel models. Understanding of error frequency and burstiness is also instrumental in parameter tuning of wireless congestion control protocols.

- *Cross-layer protocols* can use a real-time channel model to choose whether to use MAC layer retransmissions or to ignore data payload errors according to different application requirements.
- *Reliable routing protocols* for mobile networks can use MAC layer channel models to differentiate (and ultimately choose) reliable versus shortest routes to different destinations, provided that the model is able to provide real-time error characterization at different hops of the network.
- *MAC protocols* can decide when to increase/decrease the physical transmission data rate based on real-time channel estimation. An accurate channel model can predict future error characteristics, thereby saving the MAC layer protocol the overhead of switching to an inaccurate lower/higher data rate based on short-term observations.
- Real-time channel estimation provided by an accurate model can be employed by *rate-adaptive applications* to perform channel- and/or source-coding rate adaptation for efficient utilization of scarce wireless bandwidth.
- Design of effective *error-control schemes* for different wireless applications requires a thorough understanding of errors above the physical layer.
- Design of *error-resilience features* of contemporary multimedia codecs can benefit greatly from knowledge of MAC layer error characteristics.

Most benefits of a wireless MAC layer channel model can be realized when the model is able to provide real-time and online channel characterization and prediction. In complexity- and power-constrained wireless and mobile environments such channel characterization is only possible with a low-complexity model.

11.2 MARKOV CHAINS FOR WIRELESS CHANNEL MODELING

Markov chains have shown remarkable promise in modeling of many wireless error and loss processes [3–9]. Therefore, throughout the text we focus on analyzing and modeling wireless errors using Markov chains.

11.2.1 Discrete-Time Markov Chains

Let us consider a discrete-time stochastic process, X_n, that transits between states denoted as integers from a finite set $H = \{0, 1, \ldots, N-1\}$. If $X_n = i$, then the

process is said to be in state i at discrete time instance n. Whenever the process is in state i, there is a fixed probability that the next state of the process will be j. If that probability can be expressed

$$\Pr\{X_{n+1} = j \mid X_n = i, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0\} = \Pr\{X_{n+1} = j \mid X_n = i\}, \tag{11.1}$$

for all states $j, i, i_{n-1}, \ldots, i_0 \in H$ and for all $n \geq 0$, then such a stochastic process is known as a *discrete-time Markov chain* and (11.1) is called the *Markov property*. Thus, for a Markov chain the conditional distribution of any future state X_{n+1}, given the past state sequence $X_0, X_1, \ldots, X_{n-1}$ and the present state X_n, is independent of the past states and depends only on the present state X_n. Let us define $p_{i,j} = \Pr\{X_{n+1} = j \mid X_n = i\}$ as the probability of transiting to state i from j. Since $p_{i,j}$ represents a probability measure, it exhibits the following properties: (a) $p_{i,j} \geq 0$ for all i, j and (b) $\sum_{j=0}^{N-1} p_{i,j} = 1$ for $i = 0, 1, \ldots, N-1$. The probability of transiting to the next state can be represented in a matrix form. This matrix is referred to as the one-step state transition probability matrix.

The steady-state or stationary probabilities of a Markov chain represent the long-run proportion of the time spent in each state. Once the transitional probabilities of a Markov chain are known, the steady-state probabilities of being in a particular state are the unique nonnegative solutions of the following linear system of equations:

$$\pi_j = \sum_{i=0}^{N-1} \pi_i p_{i,j}, \quad j = 0, 1, \ldots, N-1$$
$$\sum_{j=0}^{N-1} \pi_j = 1.$$

A detailed treatment of the theory of Markov chains may be found in [10] and [11].

11.2.2 Memory Length of a Random Process

Consider a random process X_n that assumes values from a binary alphabet. Let us generically refer to the outputs of the process as *bits* belonging to $\{0, 1\}$. Observing the outputs of the random process will result in a binary time series. In the present channel modeling context, X_n represents the bit error data comprising *good* and *bad* bits. Correlation of the binary time series can reveal the amount of temporal dependence in the series. Specifically, one can obtain a general sense of the number of previous bits on which a particular bit of the time series depends. The value of the temporal dependence is referred to as the *memory length* or *order* of the random process.

It has been observed by previous studies that wireless impairment processes generally have a memory length greater than one. One the other hand, if we consider each bit to be an output of the random process, then the Markov property in (11.1) implies that a process correlated with more than one bit in the past cannot be characterized as a Markov chain. An obvious question here is: If a binary wireless impairment process has a memory length greater than one, then how does one use a Markov chain to model that process? The answer to this question is rather straightforward. We define a high-order Markov chain where each output of the process contains as many bits as the memory length. The output at each step is then a fixed number of bits referred to as the memory window. Since the size of the memory window is constant, at each step a new bit is added to the memory window and the oldest bit is dropped from the memory window. If the process adds new bits in the least significant bit position of the memory window, then at each step the process updates a shift register by (i) shifting the register one bit to the left to eliminate the most significant or the oldest bit and (ii) adding a new bit to the least significant bit position.

11.2.3 High-Order Markov Chains for Wireless Bit Errors

For a Markov chain with memory length k, one can represent the states of the process by 2^k possible combinations of k consecutive bits. Transition probabilities of a kth order Markov chain (k-MC) for wireless modeling are generally estimated by sliding a k bit memory window (bit by bit) over the wireless traces and by observing the frequency of a bit pattern $\underset{\rightarrow}{x} = [x_1 x_2 \cdots x_k]$ followed by a bit pattern $\underset{\rightarrow}{y} = [y_1 y_2 \cdots y_k]$ for all patterns $\underset{\rightarrow}{x}$ and $\underset{\rightarrow}{y}$.

11.3 PRACTICAL ISSUES

This section discusses some practical issues that arise when developing a high-order Markov chain model of a wireless channel.

11.3.1 Determining the Memory Length of a Markov Chain

Determining the memory length of a Markov chain has been explored in prior texts [13,14]. Autocorrelation of a process is a simple indication of the maximum memory length of a Markov chain to model the process. Let $X(n_1)$ and $X(n_2)$ be two random variables derived from a random process X_n. The sample correlation of these random variables is defined as [12]

$$\gamma(n_1, n_2) = \mathrm{E}\{X(n_1)X(n_2)\}, \tag{11.2}$$

where $\mathrm{E}\{X\}$ represents the sample mean of the random variable X. Since both $X(n_1)$ and $X(n_2)$ are derived from realizations of the same random process, the correlation is often referred to as *autocorrelation*. Let n_1 and n_2 be separated in time by a lag η such that $n_1 = 0$ and $n_2 = n_1 + \eta = \eta$. In this case, the autocorrelation becomes a function of the lag η and a sample autocorrelation coefficient can be derived from (11.2) as

$$\rho(\eta) = \frac{\mathrm{E}\{X(0)X(\eta)\} - \mathrm{E}\{X(0)\}\mathrm{E}\{X(\eta)\}}{\sigma_{X(0)}\sigma_{X(\eta)}}, \tag{11.3}$$

where σ_X represents the sample standard deviation of the random variable X. The sample autocorrelation function, when computed for different values of the lag, is a direct metric for the level of temporal dependence in the random process. Since there is a one-to-one correlation between the random variables at lag zero, the autocorrelation has its maximum value of one at this point. For a large range of statistical data, autocorrelation between two random variables decreases rapidly with an increase in the lag between them. Lag beyond which the autocorrelation coefficient drops to an insignificant value represents the memory length of the process. In slightly relaxed jargon, memory length represents the lag beyond which the random variables of a random process are uncorrelated.

11.3.2 Determining the Accuracy of a Wireless Channel Model

The accuracy of a wireless channel is generally quantified by synthesizing impairment traces from the model. The model-based traces are compared with traces collected over the channel to ascertain how closely the model is approximating the actual source. Such accuracy evaluation necessitates an appropriate statistical measure. We describe two such measures here.

11.3.2.1 Standard Error Between Cumulative Distributions

Let $p(X)$ and $q(X)$ be two probability mass functions (PMFs) of a random variable X defined over an alphabet set Ψ. Let $P(X)$ and $Q(X)$ denote the cumulative distribution functions (CDFs) of $p(X)$ and $q(X)$. The standard error between $P(X)$ and $Q(X)$ is then defined

$$S_{\mathrm{err}}\big(P(X), Q(X)\big) = \sqrt{\frac{1}{n(n-2)}\left(n\sum_{\Psi}\big(P(X)\big)^2 - \left(\sum_{\Psi}P(X)\right)^2 - \frac{\big(n\sum_{\Psi}P(X)Q(X) - \big(\sum_{\Psi}P(X)\big)\big(\sum_{\Psi}Q(X)\big)\big)^2}{n\sum_{\Psi}(Q(X))^2 - \big(\sum_{\Psi}Q(X)\big)^2}\right)}, \tag{11.4}$$

where n is the length of the CDF. The random variable X should capture a key statistic of the random process. Assume that $P(X)$ is a CDF provided by the actual random source (e.g., the MAC layer bit error process) and $Q(X)$ is a CDF provided by a model that approximates the random source. Then the standard error provides a measure of the error incurred by the model in approximating the actual source. Small values of S_{err} imply that a model is a good approximate of the actual source.

11.3.2.2 Entropy Normalized Kullback–Leibler Divergence

From a source coding perspective, entropy provides a measure of the average number of bits required to represent a source completely. For the random variable X defined earlier, entropy provides a weighted average of the minimum information of X, where *information* corresponds to the number of bits required to uniquely represent all possible outcomes of a random variable. Entropy is expressed as

$$H\big(p(X)\big) = -\sum_{X\in\Psi} p(X)\log\big(p(X)\big). \tag{11.5}$$

The Kullback–Leibler divergence [15] renders a measure of the statistical divergence between $p(X)$ and $q(X)$ as

$$D\big(p(X)\|q(X)\big) = \sum_{X\in\Psi} p(X)\log\big(p(X)/q(X)\big). \tag{11.6}$$

The Kullback–Leibler divergence provides a nonnegative statistical divergence measure, which is zero if and only if $p = q$ [15]. When a base-2 logarithmic measure is used, the Kullback–Leibler divergence gives the number of overhead bits incurred because a model (represented by $q(X)$) is used instead of the actual source (represented by $p(X)$).

In order to accurately judge the performance of a model, the Kullback–Leibler measure should be weighted with respect to the entropy. For example, let us assume that the entropy of the source is 20 bits whereas the overhead incurred by the model is 0.75 bits. The overhead is relatively insignificant since the source inherently requires a large number of bits to be represented. However, for the same overhead (of 0.75 bits), if the entropy of the source is low, say 1 bit, then an overhead of 0.75 bits is extremely high. Hence, for accurate performance evaluation of a model, both the entropy of the process (represented by some random variable X) and the Kullback–Leibler divergence should be taken into consideration.

In view of the aforementioned considerations, a new statistical divergence measure, the *entropy normalized Kullback–Leibler (ENK) divergence*, was proposed

in [8]. The ENK divergence renders a measure of the source-coding-like overhead incurred by employing a model instead of the actual random source. The ENK divergence is defined as

$$ENK\big(p(X)\|q(X)\big) = \frac{D(p(X)\|q(X))}{H(p(X))}, \tag{11.7}$$

where $D(p\|q)$ and $H(p)$ are defined in (11.6) and (11.5), respectively. A closer examination reveals that the ENK measure inherits basic properties of the Kullback–Leibler divergence: (i) nonnegativity, $ENK(p\|q) \geq 0$, (ii) nonsymmetry, $ENK(p\|q) \neq ENK(q\|p)$, and (iii) $ENK(p\|q) = 0 \Leftrightarrow p = q$.

Small values of ENK divergence indicate that the model renders a good approximate of the actual random source. Conversely, large values of the ENK imply that the source-coding-like overhead of the model is large, that is, the model is not a good approximate of the actual source. Note that we would expect the ENK between two realizations of a random source to be a small value. For instance, the ENK between two traces collected over a wireless medium under similar conditions should be quite small. This ENK value can be used as an evaluation reference for the ENK between the actual observations (i.e., traces collected over the wireless network) and the model-based observations (i.e., traces artificially synthesized by the model).

11.3.2.3 Random Variables for Performance Comparison

The performance evaluation measures described in preceding discussions are dependent on the choice of an appropriate random variable X to represent the stochastic process. Hence, X should capture a key statistic of the bit error random process. Wireless channel modeling studies generally employ two random variables to evaluate the performance of a model: (i) burst length of correctly received bits, referred to as *good bursts*; and (ii) burst length of bit errors, referred to as *bad bursts*. Both these random variables assume nonzero positive integer values. The burstiness of the bit error process is adequately characterized using these two random variables.

11.4 MODELING 802.11B BIT ERROR PROCESSES USING MARKOV CHAINS

Let us now model the bit errors at 2- and 5.5-Mbps data rates of an 802.11b local area network (LAN) [1,2] using kth order Markov chains (k-MCs). The traces used for results in this section were collected by positioning the wireless sender (server) and receivers (clients) in separate rooms to simulate a realistic business/classroom/home-network wireless setup.

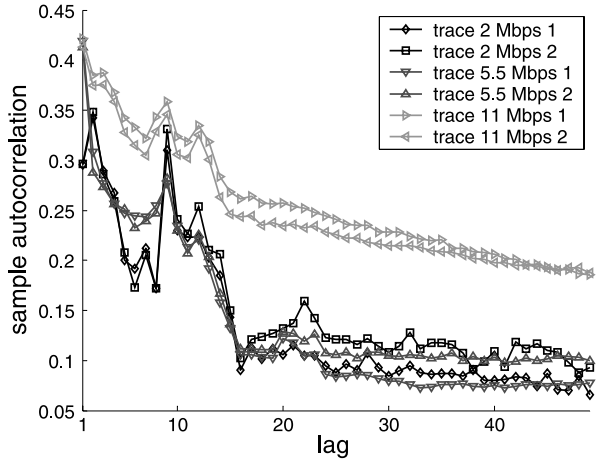


FIGURE 11.1: Autocorrelation of bit error traces.

11.4.1 Autocorrelation of the Bit Error Traces

The sample autocorrelation coefficients of six traces collected at 2, 5.5, and 11 Mbps are illustrated in Figure 11.1. The autocorrelations at 2 and 5.5 Mbps exhibit a rapidly decaying trend as the level of temporal dependence decreases with time. From the examples provided in Figure 11.1, we assume that the memory length is determined by the lag beyond which the correlation is less than 0.15, an empirically determined threshold. Based on the threshold of 0.15, the maximum memory lengths of the 5.5-Mbps traces of Figure 11.1 are 12 and 14, respectively. The correlation of both 2-Mbps traces drops below 0.15 at the lag of 16. Thus, for the 5.5- and 2-Mbps bit error processes, maximum memory lengths of 14 and 16 are identified, respectively. (We observe later on that a bit-error process might be adequately characterized by a memory length that is slightly less than the maximum memory length identified here.)

Note that the correlation at 11 Mbps is really high even at large lags. Thus, the 11-Mbps bit error process has substantial memory. A k-MC model for such a highly correlated process will be unreasonably complex. Some models (e.g., the ones proposed in [16]) can be used to model the highly correlated 11-Mbps process. These models are out of the scope of this text.

11.4.2 Markov Chains for the 2-Mbps Bit Error Process

For both ENK and standard error performance evaluations, varying order Markov chains were trained using actual 2-Mbps bit error traces. The trained models

were then used to synthesize artificial traces. The good- and bad-burst PMFs and CDFs were derived from both synthesized and actual traces. The ENK divergence between two actual traces is used as reference to quantify the performances of varying order Markov chains. Specifically, first the ENK divergence is computed by deriving $p(X)$ and $q(X)$ of (11.7) from two actual traces, say trace 1 and trace 2. This ENK value is used as a performance evaluation reference for Markov chains. To evaluate a Markov chain's accuracy, ENK divergence is computed by deriving $p(X)$ from trace 1 and $q(X)$ from a synthesized trace generated by the Markov chain. Similarly, for standard error-based performance evaluation, first $P(X)$ and $Q(X)$ of (11.4) are derived from trace 1 and trace 2, respectively. This value of standard error is used as a performance evaluation reference for Markov chains. To evaluate the accuracy of a Markov chain, standard error is computed by deriving $P(X)$ from trace 1 and $Q(X)$ from the synthesized trace generated by the Markov chain. Good- and bad-burst random variables are used for both ENK- and standard error-based performance evaluations.

The ENK-based performances of varying order Markov chains in modeling of the 2-Mbps bit error process are depicted in Figure 11.2. It is clear from Figure 11.2 that low-order Markov chains incur significant ENK overhead. Hence, low-order Markov chains cannot capture the 2-Mbps bit error behavior effectively. Nevertheless, as the order of the Markov chain increases, the ENK overhead decreases substantially and drops to a reasonable level. The accuracy of the order-10 Markov chain is comparable to the ENK divergence between two actual bit error traces. Figure 11.2 only provides analysis up to order 10, as the performance improvement saturates after the order-10 (1024 state) Markov chain. It can be observed that the 1024-state Markov chain renders a good approximate of the 2-Mbps MAC layer bit error process.

The standard errors of the good- and bad-burst CDFs rendered by the 1024-state Markov chain of the 2-Mbps process are given in Table 11.1. For comparison, standard errors of good- and bad-burst CDFs derived from two actual 2-Mbps traces are also given in Table 11.1. For both good- and bad-burst random variables, standard errors of the 1024-state Markov chain are very close to the standard errors of two actual traces. Thus, it can be concluded that a 1024-state Markov chain can adequately capture the bit error statistics of the 2-Mbps process.

11.4.3 Markov Chains for the 5.5-Mbps Bit Error Process

Performance of Markov chain models in modeling of the 5.5-Mbps bit error process is provided in Figure 11.3. High-order Markov chains perform remarkably well for the bad-burst random variable. Note that for the bad-burst random variable even smaller order chains perform quite adequately with low ENK

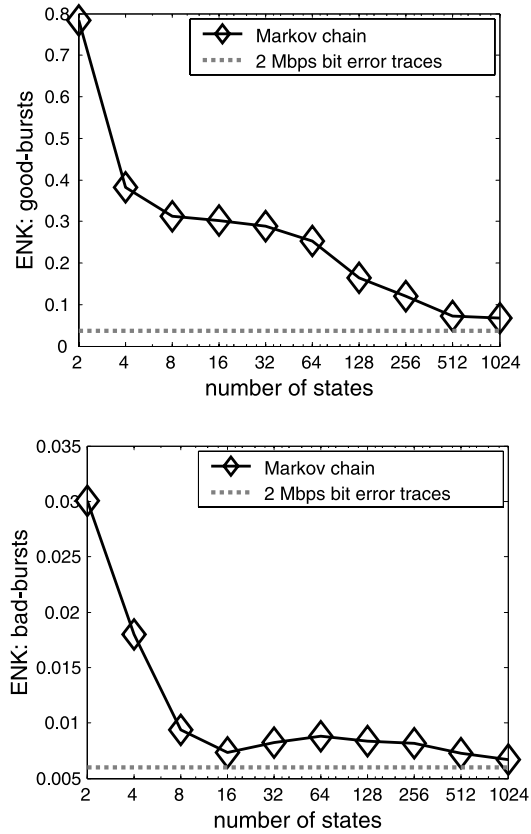


FIGURE 11.2: ENK-based modeling accuracy of varying order Markov chains for the 2-Mbps MAC layer bit error process: (top) good bursts and (bottom) bad bursts.

Table 11.1: Standard error of Markov chain-based cumulative densities at 2 and 5.5 Mbps.

	2 Mbps		5.5 Mbps	
	S_{err} between two actual traces	S_{err} between an actual trace and a trace synthesized by a 1024-state Markov chain	S_{err} between two actual traces	S_{err} between an actual trace and a trace synthesized by a 512-state Markov chain
Good bursts	0.003	0.014	0.004	0.006
Bad bursts	0.002	0.008	0.004	0.004

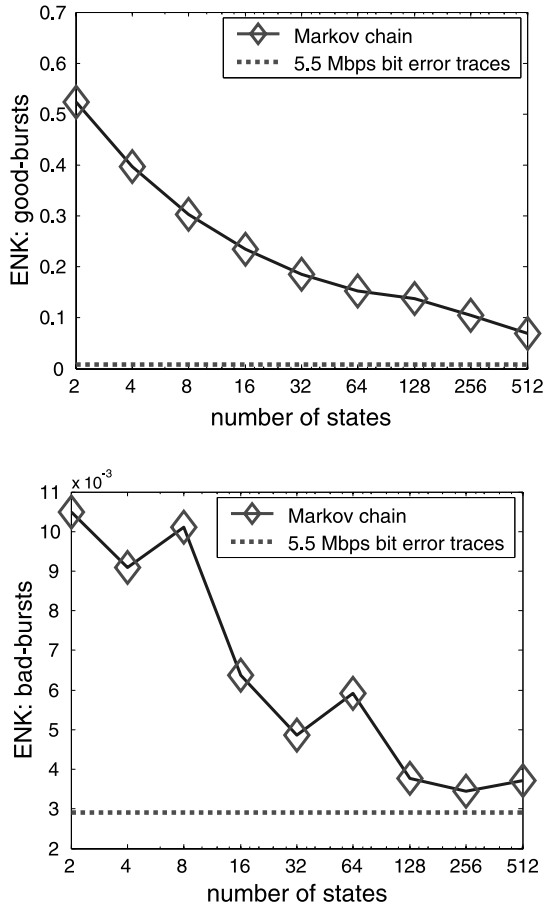


FIGURE 11.3: ENK-based modeling accuracy of varying order Markov chains for the 5.5-Mbps MAC layer bit error process: (top) good bursts and (bottom) bad bursts.

overhead for all cases. However, the good-burst random variable incurs significant overhead for low-order chains. For high-order chains, the overhead decreases and drops to a reasonable level, beginning at the order 9 (512 state) model.

The standard errors of good- and bad-burst CDFs at 5.5 Mbps are tabulated in Table 11.1. Clearly, the 512-state Markov chain provides standard errors that are comparable to the standard errors between actual 5.5-Mbps traces. Thus, a 512-state Markov chain can model the 5.5-Mbps process accurately.

11.5 REDUCING THE COMPLEXITY OF MARKOV CHAINS

The number of states in a k-MC is an exponential function of the memory length—2^k states for a process with memory-length k. This phenomenon, referred to as *state explosion*, constrains the applicability of a k-MC. For instance, a process with a memory-length 10 will result in $2^{10} = 1024$ states. Due to their high complexity, k-MCs cannot provide real-time channel characterization in resource-constrained wireless environments. Thus, despite their accuracy, the exponential complexity of Markov chains hampers their deployment in complexity-constrained wireless environments.

A natural alternative to reduce Markov chain complexity is to employ hidden Markov models (HMMs) [22]. However, in problem areas where HMMs are successful, well-defined characteristics of input data are available for preprocessing and training, for example, cepstral and linear-prediction features in speech. For wireless channel modeling, the corrupted trace regions exhibit highly random behavior, and it has been shown that simple features (e.g., energy) are not adequate to characterize error patterns in these corrupted regions [8]. Moreover, HMMs assume that the probability of staying in a state is distributed exponentially, which may not be an accurate assumption on practical wireless channels. An HMM trained using the exponential distribution assumption results in inaccurate parameterization, consequently leading to a model that is unable to approximate the bit error process [8].

Some studies have tried alternative approaches to approximate wireless bit error behavior [4–20]. This section outlines one such approach to reducing the complexity of Markov chains.

11.5.1 Observations About Markov Chains

Let us first state some important observations about k-MCs. These observations are employed in subsequent sections to derive properties of k-MCs. The first observation is a direct consequence of the binary nature of wireless impairment processes.

Observation 1. *For a binary process, if a bit-by-bit sliding window is used to compute the transition probabilities of a 2^k state Markov chain, then from a current state, $X_n = i$, in one transition the Markov chain can transit to only two possible states given by*

$$X_{n+1} = \begin{cases} (2i) \bmod 2^k, \\ (2i+1) \bmod 2^k, \end{cases} \tag{11.8}$$

where k is the memory length of the Markov chain and $i \in \{0, 1, \ldots, 2^k - 1\}$ is an arbitrary state from the Markov state space.

An example given in Figure 11.4 clearly demonstrates this observation. A memory-length $k = 4$ is used in this example. The set of all possible Markov states is $\{0, 1, 2, \ldots, 2^4 - 1 = 15\}$. The current state is $X_n = (0110)_2 = (6)_{10}$ and, as the window slides by one bit, the 0 in the most significant bit position will be dropped and a bit will be added to the least significant bit position. Because the data are binary, the chain can transit to either $(1100)_2 = (12)_{10}$ or $(1101)_2 = (13)_{10}$. Thus, in essence, Observation 1 implies that at each slide of the memory window the process' current state i is subjected to three operations: left shift by one bit, which yields $2i$, followed by an addition of a zero $(2i + 0 = 2i)$ or an addition of a one $(2i + 1)$ at the least significant bit position (LSB), followed by a modulus operation that ensures that if the current state of the process is $X_n = 2^{k-1}$, then the next state wraps around to state 0 (for $X_{n+1} = 2i$) or state 1 (for $X_{n+1} = 2i + 1$). For instance, in the preceding example with $k = 4$, if $X_n = (1000)_2 = (8)_{10} = 2^{k-1}$, then the next state will be either $X_{n+1} = (2 \times 8) \bmod 2^4 = (0)_{10} = (0000)_2$ or $X_{n+1} = (2 \times 8 + 1) \bmod 2^4 = (1)_{10} = (0001)_2$. Since each Markov state has two transition possibilities, each row of the Markov transition probability matrix will have at most two nonzero entries, given by $p_{i,(2i) \bmod 2^k}$ and $p_{i,(2i+1) \bmod 2^k} = 1 - p_{i,(2i) \bmod 2^k}$.

Intuitively, one can argue that the number of error-free bits received over any reasonable wireless channel should be much more than the number of corrupted bits. The second observation stated next formulates this claim in terms of Markov chain parameters.

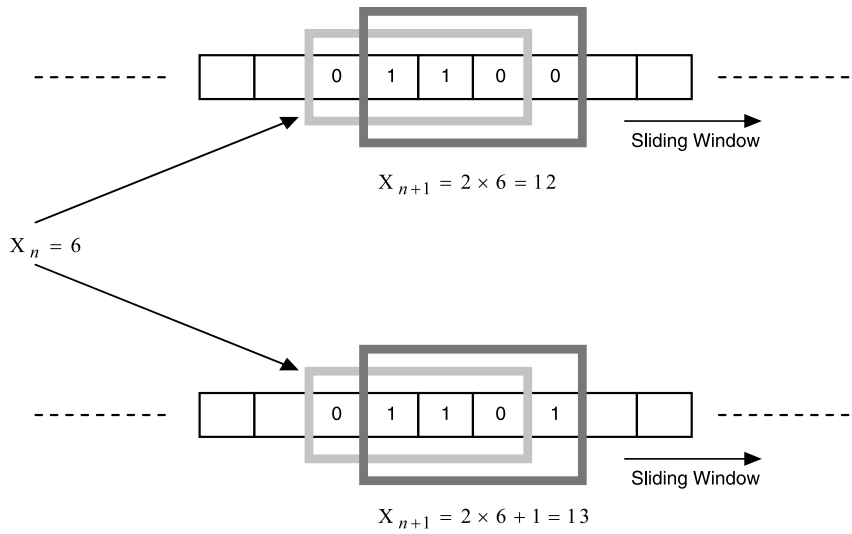


FIGURE 11.4: Transition possibilities for a fourth-order Markov chain.

Table 11.2: Empirical evidence in support of Observation 2.

	2-Mbps bit error traces	5.5-Mbps bit error traces
π_0	0.997	0.974

Observation 2. *The steady-state probability of state 0 of a kth order Markov chain for wireless channels is much greater than the steady-state probabilities of all other states,*

$$\pi_0 \gg \sum_{j=1}^{2^k-1} \pi_j, \tag{11.9}$$

where k represents the memory length and π_i represents the steady-state probability of being in state i of the Markov chain.

This observation implies that the mean time spent in state 0 of the Markov chain (i.e., the state with no errors) is much greater than the mean time spent in all other states. As explained earlier, it can be intuitively argued that this observation holds for real-life wireless/mobile channels. Table 11.2 gives the steady-state probabilities of the 802.11b 2-Mbps bit error Markov chain of order 10 and the 5.5-Mbps bit error Markov chain of order 8. Since the steady-state probability of staying in the good (all-zero) state is very close to one for the two cases shown in Table 11.2, we can safely claim that Observation 11.2 holds for the wireless channels currently under consideration.

11.5.2 Markov Chain Good-Burst and Bad-Burst Distributions

The objective of the present analysis is to ascertain partitions of Markov state space. States in a particular partition will then be grouped together to form an aggregate state in the low-complexity approximating process. We want to define the Markov state space partitions such that the resulting aggregate process, while being less complex, closely matches the Markov chain characteristics.

Since bursts of good and bad bits on the channel are two fundamental (and arguably the most critical) characteristics that should be captured by an accurate model [7–9], the main Markov chain attribute that we focus on is how it captures bursts of good and bad bits. To that end, in this section we derive generalized probability distributions of good and bad bursts for a kth order Markov chain, where k is an arbitrary positive integer. The probability distributions are derived in terms of Markov chain transition and steady-state probabilities. These distributions render useful insights into important Markov characteristics, which are employed to develop guidelines for defining Markov state space partitions in the following sections.

Before proceeding further, we employ Observation 1 to prove a necessary condition for defining partitions of a Markov chain's state space. Let H and S denote the state spaces of a Markov chain and an aggregate (approximate) process, respectively. Let $i \in H$ and $S_i \in S$ denote two arbitrary states of the Markov and the approximate process, respectively. Then the following lemma imposes a necessary condition for defining aggregate states.

Lemma 1. *The next state in an aggregate process can be determined accurately only if Markov states* $(2i) \bmod 2^k$ *and* $(2i+1) \bmod 2^k$ *do not belong to the same aggregate state,*

$$(2i) \bmod 2^k \in S_j \Rightarrow (2i+1) \bmod 2^k \notin S_j, \tag{11.10}$$

where k is the memory length, $i \in H$, and $S_j \in S$.

PROOF. Lemma 1 is easily proven by contradiction. In essence, this lemma implies that both transition possibilities of a Markov state cannot be aggregated in a single state. As mentioned in Observation 1, $(2i) \bmod 2^k$ and $(2i+1) \bmod 2^k$ are the only possible transitions for state i. Let there exist an aggregate state S_j that contains both states $(2i) \bmod 2^k$ and $(2i+1) \bmod 2^k$. Also, let S_q represent an aggregate state that contains state i. Then, p_{S_q,S_j} does not give any information about whether a good or a bad bit should be added to the memory window.

To simplify notation, from this point forward we drop the $\bmod 2^k$ operation (where k is the memory length) on Markov states. Thus, state $i \bmod 2^k$ is simply written as state i. Let I and B denote the good- and bad-burst random variables. We want to derive closed-form expressions of I and B in terms of Markov chain parameters. The expressions for good- and bad-burst random variables render insights into how a Markov chain captures these random variables. The following theorem states the Markov probability distribution of good bursts.

Theorem 1. *The probability distribution of a good burst of length exactly l,* $\Pr\{I = l\}$*, for a kth order Markov chain is*

$$\Pr\{I = l\} = \sum_{i=0}^{2^{k-1}-1} \pi_{2i+1} \times \mu_i \times \prod_{j=0}^{\min\{k-2,l-2\}} p_{(2i+1)2^j,(2i+1)2^{j+1}},$$

$$\forall k,\ l > 0,\ \text{where } \mu_i = \begin{cases} p_{(2i+1)2^{l-1},(2i+1)2^l} \times p_{(2i+1)2^l,(2i+1)2^{l+1}+1}, & l < k, \\ p_{2^{k-1},0}(p_{0,0})^{l-k} p_{0,1}, & l \geq k. \end{cases} \tag{11.11}$$

PROOF. Before proceeding with the proof, we recall that the subscripts of all transition and steady-state probabilities are modulo 2^k. Let us focus on the proof of the $l \geq k$ case, as the proof of the other case is much simpler and follows a similar procedure. Given any current state, a good burst (i.e., burst of 0's) will start if the current state has a 1 in the LSB position of the memory window, that is, the current state represents an odd-numbered Markov state $X_n = 2i + 1$, $0 \leq i \leq 2^{k-1} - 1$.

Without loss of generality, consider the state path given in Figure 11.5. For a good burst of length l starting in the current odd-numbered state, the next $k - 1$ transitions will be $(2i + 1), (2i + 1)2, (2i + 1)2^2, \ldots, (2i + 1)2^{k-1}$. Note that $(2i + 1) \bmod 2^{k-1} = 2^{k-1}$ and, based on the discussion in Observation 1, the process wraps around to state 0 at this point, that is, at point $X_{n+k-1} = 2^{k-1}$, the good burst continues and the process wraps around, $X_{n+k} = 0$. This transition sequence is followed by $l - k$ zero bits, that is, the next $l - k$ transitions are from state 0 to state 0, giving $X_{n+k} = X_{n+k+1} = X_{n+k+2} = \cdots = X_{n+l} = 0$. The good burst ends when a 1 bit is encountered at the $(l + 1)^{\text{st}}$ transition and the Markov process moves to $X_{n+l+1} = (00 \ldots 01)_2 = (1)_{10}$. When expressed in the form of probabilities, this state transition path will have to be summed over all possible odd-valued Markov states,

$$\Pr\{I = l\} = \pi_1 \left[\begin{array}{l} p_{1,(1)2} \times p_{(1)2,(1)2^2} \times \cdots \times p_{(1)2^{k-2},(1)2^{k-1}} \\ \times\, p_{(1)2^{k-1},0} \times (p_{0,0})^{l-k} \times p_{0,1} \end{array} \right] + \pi_3 \left[\begin{array}{l} p_{3,(3)2} \times p_{(3)2,(3)2^2} \times \cdots \times p_{(3)2^{k-2},(3)2^{k-1}} \\ \times\, p_{(3)2^{k-1},0} \times (p_{0,0})^{l-k} \times p_{0,1} \end{array} \right] + \cdots$$

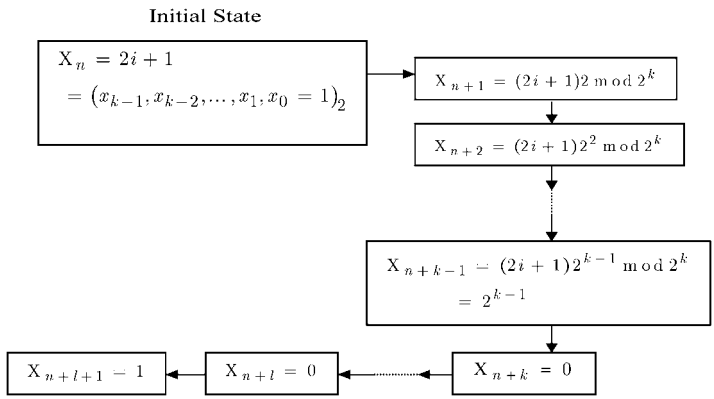


FIGURE 11.5: State transitions of a kth order Markov chain with a good burst of length $l \geq k$.

$$+\pi_{2^k-1}\begin{bmatrix} p_{2^k-1,(2^k-1)2} \times p_{(2^k-1)2,(2^k-1)2^2} \\ \times \cdots \times p_{(2^k-1)2^{k-2},(2^k-1)2^{k-1}} \\ \times p_{(2^k-1)2^{k-1},0} \times (p_{0,0})^{l-k} \times p_{0,1} \end{bmatrix}.$$

Taking out common terms yields

$$\Pr\{I=l\} = p_{2^{k-1},0}(p_{0,0})^{l-k}p_{0,1}\left[\sum_{i=0}^{2^{k-1}-1}\pi_{2i+1}\prod_{j=0}^{k-2}p_{(2i+1)2^j,(2i+1)2^{j+1}}\right],$$

which is the same as the expression in Theorem 1 for all $l \geq k$.

Similarly to Theorem 1, the probability distribution of a bad burst of length l is given in the following theorem.

Theorem 2. *The probability distribution of a bad burst of length exactly l, $\Pr\{B=l\}$, for a kth order Markov chain is*

$$\Pr\{B=l\} = \sum_{i=0}^{2^{k-1}-1}\pi_{2i} \times \mu_i \times \prod_{j=0}^{\min\{k-2,l-2\}} p_{(2i+1)2^j-1,(2i+1)2^{j+1}-1},$$

$$\forall k,l>0,\ \textit{where}\ \mu_i = \begin{cases} p_{(2i+1)2^{l-1}-1,(2i+1)2^l-1} \times p_{(2i+1)2^l-1,(2i+1)2^{l+1}-2}, & l<k, \\ p_{2^{k-1}-1,2^k-1} \times (p_{2^k-1,2^k-1})^{l-k} \times p_{2^k-1,2^k-2}, & l \geq k. \end{cases} \tag{11.12}$$

Proof of this theorem is similar to the proof of Theorem 1.

The expressions for good- and bad-burst probability distributions given in Equations (11.11) and (11.12) are rather convoluted. Hence in their present forms, these expressions neither offer any obvious insight into the random process' behavior nor are they amenable to further analysis. In the following section, we employ Observation 2 to simplify the probability distribution expressions of Equations (11.11) and (11.12). The simplification in turn leads us to the design guidelines that should be followed by a low complexity model.

11.5.3 Simplification of Good-Burst Distribution

Due to Observation 2 the steady-state probability of state 0 is very high and, consequently, the steady-state probabilities of odd states in the good-burst expression of (11.11) are negligible. Thus, the terms involving a transition to or from state 0 of the will dominate the good-burst probability distribution of (11.11). Moreover, since the channel usually stays in the good state for practical wireless networks, the good-burst length should in general be significantly greater than the

memory length. Hence, an effective good-burst probability distribution $\Pr\{I = l\}$ should accurately capture the $l \geq k$ behavior. The good-burst probability distribution of (11.11) for $l \geq k$ can be rewritten

$$\Pr\{I = l\} \approx p_{2^{k-1},0}(p_{0,0})^{l-k} p_{0,1}, \quad \forall l \geq k > 0. \tag{11.13}$$

Although the aforementioned expression is an approximation of the Markov chain's good-burst probability distribution, it is clearly more tractable for analysis. A close look at (11.13) reveals that the parameter characterizing the (approximate) probability distribution is the probability of a good bit transmission followed by another good bit transmission ($p_{0,0}$) since this is the only parameter in (11.13) that involves the good-burst length (l). Hence, one important consideration while grouping states should be that the all-zero (i.e., no-error) state is not grouped with a large number of other states. This is also a natural consequence of Observation 2, which implies that the mean time spent in the all-zero (i.e., no-error) state is significantly higher than all other states.

Similarly, in addition to the state 0, two other important states are state 2^{k-1} and state 1 since $p_{2^{k-1},0}$ and $p_{0,1}$ are the only parameters, other than $p_{0,0}$, that appear in the approximate probability distribution given in (11.13). Hence, due to their relative importance in describing real-life wireless and mobile channels, a good model, in addition to state 0, should not group states 1 and 2^{k-1} with too many other states. This guideline will be employed to define a constant complexity model later.

11.5.4 Simplification of Bad-Burst Distribution

For the bad-burst probability distribution of (11.12), we again invoke Observation 2 and neglect the terms in (11.12) that are not multiplied with π_0. Using this approximation, the bad-burst distribution (11.12) can be written

$$\Pr\{B = l\} \approx \pi_0 \mu_0 \prod_{j=0}^{\min\{k-2,l-2\}} p_{2^j-1,2^{j+1}-1},$$

$$\text{where } \mu_0 = \begin{cases} p_{2^{l-1}-1,2^l-1} p_{2^l-1,2^{l+1}-2}, & l < k, \\ p_{2^{k-1}-1,2^k-1} (p_{2^k-1,2^k-1})^{l-k} p_{2^k-1,2^k-2}, & l \geq k. \end{cases} \tag{11.14}$$

The only terms appearing in (11.14) after the approximation involve states 0, $2^k - 2$, and $2^j - 1$, for any $1 \leq j \leq k$. From Observation 1 and the good-bursts approximation, we have already established that state 0 should not be aggregated with many other states. This deduction is reasserted here. Moreover, it is preferable not to aggregate state $2^k - 2$ with many other states. Also, if possible, all Markov states $2^j - 1$, where $1 \leq j \leq k$, should not be grouped with too many other states.

11.5.5 Guidelines for Approximating a kth Order Markov Chain

The analyses of preceding sections can be summarized in the following guidelines:

Guideline 1: Any state aggregation should satisfy the condition given in Lemma 1.
Guideline 2: State 0 should not be aggregated with other states.
Guideline 3: States 2^{k-1} and 1 should be aggregated with a minimal number of other states.
Guideline 4: States $2^k - 2$ and $2^j - 1$, for all $1 \leq j \leq k$, should be aggregated with a minimal number of other states.

Note that Guidelines 1 and 2 are more assertive than Guideline 3 and Guideline 4. This is due to the analysis provided in the previous section, which outlined that (i) Guideline 1 is necessary for an accurate model and (ii) Guideline 2, which is a consequence of Observation 2, is asserted by the approximate distributions of both good and bad bursts.

It can be observed that Guideline 1, Guideline 2, and Guideline 3 can be easily satisfied in a low-complexity model. However, Guideline 4 is somewhat problematic because putting each $2^j - 1$ state, for all $1 \leq j \leq k$, in a separate partition (i.e., separate aggregate state) makes the total number of states of the approximate model an increasing function of the memory-length k. Thus, satisfying Guideline 3 implies that the resultant complexity (i.e., number of states) of the aggregate model will at least be a linear function of the memory length. We, however, want to keep the number of states in the model independent of the underlying process' memory length. Nevertheless, if linear complexity is acceptable, then Guideline 4 should be incorporated into the design of future wireless channel models.

11.5.6 Constant Complexity Model

Based on the analysis of the last section, a *constant complexity model* (CCM) adhering to Guideline 1, Guideline 2, and Guideline 3 can be developed. The CCM keeps Markov states 0, 1, and 2^{k-1} each in a separate partition, while grouping all the remaining Markov states into two partitions. The resulting model always has five states irrespective of the memory length. The structure and transition possibilities of the CCM are illustrated in Figure 11.6. It is clearly outlined by Figure 11.6 that the CCM assigns separate (isolated) states to Markov states 0, 1, and 2^{k-1}, thereby adhering to Guideline 2 and Guideline 3. All remaining *even* Markov states are grouped in a single aggregate CCM state, while all remaining *odd* Markov states are grouped in another aggregate state. Note that none of the CCM states contain both an odd and an even state, that is, an aggregate state contains either even states or odd states. Thus, Guideline 1, which requires that

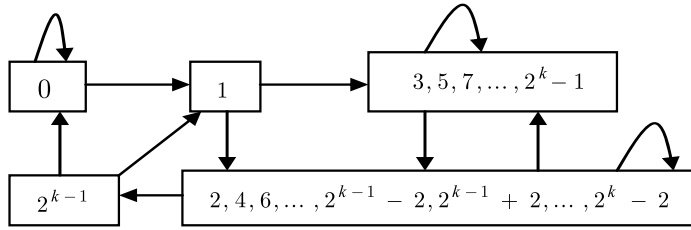


FIGURE 11.6: State aggregation and transitions for the CCM.

Markov states $2i$ and $2i + 1$ are never aggregated together, is also satisfied by the CCM. Clearly, irrespective of the underlying process' memory length, the CCM always comprises five states. Based on our analysis, this five-state CCM should follow the behavior of the underlying 2^k state Markov process quite closely. This CCM efficacy is highlighted adequately in the next section where we compare its performance with Markov chains.

The Markov state space partitioning used by the CCM is only one of the many possible state assignments. Low-complexity channel models can also define other state partitions that should perform adequately as long as the aforementioned guidelines are followed.

11.5.7 Comparison of CCM with Markov Chains

11.5.7.1 Modeling Accuracy for the 2-Mbps Bit Error Process

Figure 11.7 provides ENK-based performance comparison of CCMs with varying memory lengths. Although the CCMs of Figure 11.7 have different memory lengths, the total number of states is five for all the CCMs. ENK overhead of the 1024-state Markov chain model formulates a criterion for performance evaluation of the CCMs. It is clear from Figure 11.7 (top) that for the good-burst random variable, CCMs with memory lengths of five and above perform as well as the 1024-state Markov model. Figure 11.7 (bottom) shows that the CCM ENK overhead for the bad-burst random variable is relatively higher than the Markov model. Nevertheless, in absolute terms the CCM ENK overhead is quite small. While keeping both good and bad bursts under consideration, the CCM model with a memory length of eight provides the best performance. The performance saturates after a memory length of eight, and therefore higher memory lengths are not shown in Figure 11.7. In conclusion, ENK divergence highlights that the CCM provides an accurate and low-complexity bit error model for 802.11b LANs operating at 2 Mbps. This performance substantiates the earlier analysis suggesting that a 5-state CCM can render a performance comparable to the respective 2^k-state Markov chain.

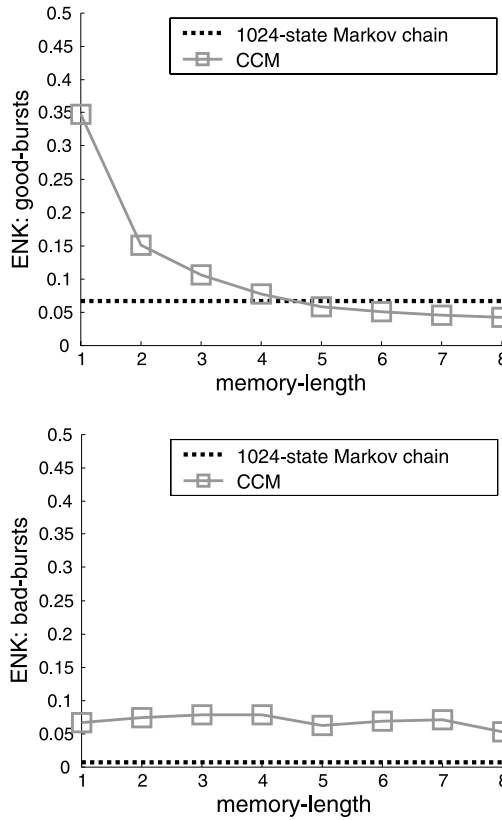


FIGURE 11.7: ENK-based modeling accuracy of the CCM for the 2-Mbps bit error process: (top) good bursts and (bottom) bad bursts.

Table 11.3: Standard errors of CCM and Markov chains.

	2 Mbps		5.5 Mbps	
	1024-state Markov chain	5-state CCM	512-state Markov chain	5-state CCM
Good bursts	0.018	0.018	0.011	0.018
Bad bursts	0.007	0.007	0.009	0.01

Table 11.3 lists the standard error of good- and bad-burst CDFs generated by a 1024-state Markov model and a 5-state CCM with a memory length of eight. It can be clearly seen from Table 11.3 that the 5-state CCM perfectly matches the performance of the 1024-state Markov model for both good and bad bursts. Thus

it can be concluded that the 5-state CCM, while providing orders of magnitude reduction in complexity, renders a performance that is quite comparable to the 1024-state Markov model.

11.5.7.2 Modeling Accuracy for the 5.5-Mbps Bit Error Process

ENK-based performances of CCMs with varying memory lengths at 5.5 Mbps are outlined in Figure 11.8. For the good-burst random variable, performances of CCMs with memory lengths six and higher are comparable to the 512-state Markov chain. Thus, the CCM captures the good-burst behavior of the 5.5-Mbps

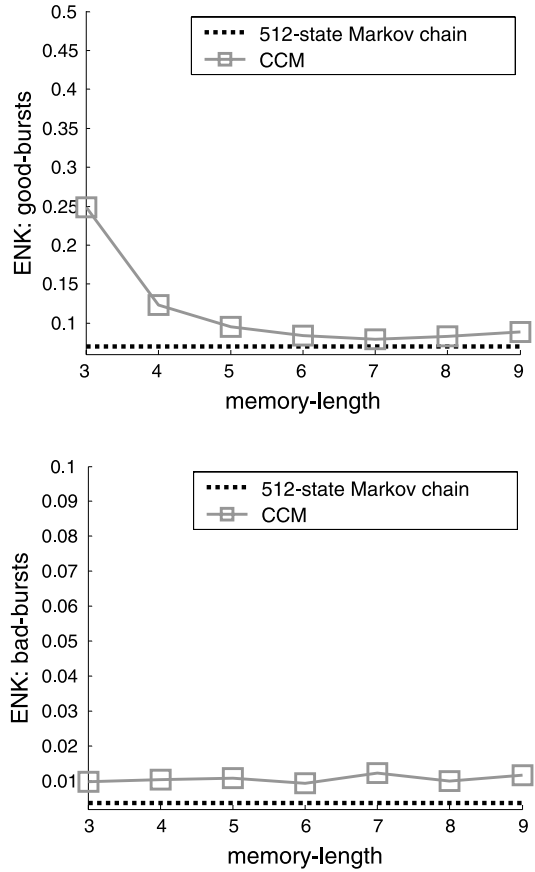


FIGURE 11.8: ENK-based modeling accuracy of the CCM for the 5.5-Mbps bit error process: (top) good bursts and (bottom) bad bursts.

channel very accurately. Similarly, Figure 11.8 (bottom) shows that the bad-burst ENK overhead of CCM is also very small for all memory lengths.

Table 11.3 compares the standard error-based performances of the 512-state Markov model and the 5-state CCM with a memory length of six. Table 11.3 reemphasizes that the CCM performance in modeling good and bad bursts is quite close to the Markov chain. Thus, it can be concluded that even for the 5.5-Mbps channel, the performance of the CCM is comparable to the exponential-complexity Markov model.

11.6 SUMMARY AND FURTHER READING

The objective of this chapter was to introduce the readers to the somewhat recent notion of analyzing and modeling bit errors at wireless MAC layers. This chapter discussed the theoretical aspects and practical issues involved in developing the widely used kth order Markov chain model for wireless MAC layer channels. It was highlighted that kth order Markov models, although very accurate, are too complex to be used in practical wireless systems. Consequently, we described a constant-complexity model that approximated the good- and bad-burst behavior of kth order Markov chains.

The kth order Markov chain models described in this chapter have been used to model many error and loss phenomena. Readers interested in packet-loss Markov models for reliable wireless protocols should refer to [6]. Similarly, read [7] to learn about Markov models of frame losses over cellular GSM networks. Refer to [9] to get insight into the impact of physical layer parameters (e.g., modulation type, antenna diversity) on MAC layer bit errors and channel models. Also note that mitigation of the exponential Markov chain complexity has been investigated by many studies, and the complexity reduction technique presented in this chapter is only one of the proposed methods. Another common complexity reduction technique is Markov chain lumpability [10]. Refer to [17] and [18] to understand how Markov chain lumpability can be employed to reduce modeling complexity over wireless channels. Other techniques that use data-driven heuristics to reduce Markov chain complexity have been proposed [4,19,20]. All of these approximate models (including the constant-complexity model derived earlier) invoke certain wireless channel assumptions to reduce Markov chain complexity. Thus to select the best-fit low-complexity model, one should evaluate the strengths and weaknesses of all relevant approximate channel models while taking the particular desired application(s) into consideration.

REFERENCES

[1] ISO/IEC 8802-11:1999(E). “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,” August 1999.

[2] IEEE Std. 802.11b-1999. "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz band," September 1999.

[3] M. Zorzi and R. R. Rao. "On the Statistics of Block Errors in Bursty Channels," *IEEE Transactions on Communications*, vol. 45, no. 6, pp. 660–667, June 1997.

[4] S. A. Khayam and H. Radha. "Linear-Complexity Models for Wireless MAC-to-MAC Channels," *ACM Wireless Networks Journal (WINET)*, vol. 11, no. 5, pp. 533–545, September 2005.

[5] G. T. Nguyen, R. Katz, and B. Noble. "A Trace-Based Approach for Modeling Wireless Channel Behavior," *Winter Simulation Conference*, December 1996.

[6] H. Balakrishnan and R. Katz. "Explicit Loss Notification and Wireless Web Performance," *IEEE Globecom*, 1998.

[7] A. Konrad, B. Y. Zhao, A. D. Joseph, and R. Ludwig. "A Markov-Based Channel Model Algorithm for Wireless Networks," *ACM WINET*, vol. 9, pp. 189–199, 2003.

[8] S. A. Khayam and H. Radha. "Markov-Based Modeling of Wireless Local Area Networks," *ACM MSWiM*, September 2003.

[9] A. Willig, M. Kubisch, C. Hoene, and A. Wolisz. "Measurements of a Wireless Link in an Industrial Environment Using an IEEE 802.11-Compliant Physical Layer," *IEEE Trans. on Industrial Electronics*, vol. (49)6, pp. 1265–1282, December 2002.

[10] J. G. Kemeny and J. L. Snell. *Finite Markov Chains*, Springer-Verlag, New York, 1976.

[11] S. M. Ross. *Introduction to Probability Models*, Academic Press, 7th ed., February 2000.

[12] P. Brockwell and R. Davis. *Introduction to Time Series and Forecasting*, Springer-Verlag, New York, 1996.

[13] N. Merhav, M. Gutman, and J. Ziv. "On the Estimation of the Order of a Markov Chain and Universal Data Compression," *IEEE Trans. on Information Theory*, vol. 35, pp. 1014–1019, September 1989.

[14] M. J. Weinberger, J. J. Rissanen, and M. Feder. "A Universal Finite Memory Source," *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 643–652, 1995.

[15] T. Cover and J. Thomas. *Elements of Information Theory*, Wiley, New York, 1991.

[16] R. J. Adler, R. E. Feldman, and M. S. Taqqu, eds. *A Practical Guide to Heavy Tails*, Birkhäuser, 1998.

[17] A. M. Chen and R. R. Rao. "Wireless Channel Models: Coping with Complexity," *Wireless Multimedia Network Technologies*, pp. 271–288, Kluwer Academic Publishers, 1999.

[18] A. M. Chen and R. R. Rao. "On Tractable Wireless Channel Models," *IEEE PIMRC*, September 1998.

[19] A. Willig. "A New Class of Packet- and Bit-Level Models for Wireless Channels," *IEEE PIMRC*, October 2001.

[20] A. Köpke, A. Willig, and H. Carl. "Chaotic Maps as Parsimonious Bit Error Models of Wireless Channels," *IEEE Infocom*, March 2003.

[21] P. Ji, B. Liu, D. Towsley, Z. Ge, and J. Kurose. "Modeling Frame-Level Errors in GSM Wireless Channels," *Performance Evaluation Journal*, vol. 55, nos. 1–2, pp. 165–181, January 2004.

[22] L. R. Rabiner. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

12

Cross-Layer Wireless Multimedia

Mihaela van der Schaar

12.1 INTRODUCTION

Wireless networks are poised to enable a variety of existing and emerging applications due to their low cost and flexible infrastructure. Figure 12.1 shows the evolution of different wireless technologies with the X axis representing the throughput and the Y axis representing the mobility. The depicted classes of technologies are Wide Area Networks (WANs), Local Area Networks (LANs), and Personal Area Networks (PANs). Cellular networks belong to the class of WANs, Bluetooth and Ultra Wide Bands (UWBs) belong to PANs, and Wireless LANs (WLANs) and HiperLANs belong to LANs. WANs offer greater mobility, but lower data rates, while LANs offer higher bandwidths, but a limited coverage. PAN technologies are often deployed for cable replacement, whereas WLANs are envisioned as the wireless replacement of wired LANs. However, these wireless networks exhibit a large variation in channel conditions not only because of the different access technologies, but also due to multipath fading, cochannel interference, noise, mobility, handoff, and so on, as well as competing traffic from other wireless users. Thus, as the use of these wireless networks spreads beyond simple data transfer to bandwidth-intense, delay-sensitive, and loss tolerant multimedia applications (such as video-conferencing, emergency services, surveillance, telemedicine, remote teaching and training, augmented reality, and entertainment), addressing Quality of Service (QoS) issues becomes essential. Currently, a multitude of protection and adaptation strategies exists in the different layers of the Open Systems Interconnection (OSI) stack. Hence, an in-depth understanding and comparative evaluation of these strategies are necessary to effectively assess and enable the possible trade-offs in multimedia quality, power consumption, implementation complexity, and spectrum utilization that are provided by the various OSI

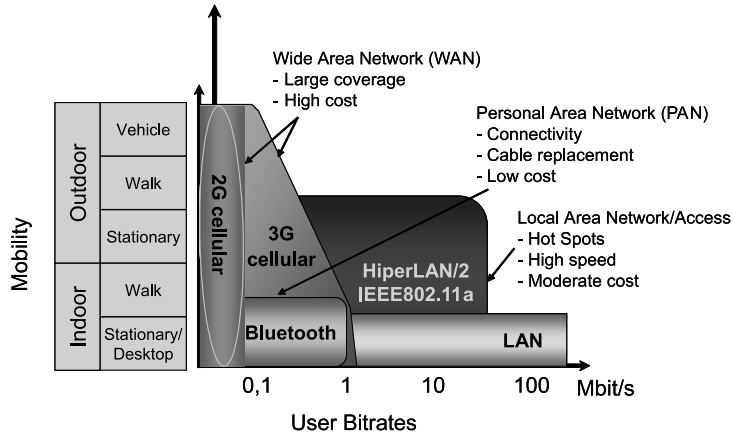


FIGURE 12.1: Current wireless solutions space [72].

layers. This further opens the question of cross-layer optimization and its effectiveness in providing an improved solution with respect to the trade-offs just listed.

This chapter formalizes the cross-layer problem, discusses its challenges and relevant standards, presents several existing solutions, and highlights the key principles for cross-layer design. We discuss a cross-layer framework for *jointly* analyzing, selecting, and adapting the different strategies available at the various OSI layers in terms of multimedia quality, consumed power, and spectrum utilization. Developing such an integrated cross-layer framework is of fundamental importance, since it not only leads to improved multimedia performance over existing wireless networks, but it also provides valuable insights into the design of next-generation algorithms and protocols for wireless multimedia systems. Moreover, such a cross-layer approach does not necessarily require a redesign of existing protocols [4] and can be directly applied across existing application and lower layer standards and de facto solutions.

12.1.1 Challenges and Requirements for Wireless Transmission of Multimedia

Wireless networks are heterogeneous in bandwidth, reliability, and receiver device characteristics. In wireless channels, packets can be delayed (due to queuing, propagation, transmission, and processing delays), lost, or even discarded due to complexity/power limitations or display capabilities of the receiver. Hence, the experienced packet losses can be up to 10% or more, and the time allocated to the

various users and the resulting goodput[1] for multimedia bit stream transmission can also vary significantly in time.

This variability of wireless resources has considerable consequences for multimedia applications and often leads to unsatisfactory user experience due to the following characteristics.

- High bandwidths—many consumer applications, for example, High-Definition TV, require transmission bit rates of several Mbps.
- Very stringent delay constraints—delays of less than 200 ms are required for interactive applications, such as videoconferencing and surveillance, while for multimedia streaming applications, delays of 1–5 s are tolerable. Packets that arrive after their display time are discarded at the receiver side or, at best, can be used for concealing subsequently received multimedia packets.

Fortunately, multimedia applications can cope with a certain amount of packet losses depending on the used sequence characteristics, compression schemes, and error concealment strategies available at the receiver (e.g., packet losses up to 5% or more can be tolerated at times). Consequently, unlike file transfers, real-time multimedia applications do not require a complete insulation from packet losses, but rather require the application layer *to cooperate* with the lower layers to select the optimal wireless transmission strategy that maximizes the multimedia performance.

Thus, to achieve a high level of acceptability and proliferation of wireless multimedia, in particular wireless video, several key requirements need to be satisfied by multimedia streaming solutions over such channels: (i) easy adaptability to wireless bandwidth fluctuations due to cochannel interference, multipath fading, mobility, handoff, competing traffic, and so on; (ii) robustness to partial data losses caused by the packetization of video frames and high packet error rates; and (iii) support for heterogeneous wireless clients with regard to their access bandwidths, computing capabilities, buffer availabilities, display resolutions, and power limitations.

12.1.2 Need for Cross-Layer Optimization

In recent years, to address the aforementioned requirements, the research focus has been to adapt existing algorithms and protocols for multimedia compression and transmission to the rapidly varying and often scarce resources of wireless networks [5]. For instance, network adaptive multimedia compression, bandwidth, and channel condition bit stream adaptation, prioritization and layering

[1] This is the correctly received bit-rate/bandwidth and represents the effective bandwidth that can be used by the application layer for video bitstream transmission.

mechanisms, error concealment strategies, rate–distortion modeling, joint source-channel coding, streaming strategies, multiuser resource management and allocation protocols and algorithms, distortion and channel aware scheduling, link layer adaptation, and power and system optimization strategies have been developed.

However, these solutions often do not provide adequate support for multimedia applications in crowded wireless networks, when the interference is high, or when the stations are mobile. This is because the resource management, adaptation, and protection strategies available in the lower layers of the OSI stack—Physical (PHY) layer, Medium Access Control (MAC) layer, and Network/Transport layers—are optimized without *explicitly* considering the specific characteristics of the multimedia applications, and conversely, multimedia compression and streaming algorithms do not consider the mechanisms provided by the lower layers for error protection, scheduling, resource management, and so on [5]. A set of relevant references discussing cross-layer optimization across lower layers of the protocol stack, without considering the multimedia communications requirements, can be found elsewhere [61–71]. This application-layer agnostic (or simplified) optimization leads to a simpler implementation, but can result in very poor performance (objective and perceptual quality) for real-time multimedia transmission when the available wireless resources are limited. As shown later in this chapter, improvements of up to 5 dB can be achieved in such cases through (often low-complexity) cross-layer optimizations that consider the unique features of multimedia applications.

12.1.3 Chapter Outline

To summarize the aim of this chapter is not to provide a complete single solution to the very complex problem of wireless multimedia transmission. Instead, the chapter is aimed at presenting a possible unified and formal approach to the difficult problem of real-time multimedia transmission over wireless networks by familiarizing the reader to several key principles for identifying optimized solutions for cross-layer transmission, efficient designs, and possible practical solutions.

Section 12.2 starts by presenting a short summary of the 802.11 WLAN standard's key features at the MAC and PHY layers and discusses their impact on wireless multimedia. In this chapter, we will mainly focus on WLAN networks to illustrate the design principles, fundamentals, and solutions for optimized multimedia transmission. However, these can be easily applied to other existing and emerging wireless networks, for example, 3G, 4G, and PAN wireless networks. Section 12.3 motivates the need for cross-layer optimization through a simple wireless video streaming example, where the ad hoc choices of lower layers parameters can have a significant impact on video quality. Section 12.4 formalizes the cross-layer design problem and discusses the challenges associated with solving this problem. Moreover, a categorization of the various cross-layer solutions is

also presented. Next, to solve the cross-layer optimization problem, several different methods are proposed: Section 12.5 discusses a joint MAC–application-layer optimization for adaptive retransmission using queuing theory, while Section 12.6 presents a similar MAC–application-layer optimization for adaptive retransmission and packet size adaptation using Lagrangian optimization. Section 12.7 discusses the problem of efficient wireless resource allocation (i.e., allocation of transmission opportunities) using 802.11e. To enable the complex cross-layer optimization to be performed in real time, a low-cost solution relying on classification is outlined in Section 12.8. Section 12.9 discusses fairness strategies for dynamic multiuser wireless interaction. Section 12.10 presents a brief summary of the chapter and provides a list of relevant further reading.

It should be noted that the notation slightly varies across the various sections of the chapter and, for simplicity of notation, that the consistency of notation was only observed within an individual section. For instance, notations for retransmission limits, modulation strategies, packet sizes, and so on are adapted per section.

12.2 SHORT SUMMARY OF 802.11 WIRELESS LAN STANDARD AND IMPACT ON WIRELESS MULTIMEDIA

Before discussing the cross-layer design principles and solutions, we present several basic features of the IEEE 802.11 Wireless LANs [1–3], as this standard is used in the remainder of this chapter to illustrate the cross-layer design. However, note that the methodology for cross-layer design described in this chapter can similarly be applied to other WLAN, PAN, or WAN standards discussed in Figure 12.1. IEEE 802.11 is a wireless version of Ethernet that supports only best-effort services and that has enabled ubiquitous and low-cost broadband wireless access in home, enterprise, and public places such as airports, group meetings, and coffee shops. IEEE 802.11 has several working groups devoted to expanding the application domain of WLAN to include QoS provisioning, increased mobility support, security, and increased data rates. Figure 12.2 lists the various IEEE 802.11 focus areas. The IEEE 802.11a/b/g/n PHY layer provides data rates starting at 6 Mbps up to 54 Mbps, whereas IEEE 802.11b provide rates from 1 Mbps up to 11 Mbps. Alternatively, the emerging IEEE 802.11n standard uses Multiple Input Multiple Output (MIMO) antenna technology to increase data rates beyond 108 Mbps. IEEE 802.11 c/d/f define the bridging functions, international roaming, and interaccess point protocol, respectively, which are able to facilitate bridging function, fast roaming, and means to communicate between different WLAN access points. The IEEE 802.11e adapts the conventional 802.11a/b/g MAC protocol to accommodate QoS and is described in more detail later in this chapter. IEEE 802.11h/i/k define the dynamic frequency selection and transmit power control,

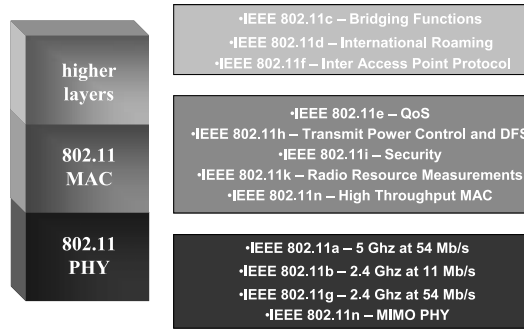


FIGURE 12.2: Overview of IEEE 802.11 WLAN standards.

security, and radio resource measurement to increase the capabilities of WLAN when there is interference and to prevent others from entering into the network unnecessarily.

Next, we discuss some of the basic functionalities provided by 802.11 networks at the various layers that can significantly impact multimedia transmission.

- *PHY layer*. In the 802.11 WLAN standard, several modulation and coding schemes are available to a wireless station. Modulation schemes that have symbols closer to each other in the constellation diagram can result in erroneous decoding. Varying code rates can be employed within each modulation scheme to adapt to changing channel conditions by allowing more bits for channel coding (lower code rates) as conditions deteriorate. As the channel code rate decreases, the effective transmission rate reduces, and hence the achievable throughput for transmission reduces. The 802.11a PHY [1] is based on Orthogonal Frequency Division Multiplexing and provides eight different PHY modes with different modulation schemes and code rates, offering transmission data rates ranging from 6 to 54 Mbps (6, 9, 12, 18, 24, 36, 48, and 54 Mbps, respectively). Figure 12.3 depicts the bit error rate (BER) as a function of the channel condition represented here by the Signal-to-Noise Ratio (SNR) for the various modulation schemes in 802.11a. More details about the channel model can be found in [1,2,10,26].

Similarly, 802.11b also has various PHY modes that can be selected, but unlike 802.11a, the maximum throughput is only 15 Mbps and only four different PHY modes exist. It should also be noted that the aforementioned transmission data rates are at the PHY layer and do not include packetization and transmission overheads incurred at the PHY and MAC layers. Hence, the resulting application-layer rate for multimedia transmission is often significantly lower than the maximum PHY rate.

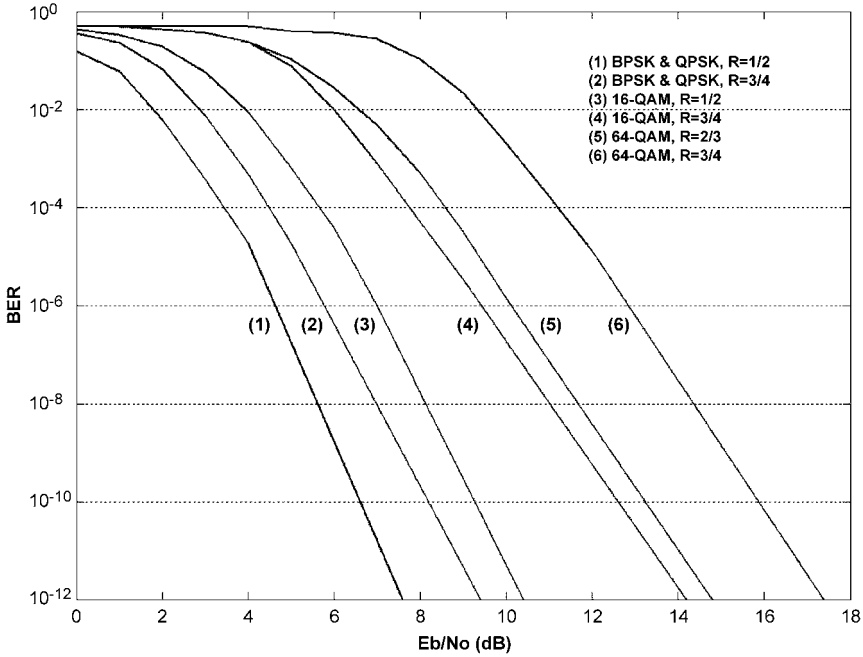


FIGURE 12.3: BER vs. SNR for 802.11a WLAN networks [72].

- *MAC layer*. For a wireless device transmitting delay-sensitive multimedia content, periodic access to the shared wireless medium is paramount. In wireless networks, this access is controlled by the MAC layers. Hence, we briefly discuss the impact of the various existing WLAN MAC layer protocols on multimedia. We start by briefly presenting the 802.11a/b/g WLAN standard, which allows two different MAC mechanisms, namely the distributed coordination function (DCF) and the point coordination function (PCF). Subsequently, we discuss the 802.11e WLAN standard, which provides additional features to the conventional MAC to better enable multimedia streaming.

12.2.1 Distributed Coordination Function

DCF is the basic MAC mechanism, which is based on carrier sense multiple access with collision avoidance. In [54], a very good analytical model of the DCF protocol performance is presented.

In the DCF mode of operation, each station in the WLAN contends for the medium and relinquishes control after transmitting a single packet (MAC frame).

Hence, the DCF MAC strategy provides distributed, fair access to the wireless medium for competing wireless stations. With the DCF mechanism, over a long period of time all users will get equal access to the wireless network. This works well for traditional data applications such as ftp transfers, web browsing, and other delay-insensitive multimedia applications. However, this type of fairness is not appropriate when dealing with real-time multimedia applications that exhibit different delay deadlines and bandwidth requirements. For example, if a video streaming application does not gain timely access to the wireless medium while trying to transmit a very important portion of a compressed bit stream (e.g., the base layer or an "I" video frame) because it is being preempted by competing users, this will lead to unacceptable incurred delays and thus a significantly degraded video quality and a negative user experience. Due to these key disadvantages, this access mechanism is not very suitable for video streaming applications and is not discussed in much detail in this chapter.

12.2.2 Point Coordination Function

PCF is an optional channel access function in the 802.11 standard that is designed to support delay-sensitive applications such as multimedia streaming. Contention-free access to the wireless medium is controlled by a point coordinator (PC) collocated with the access point. PCF is based on a poll-and-response protocol to control access to the shared wireless medium and to eliminate contention among wireless stations. The PC is the central controller, which grants access to the medium. The PC gains control of the medium periodically. Once the PC gains control of the medium, it begins a contention-free period (CFP) during which access to the medium is completely controlled by the PC; after a CFP is finished, a contention period (CP) during which the mandatory DCF is used starts. During the CFP, the PC can deliver downlink traffic to the individual stations without any contention. The PC can also send a contention-free poll (CF-Poll) that allows the stations to send uplink traffic to the PC. If the station that is being polled has uplink traffic to send, it can transmit one packet (MAC frame) for each CF-Poll received. If the station does not have any pending packet, it responds with a data packet without any content, that is, a Null data packet. During the CFP, a wireless station can only transmit after being polled by the PC.

Since there is no contention, a certain QoS level is provided for multimedia applications (even though there is no actual guarantee provided about the actual goodput allocated to an application, which depends on the experienced packet-loss rate incurred due to interference, etc.). Hence, PCF is often used for wireless multimedia streaming, as it provides real-time applications a guaranteed transmission time (opportunity), that is, all stations are polled for a certain amount of time during a service interval.

12.2.3 Enhanced Distributed Channel Access (EDCA)

EDCA is a superset of the 802.11 DCF protocol adopted by the 802.11e standard. In DCF, all wireless stations compete for the wireless medium with the same priority. In EDCA, however, this mechanism is extended to four levels of priorities or access categories (AC). With a shorter maximum back-off time, the higher priority AC wins access to the medium more frequently than the lower priority AC. Therefore, statistically, packets with the highest AC are given access to the medium more frequently than those packets with a lower AC. However, EDCA can be viewed as a differentiated service (DiffServ) QoS that can assist multimedia applications by enabling them to map the various priority packets of the bit streams into various AC classes. Nevertheless, due to the nondeterministic nature of EDCA, it is not possible, except in very lightly loaded networks, to guarantee parameters such as bandwidth, jitter, and latency. The inefficiency arises due to contention and back-off mechanisms as in the DCF case and hence they are not suitable for multimedia streaming.

12.2.4 Hybrid Coordination Function (HCF) Controlled Channel Access (HCCA)

Similar to PCF, HCCA provides real-time applications a guaranteed transmission time (opportunity), that is, all stations are polled for a certain amount of time during a service interval. However, while HCCA provides multimedia streaming applications a certain level of QoS, there are no tight guarantees for parameters such as bandwidth, jitter, and latency, as is shown in Section 12.7

In conclusion, in this chapter we will assume a polling-based MAC for multimedia transmission such as PCF and HCCA. Hence, each service interval (SI) t_{SI} is divided among the various users based on a certain initial admission control policy. For instance, if there are M users in the network, the resource allocation is represented by the transmission opportunity time vector $[t_1, \ldots, t_M] \in \mathbb{R}_+^M$, where t_i $(0 \leq t_i \leq t_{SI})$ represents the transmission time allocated by the MAC to a user i every SI. Such time allocation for the various users is assumed in subsequent sections for discussion of the cross-layer optimized transmission. A detailed description of how the transmission opportunities are allocated to the users and how the stations are polled by the resource coordinator is given in Section 12.7.

However, it should be noted that none of the aforementioned MAC standards provides strict QoS guarantees for multimedia applications; also, the system-wide resource management is not always fair or efficient for such applications. This is due to the time-varying nature of the wireless channel and multimedia characteristics and to the lack of cross-layer awareness of the application and MAC layers about each other.

12.3 EXAMPLE OF CROSS-LAYER IMPACT ON THROUGHPUT EFFICIENCY AND DELAY FOR VIDEO STREAMING

To understand the possible impact of the cross-layer design, let us analyze the impact of the various layers of the protocol stack on the throughput efficiency and resulting delay performance. For illustration, let us assume that the polling-based mode of the 802.11a MAC standard (PCF) is used for video transmission. To protect video data, the adaptive deployment of retransmission at the MAC layer and that of Reed–Solomon (RS) codes at the application layer is considered in addition to the PHY layer modulation and coding strategies provided by 802.11a.

To analyze the overhead impact, the following simplified assumptions are made: (a) the video packets are of length L_a bytes and these packets are not fragmented in any of the lower layers and (b) the overhead of the higher layer protocols, such as RTP, UDP, and IP, is considered to be O bytes. The overhead of the MAC and PHY is not included in this. The average packet transmission duration computed in the following section accounts for the MAC and PHY overhead. The MAC-layer retransmission limit is denoted in this section as R.

12.3.1 Average Packet Transmission Duration

This section analyzes the average transmission duration of a MAC frame under different conditions. This is used later in the computation of application-layer throughput efficiency. Assuming that a packet with L-byte payload is transmitted using PHY mode m, the probability of a successful transmission is given by

$$P^m_{good\_cycle}(L) = \big(1 - P^m_{e,ack}\big)\big(1 - P^m_{e,data}(L)\big), \tag{12.1}$$

where $P_{e,ack}$ is the CF-ACK packet error probability and $P_{e,data}$ is the data packet error probability. These can be calculated from the corresponding packet sizes (including the headers and the payload) and the BER. (See Figure 12.3 and [1] for more details on the various PHY modes and the resulting BER.) The average transmission duration for a good cycle, T^m_{good}, where neither the data packet nor the CF-ACK packet is in error, can be obtained from the timing intervals given in Figure 12.4. Similarly, the average transmission duration for a bad cycle, T^m_{bad}, in a cycle where either the data packet or the CF-ACK packet is in error can be computed from the timing intervals given in Figure 12.5. The average transmission duration for a packet with an L-byte payload, given that the transmission is successful with the retransmission limit of R, can be obtained as

$$D^m_{av,succ}(L, R) = \sum_{i=0}^{R} \frac{P^m_{succ}(i|L)}{P^m_{succ}(L, R)}\big[i\,T^m_{bad}(L) + T^m_{good}(L)\big], \tag{12.2}$$

FIGURE 12.4: Successful downlink packet (MAC frame) transmission and associated timing (SIFS, Short Interframe Space; ACK, Acknowledgment).

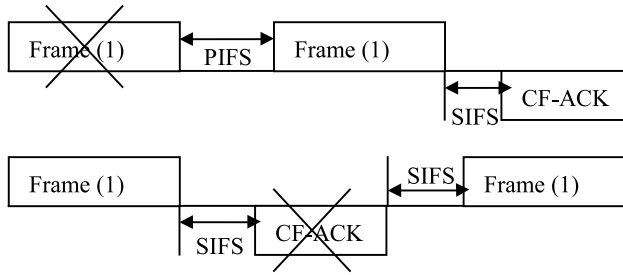


FIGURE 12.5: Retransmission due to packet or CF-ACK transmission error (SIFS, Short Interframe Space; PIFS, PCF Interframe Space).

where the probability that the packet with L-byte data payload is transmitted successfully after the ith retransmission using PHY mode m, is given by

$$P^m_{succ}(i|L) = \left[1 - P^m_{good\_cycle}(L)\right]^i P^m_{good\_cycle}(L), \tag{12.3}$$

and the probability that the packet with an L-byte data payload is transmitted successfully within the R retransmission limit under PHY mode m is given by

$$P^m_{succ}(L, R) = 1 - \left[1 - P^m_{good\_cycle}(L)\right]^{R+1}. \tag{12.4}$$

The average transmission duration for a packet with L-byte payload, given that the transmission is not successful with the retransmission limit R, is

$$D^m_{av,unsucc}(L, R) = (R + 1)T^m_{bad}(L). \tag{12.5}$$

Now, the average transmission duration for a packet with L-byte payload and with a retransmission limit of R is

$$\begin{aligned} D^m_{av}(L, R) = {} & D^m_{av,succ}(L, R)P^m_{succ}(L, R) \\ & + D^m_{av,unsucc}(L, R)\big(1 - P^m_{succ}(L, R)\big). \end{aligned} \tag{12.6}$$

12.3.2 Throughput Efficiency and Delay Analysis with Application-Layer RS Code

The throughput efficiency of 802.11a with the use of the (N, K) RS erasure code at the application layer can be computed based on the average packet transmission duration obtained earlier. Note that here the choice of N and K was determined empirically. However, a model-based joint source-channel coding approach can also be deployed to optimally determine these parameters. For more details, the interested reader is referred to [7–9,17] and the related error protection chapter in this book.

The RS decoder can correct up to $N - K$ packet erasures. If there are more than $N - K$ packet erasures, then this results in a decoding failure. Therefore, the probability of error after RS decoding is

$$P_{RS}^{m} = 1 - \sum_{i=0}^{N-K} \binom{N}{i} \left(P_r^m\right)^i \left(1 - P_r^m\right)^{N-i}, \tag{12.7}$$

where the resulting residual error probability P_r^m of the data packet after R retransmissions is

$$P_r^m = 1 - P_{succ}(L, R). \tag{12.8}$$

When a decoding failure happens, there are $N - i$ $(< K)$ correctly received packets, including both video and parity packets. These video packets can be utilized for video decoding and, on average, $(K/N)(N - i)$ packets out of $N - i$ correctly received packets are video packets. Therefore, the throughput efficiency, taking into the account the application-layer RS coding and the header overheads of the higher layer protocols, is

$$E_{RS}^{m}(L_a, R, N, K) = \frac{8L_a\left(K\left(1 - P_{RS}^{m}\right) + \sum_{i=N-K+1}^{N}(N-i)\frac{K}{N}\binom{N}{i}(P_r^m)^i(1-P_r^m)^{N-i}\right)}{N D_{av}^{m}(L_a + O, R) DR(m)}, \tag{12.9}$$

where DR represents the maximum PHY data rates 6, 9, 12, 18, 24, 36, 48, and 54 Mbps for the various modes m. The numerator here corresponds to the average number of actually received video data bits, and the denominator corresponds to the total average number of bits that could have been transmitted in the time required to send those useful data bits successfully.

However, the impact of the various layers' overheads on the video quality is not only dependent on the throughput efficiency, but also on the overall delay incurred

by the various packets, which need to be dropped if their deadline is exceeded. In this section, the total delay considered comprises different components: the delay due to buffering for RS coding at the transmitter, the RS encoding delay, the delay incurred in the transmission and the retransmission of packets, the buffering delay at the receiver for RS decoding, and the RS decoding delay. At the transmitter, there is no delay due to buffering and as each video packet is stored in the interleaver, it can be transmitted simultaneously, since the RS coding is applied across the video packets and the data transmission is along (not across) the video packets. (See the error protection chapter in this book for more details.) We assume that the process delay due to RS encoding or decoding is small and can be neglected compared to the transmission delay. In certain applications, such as the transmission of a video stored in a residential media server, it may be even possible to perform the (scalable) RS encoding before transmission. The maximum transmission delay depends on the length of a packet, the maximum number of retransmissions, and the specific 802.11a PHY mode m that is used. The worst-case delay[2] for R retransmissions is

$$D_{\max}(m, L) = (R+1)\left(T^m_{data}(L) + T^m_{ack} + 2aSIFSTime\right). \tag{12.10}$$

If there are no packet erasures, then there is no buffering delay at the decoder. Each video packet can be delivered to the video decoder as soon it is received. In the presence of erasures out of the first k packets, the receiver needs to buffer up to n packets that belong to the current RS block before performing the erasure decoding. Therefore, the maximum buffering delay at the decoder is $N \times D_{\max}(m, L)$. Assuming a packet size of 2000 bytes, $R = 8$, and PHY mode 5, the value of $D_{\max}$ is 6.876 ms. Therefore, the total transmission delay for an RS code with $N = 63$ is 433.19 ms. To determine the acceptability of this delay for a particular video streaming application, the video encoding and decoding delay need to be further added on top of the transmission delay.

12.3.3 Impact of Cross-Layer Optimization on Video Quality

This section discusses the interaction between the various transmission strategies deployed by the different OSI layers and their impact on the resulting video quality, thereby highlighting the need for cross-layer optimization.

First, we will determine the optimal packet size that should be selected at the application layer to maximize the video quality Q for a given RS code and retransmission limit. Based on the previously computed [see (12.10)] throughput efficiency E^m_{RS} that takes into the account the application-layer RS coding and

[2]Note that the worst-case delay happens when all of the first R transmissions fail due to CF-ACK transmission failures, not data frame transmissions failures.

the header overheads of the higher layer protocols, the associated video quality $Q_{E_{RS}^m}$ can be computed using, for example, analytical rate–distortion models [17, 18]. The optimal packet size L_a^* can be then computed

$$L_a^* = \arg\max_{L_a} Q_{E_{RS}^m}. \tag{12.11}$$

Equation (12.11) can be solved by evaluating the E_{RS}^m function in (12.10) and determining the resulting $Q_{E_{RS}^m}$ for all possible values of L_a. In a practical implementation, we can use a look-up table by precomputing the values.

Figure 12.6 shows optimal packet sizes for a fixed APP-layer RS code (63,49) with three different numbers of maximum retransmissions, $R = 0$, 1, and 2. It can be seen that for low SNRs, the optimal packet size is the largest for the case corresponding to $R = 2$. For low SNRs, using the maximum allowed number of link layer retransmissions makes the link more reliable, and hence allows the use of larger packet sizes. As the SNR improves, the optimal packet size corresponding to the case of $R = 0$ increases rapidly. This is due to the fact that as the underlying link becomes more reliable at higher SNRs, the resultant packet erasures can be handled by the application-layer FEC even in the absence of any retransmissions.

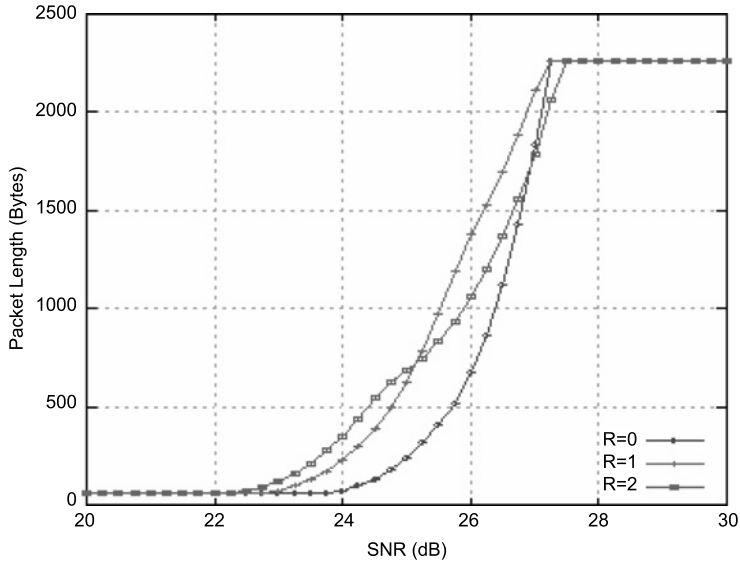


FIGURE 12.6: Comparison of optimal packet sizes for different retransmission limits.

Importantly, note that for an SNR of 26 dB, the difference in quality (PSNR) for the *Coastguard* video sequence is significant for various retransmission choices, varying from a very poor quality of PSNR = 28 dB for $R = 0$, and an acceptable quality of PSNR = 32 dB for $R = 2$, to a very good quality of PSNR = 35 dB for $R = 1$.

12.4 CROSS-LAYER DESIGN

12.4.1 Problem Definition

We formulate the cross-layer design problem as an optimization with the objective to select a joint strategy across multiple OSI layers. Initially, for simplicity, we limit our discussion to PHY, MAC, and Application (APP) layers. Hence, we mainly consider only one-hop wireless networks, where the network and transport layers play a less important role. Nevertheless, the proposed framework can easily be extended to include other layers. (For example, for an extension of the cross-layer design to multihop wireless networks, see [21,60].) For multimedia transmission over multihop wireless networks, the reader is referred to Section 12.10 for a list of related literature.

Let us consider M autonomous wireless stations (WSTAs) that are streaming video content in real time over a shared one-hop WLAN infrastructure. These WSTAs are competing for the available wireless resources $\boldsymbol{\mathcal{R}}$ ($\boldsymbol{\mathcal{R}} \in \mathbb{R}_+$). To use the resources effectively, the wireless stations adapt their cross-layer strategies.

We assume that the channel condition experienced by WSTA i can be characterized by the measured SNR, SNR_i, which varies over time.[3] The current state information for WSTA i is encapsulated in vector $\mathbf{x}_i$, which includes the channel condition SNR_i and the video source characteristics [19] ξ_i, that is, $\mathbf{x}_i = (SNR_i, \xi_i)$. We will refer to this vector as "private information" of the WSTA. Based on the private information, each WSTA jointly optimizes the various transmission strategies available at the different layers of the OSI stack. The PHY strategies may represent the various modulation and channel coding schemes existing for a particular wireless standard. The MAC strategies correspond to different packetization, retransmission, scheduling, admission control, and FEC mechanisms. At the APP layer, strategies may include adaptation of video compression parameters (including enabling spatio-temporal–SNR trade-offs), packetization, traffic shaping, traffic prioritization, scheduling, retransmission, and FEC mechanisms.

As mentioned in Section 12.2, in a typical resource allocation scenario, the resource allocation is represented by the transmission opportunity time vector

[3] Other metrics could also be incorporated in addition to SNR to characterize the channel condition (see, e.g., Chapter 13).

$\mathbf{T}(\boldsymbol{\mathcal{R}}) = [t_1, \ldots, t_M] \in \mathbb{R}_+^M$, where t_i $(0 \leq t_i \leq t_{SI})$ represents the time allocation by the resource coordinator to a specific WSTA i. (The length of the SI, t_{SI}, is determined based on the channel conditions, source characteristics, and application-layer delay constraints [23], and is defined in more detail in the next sections.) This vector denotes the allocated time to WSTA i (with $\sum_{i=1}^{M} t_i \leq t_{SI}$) by either the polling-based MAC resource allocation (see Section 12.3 for details on various MAC strategies) or is obtained on average by a WSTA, through the contention process [54]. As highlighted in Section 12.3, the polling-based MAC is mostly used in practice for video transmission applications and forms the basis of the cross-layer design in this chapter.

Given a static time allocation, and the specific constraints of the WSTA (e.g., application-layer delay constraints), the cross-layer design problem can be formulated as an optimization with a certain objective (e.g., maximize goodput, minimize consumed power) based on which optimal joint strategy across the multiple OSI layers is selected. Let s_i represent a cross-layer strategy available to WSTA i, which lies in the set of feasible PHY, MAC, and APP layers strategies $\boldsymbol{\mathcal{S}}_i$ for that station. The cross-layer strategy s_i is adopted in real time by the WSTA i. Then, given the private information $\mathbf{x}_i$ and the predetermined time allocation t_i, a cross-layer strategy s_i results in the utility $u_i(t_i, s_i, \mathbf{x}_i)$, which for video streaming applications represents the expected received video quality in terms of perceived quality or PSNR. Hence, the optimal cross-layer strategy can be found

$$\begin{aligned} s_i^{opt} &= \arg\max_{s_i \in \boldsymbol{\mathcal{S}}_i} u_i(t_i, s_i, \mathbf{x}_i), \\ &\text{s.t. } Delay(t_i, s_i, \mathbf{x}_i) \leq Delay_i^{\max}. \end{aligned} \tag{12.12}$$

In the formulation just given, $Delay_i^{\max}$ represents the delay constraint for the particular video transmitted by WSTA i and $Delay(t_i, s_i, \mathbf{x}_i)$ represents the delay incurred by the cross-layer strategy s_i for the specific private information $\mathbf{x}_i$ and resource allocation t_i. Importantly, note that, unlike multimedia communications over wired networks, which need to fulfill traditional optimizations in terms of rate and distortion [17], in the wireless multimedia transmission case, the constraint becomes meeting the strict deadlines of the video transmission $Delay_i^{\max}$ given the allocated transmission time t_i (i.e., transmission opportunities allocated or gained by a wireless user). Whenever these delay constraints are not met, the packets are not received in time by the decoder, thereby impacting the resulting utility u_i.

Figure 12.7 depicts a conceptual scheme of the aforementioned cross-layer optimization framework.

Finding the optimal solution to the aforementioned cross-layer optimization problem is difficult because:

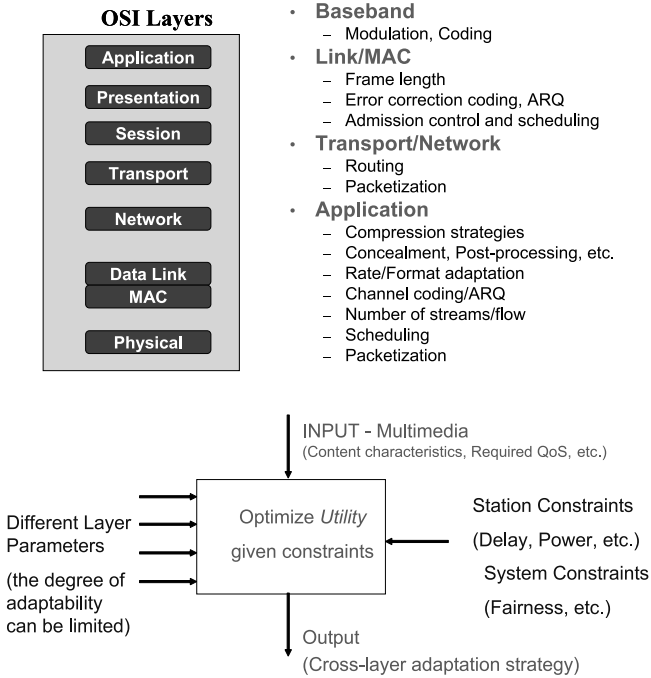


FIGURE 12.7: Conceptual framework of cross-layer optimization.

- Deriving analytical expressions for $u_i(t_i, s_i, \mathbf{x}_i)$ and $Delay(t_i, s_i, \mathbf{x}_i)$ as functions of channel conditions is very challenging, as these functions are nondeterministic (only worst case or average values can be determined), nonlinear, and there are dependencies between some of the strategies s_i [10,11].
- The algorithms and protocols at the various layers have often different objectives and have been traditionally optimized separately. Moreover, various layers operate on different units of the multimedia traffic and take as input different types of information. For instance, the PHY is concerned with symbols and depends heavily on the channel characteristics, whereas the application layer is concerned with the semantics and dependencies between flows and depends heavily on the multimedia content.
- The wireless channel conditions and multimedia content characteristics may change continuously, requiring constant updating of the parameters.
- Formal procedures are required to establish optimal *initialization*, *grouping* of strategies at different stages (i.e., which strategies should be optimized

jointly), and *ordering* (i.e., which strategies should be optimized first) for performing the cross-layer adaptation and optimization.

- For the joint optimization of the strategies, one can use derivative and nonderivative methods (such as linear and nonlinear programming). Because this is a complex multivariate optimization with inherent dependencies (across layers and among strategies), an important aspect of this optimization is determining the best *procedure* for obtaining the optimal strategy s_i^{opt}. This involves determining the initialization, grouping of strategies at different stages, a suitable order in which the strategies should be optimized, and even which parameters, strategies, and layers should be considered based on their impact on multimedia quality, delay, or power. The selected procedure determines the rate of convergence and the values at convergence. The rate of convergence is extremely important, as the dynamic nature of the wireless channels requires rapidly converging solutions. Depending on the multimedia application, wireless infrastructure, and flexibility of the adopted WLAN standards, different approaches can lead to optimal performance. A categorization of the possible solutions is given in the next section.
- Finally, different practical considerations (e.g., buffer sizes, ability to change retry limits or modulation strategies at the packet level [11]) for the deployed wireless standard must be taken into account to perform the cross-layer optimization.

12.4.2 Categorization of Cross-Layer Solutions

To gain further insights into the principles that guide cross-layer design and to compare the various solutions, we propose the following classification of the possible solutions based on the order in which the cross-layer optimization is performed.

- Top-down approach—The higher layer protocols optimize their parameters and the strategies at the next lower layer. This cross-layer solution has been deployed in most existing systems, wherein the APP dictates the MAC parameters and strategies, while the MAC selects the optimal PHY layer modulation scheme. Section 12.6 presents such a cross-layer optimization approach.
- Bottom-up approach—The lower layers try to insulate the higher layers from losses and bandwidth variations. This cross-layer solution is not optimal for multimedia transmission due to incurred delays and unnecessary throughput reductions. The beginning of Section 12.6 presents such a cross-layer optimization approach.
- Application-centric approach—The APP layer optimizes the lower layer parameters one at a time in a bottom-up (starting from the PHY) or top-

down manner, based on its requirements. However, this approach is not always efficient, as the APP operates at slower timescales and coarser data granularities (multimedia flows or group of packets) than lower layers (that operate on bits or packets), and hence it is not able to instantaneously adapt their performance to achieve an optimal performance. Section 12.6 presents such a cross-layer optimization approach.

- MAC-centric approach—In this approach, the APP layer passes its traffic information and requirements to the MAC, which decides which APP layer packets/flows should be transmitted and with which delay or packet-loss requirement. The MAC also selects the PHY layer parameters based on the available channel information and higher layer requirements. The disadvantage of this approach resides in the inability of the MAC layer to perform adaptive source-channel coding trade-offs given the time-varying channel conditions and multimedia requirements. Section 12.5 presents such a cross-layer optimization approach.
- Integrated approach—In this approach, strategies are determined jointly across the various protocol layers. Unfortunately, exhaustively trying all the possible strategies and their parameters in order to choose the composite strategy leading to the best quality performance is impractical due to the associated complexity and incurred delay. A possible solution to solve this complex cross-layer optimization problem in an integrated manner is to use learning and classification techniques that use off-line training data to categorize the various channel conditions and application requirements and identify what are the optimal choices of cross-layer interactions for the various identified categories and, subsequently, use this information to drive the online cross-layer optimization. Section 12.8 presents such a cross-layer optimization approach.

The aforementioned cross-layer approaches exhibit different advantages and drawbacks for wireless multimedia transmission, and the best solution depends on the application requirements, used protocols, and algorithms at the various layers, complexity, and power limitations. Next, we will give several illustrative examples on how to perform the cross-layer optimization and highlight the improvements in multimedia quality and power consumption.

12.5 CROSS-LAYER MAC–APPLICATION-LAYER OPTIMIZATION FOR ADAPTIVE RETRANSMISSION USING QUEUING THEORY

This section illustrates how queuing theory can be used to model the transmission of video packets over an 802.11 WLAN network and how the joint MAC–application-layer interaction can be optimized based on the queuing models, while

fulfilling complexity constraints (such as the MAC buffer limitation). We first design an optimal MAC retransmission limit adaptation strategy to maximize the achieved video quality and subsequently show that by jointly optimizing the MAC retransmission limit along with the application-layer rate adaptation and prioritized scheduling strategies, the decoded video quality can be improved significantly.

12.5.1 MAC-Layer Retransmission Limit Adaptation

First, we consider the optimization of the MAC-layer retransmission in isolation by aiming to maximize the resulting goodput. We assume that the MAC layer is aware of the packet-loss probability of the channel (after the PHY layer channel coding and modulation strategy was deployed), as well as the fixed multimedia traffic rate. This is a realistic assumption for existing 802.11-based wireless video solutions. (See Chapter 13.) Thus, we assume a very simple form of cross-layer parameter communication. However, because of this simplified cross-layer optimization, since the MAC is not aware of the video characteristics, the relative importance and dependencies between packets, the impact of losing specific packets on the quality, and so on, the problem of maximizing the video quality reduces to minimizing the MAC packet loss rate.

At the MAC, packet losses occur due to two reasons: link erasures and buffer overflows. While the loss due to link erasures decreases with an increasing retransmission limit R, the loss due to the buffer overflow increases with an increasing R. Thus, we need to have a strategy to optimally select the R that minimizes the overall MAC packet loss due to buffer overflow *and* link errors. We first show how an analytical solution for the optimal R can be obtained using a fluid model for the buffer queue under static channel error conditions and then illustrate the resulting improvements in multimedia quality using real-time retransmission limit adaptation [11].

A simplified analysis based on a fluid model for the queuing system is presented next. For this, a constant arrival rate λ of the multimedia packets with uniform (packet) size is assumed. Let P be the packet loss probability (controlled by the PHY layer) of the link without retransmission, p_B be the buffer overflow rate, and $p_L = P^{R+1}$ be the link packet erasure rate (i.e., the packet drop rate after R unsuccessful retransmissions). If C is the service rate of the link, the effective utilization factor of the link ρ may be defined as $\rho(P) = \lambda/C(1-P)$. The overall loss rate $p_T(R, P)$, which is defined as the sum of p_B and p_L,[4] may be derived [11]

[4]We assume both p_B and p_L are relatively small such that they can be added together to approximate the total loss rate.

$$p_T(R, P) = p_B(R, P) + p_L(R, P) = 1 - \frac{1}{\rho(P)} \frac{1}{1 - P^{R+1}} + P^{R+1}. \quad (12.13)$$

When P is fixed, $p_B(R)$ monotonically increases with R, while $p_L(R)$ decreases at the same time (see Figure 12.8). To minimize $p_T(R)$, we temporarily relax the discrete constraint on R, assuming it is a continuous variable. R can then be found by solving the equation $\frac{dp_T(R)}{dR} = 0$, which leads to

$$R = \log_P \left(1 - \frac{1}{\sqrt{\rho}}\right) - 1. \quad (12.14)$$

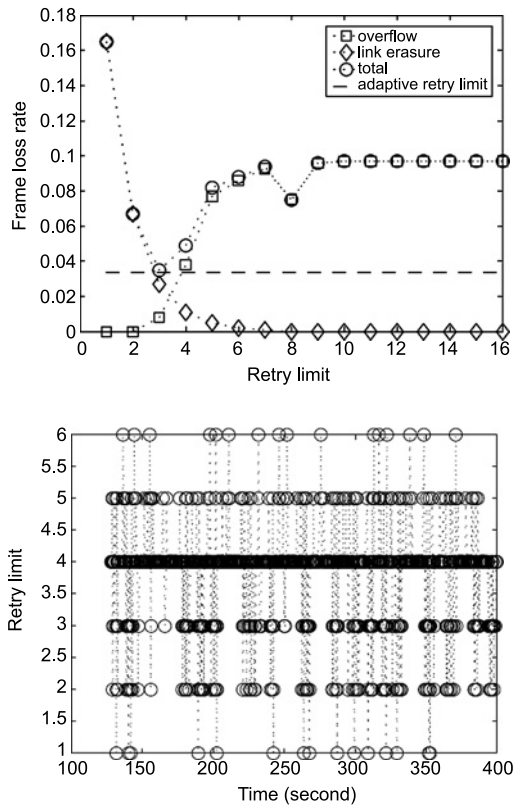


FIGURE 12.8: (Top) MAC PLR under fixed- and RTRO-based retransmission strategies; (bottom) trace of retransmission limit adaptation.

Interestingly, it can be found that at this point $p_B(R) = p_L(R)$, implying that the optimal R is located at the intersection point of the two functions—$p_B(R)$ and $p_L(R)$. This can also be observed from Figure 12.8 (top). The optimized retransmission limit can then be obtained by rounding R to the closest integer. Therefore, we conjecture that we can determine the optimal R as $\arg\min_R |p_L(R) - p_B(R)|$. In other words, the optimal R is chosen as the one that can strike a balance between overflow loss and link loss. In [11], this conjecture was also proven using an M/G/1 queuing model for the video traffic, as well as using real video sequences and transmitting them using an NS-2 simulator-based implementation of the 802.11a/b/g MAC standard. The dependencies of the optimal R on the multimedia packet arrival rate, the experienced link packet loss rate (PLR), and multimedia traffic characteristics (CBR versus VBR) can also be found in [11].

This analysis resulted in the following simple iterative algorithm for *real-time retransmission limit optimization* (RTRO):

1. The network queue *and* the MAC layer monitor the overflow rate $p_B(R)$ and the packet error rate $p_L(R)$.
2. If $p_B < p_L$, then R is increased; if $p_B > p_L$, then R should be decreased.

In Figure 12.8 (top), one typical simulation result is presented based on [11] to show the effectiveness of the aforementioned RTRO strategy. From Figure 12.8, we can observe that an optimal static setting for R exists, depending on the channel conditions and traffic characteristics, which can minimize p_T. However, as illustrated in Figure 12.8, the optimal retransmission setting changes with channel conditions and traffic characteristics, and thus the MAC needs to continuously adapt (optimize) the retransmission limit. Fortunately, even a simple adaptive cross-layer strategy, such as the RTRO strategy, is able to quickly track the optimal retransmission limits, as shown in Figure 12.8 (bottom). See [11] for more details and results.

12.5.2 Joint Application–MAC Cross-Layer Optimization

The MAC-layer RTRO adaptation can also be jointly optimized with the application-layer rate adaptation and prioritized scheduling strategies by associating different retransmission limits to different priority packets. A synopsis of the discussed Application–MAC cross-layer transmission algorithm is given for illustration purposes. See [11] for more details.

The application layer can classify various packets/frames/layers of the compressed video bit stream into different priority classes having different delay requirements, dependencies, relative importance, and impact on the received video quality (see also next section). Let vector $\mathbf{P}_V = [P_{V1} \quad P_{V2} \quad \cdots \quad P_{VN}]$ specify the tolerable MAC packet loss rates of all the video layers (determined by the

QoS requirement of the scalable video). To maximize the video quality $\mathbf{Q}$, unequal error protection (UEP) needs to be provided: higher priority packets need to be transmitted first and with a lower PLR, as they have the highest impact on multimedia quality, while the lower priority packets can be discarded or transmitted with a higher PLR when the channel conditions worsen. To provide UEP, multiple priority queues are maintained at the interface between MAC and application layer [11] and different retransmission limits are used for each of the video layers. All the queues are managed by a common absolute Priority-Queuing (PQ) discipline. To achieve application-layer rate adaptation and prioritized scheduling, several new features can be added to conventional PQ. If c_i is the incoming rate of packets into priority queue i and C is the total available link capacity, then the perceived link capacity of queue j in the worst case[5] can be approximately expressed as $C_j = \max\{0, C - \sum_{i=1}^{j} c_i\}$, where queue priorities decrease with increasing i. As long as $c_j < C_j$, queue j will have few overflow losses. However, all queues will still be exposed to the same packet erasure rate.

The aforementioned analysis for the fluid model and M/G/1 model can be further extended to include a multiqueue system and, based on this, a systematic retry-limit configuration method for the MAC can be determined to optimize the video quality (see [11] for more details). Let $\mathbf{R} = [R_1 \quad R_2 \quad \cdots \quad R_N]$ be the set of the retransmission limits for the different priority layers/packets and $\mathbf{s} = [s_1(R_1, P) \quad s_2(R_2, P) \quad \cdots \quad s_N(R_N, P)]$ be the set of average number of link retransmissions given $\mathbf{R}$ and P. Given the departure rates from the queues to the link $\mathbf{\Lambda} = [\Lambda_1 \quad \Lambda_2 \quad \cdots \quad \Lambda_N]$, determined by the application-layer prioritized scheduling strategy, the overall average number of packet retransmissions can be calculated as

$$\bar{s}(\mathbf{R}, P) = \frac{\mathbf{\Lambda} \cdot \mathbf{s}^T(\mathbf{R}, P)}{\mathbf{\Lambda} \cdot \mathbf{1}}, \tag{12.15}$$

where $\mathbf{1} = [1 \quad 1 \quad \cdots \quad 1]^T$.

We introduce a *shadow retry limit* (SRL) for the MAC, with a corresponding retransmission limit vector $\mathbf{R}^{\mathbf{srl}}$ (all its elements are equal to SRL). This $\mathbf{R}^{\mathbf{srl}}$ is maintained by the MAC but is not enforced on any of the queues. The optimal $\mathbf{R}^{\mathbf{srl}}$ of a multiqueue system can be computed by lumping all the overflows and link erasures observed from different queues into a single overflow rate and link erasure rate and running the MAC-driven RTRO algorithm developed for a single-queue system. Meanwhile, by assuming the same average number of packet retransmissions as that of $\mathbf{R}^{\mathbf{srl}}$, the MAC can compute the actual retransmission limit vector $\mathbf{R}^{\mathbf{re}}$ (with unequal elements) that will be applied to the

[5] Worst case means that every arrival of queue j has to wait for the end of the service of a packet from a higher priority queue.

queues. The mapping from $\mathbf{R}^{\mathbf{srl}}$ to $\mathbf{R}^{\mathbf{re}}$ is performed using the following procedure.

1. Calculate vector $\mathbf{R}_V = [\lceil \log_P P_{V1} - 1 \rceil \ \lceil \log_P P_{V2} - 1 \rceil \ \cdots \ \lceil \log_P P_{VN} - 1 \rceil]$, which specifies the minimum retransmission limit for the queues that can satisfy $\mathbf{P_v}$.
2. Set $i = 1$.
3. Construct $\mathbf{R}^{\mathbf{re}} = [R_{V1} \ \cdots \ R_{Vi} 0 \ \cdots \ 0]$.
4. If $\bar{s}(\mathbf{R}^{\mathbf{re}}, P) < \bar{s}(\mathbf{R}^{\mathbf{srl}}, P)$, go to step 5. Otherwise, continuously reduce the ith components retransmission limit R_i^{re} by 1 until $\bar{s}(\mathbf{R}^{\mathbf{re}}, P) \leq \bar{s}(\mathbf{R}^{\mathbf{srl}}, P)$ *again, and stop*.
5. If $i = N$, stop; otherwise, $i = i + 1$ and go to step 3.

More details about the deployed algorithm and its theoretical foundation can be found in [11].

To highlight the impact of this cross-layer optimization on the achieved multimedia quality we compare the following adaptation schemes:

- No optimized strategies are deployed at the MAC or application layers, that is, no RTRO.
- MAC-layer optimization (RTRO), but with no application-layer awareness.
- Application-layer optimization (rate adaptation and prioritized scheduling), but no MAC-layer optimization.
- Joint application–MAC cross-layer optimization.

The impact of these cross-layer strategies on the perceived video quality was evaluated by performing a visual experiment [11]. Since the experiments are conducted at relatively low bit rates and in the presence of packet losses, impairments are expected, and thus, the selected five scales are very annoying (1), annoying (2), slightly annoying (3), perceptible but not annoying (4), and imperceptible (5). The statistical scores summarized in Table 12.1 clearly illustrate the advantages of cross-layer optimization.

Table 12.1: Subjective video quality experiment.

Deployed strategies	Visual score
No optimization at MAC and application	1.4
MAC-layer optimization (RTRO)	1.9
Application-layer optimization	3.8
Joint application–MAC cross-layer optimization	4.6

12.6 CROSS-LAYER MAC–APPLICATION LAYER FOR ADAPTIVE RETRANSMISSION AND PACKETIZATION USING LAGRANGIAN OPTIMIZATION

This section illustrates a different optimization and modeling methodology (as opposed to the queuing-driven method discussed in the previous section) for the problem of cross-layer design for optimized wireless multimedia transmission, based on Lagrange multipliers.[6] We consider a similar cross-layer problem as before, namely the joint application-layer adaptive packetization and prioritized scheduling and MAC-layer retransmission strategy. The cross-layer problem is posed as a distortion minimization given delay constraints, and analytical solutions are derived based on the well-known Lagrangian optimization framework. In this process, we highlight an important aspect of wireless multimedia transmission: the need to convert conventional rate-constraint problems as conventionally considered by Lagrangian-based video optimizations into time-constrained problems based on the amount of time allocated by the resource moderator of the WLAN (i.e., access point) to a specific station. Moreover, we also discuss the difference between off-line optimizations as opposed to online solutions that can *explicitly* consider real-time available information about previously transmitted packets.

12.6.1 Motivation for Cross-Layer Optimization: A Simple Packetization Example

First, we highlight the need for joint application-MAC layer optimization by evaluating existing adaptive packetization algorithms currently deployed at the MAC layer, which do not explicitly consider the video applications delay constraints and distortions. Section 12.4 mentioned the overhead associated with the packetization and transmission protocols at the various layers of the OSI stack. We refer to this overhead (in terms of bits) discussed in the previous section as L^{Header}. (As before, L^{Header} reflects the packet header as well as the protocol overhead necessary to send a packet in a practical implementation.) Since the MAC is agnostic to the bit stream distortions or the video application delay constraints, the optimization at this layer is simply aimed at maximizing the throughput. Given the channel SNR and the PHY modes, the optimal packet size L^* that maximizes the goodput is computed analytically in [6,26] as

$$L^* = \frac{L^{Header}}{2} + \frac{1}{2}\sqrt{\left(L^{Header}\right)^2 - \frac{4b(L^{Header})^2}{\ln(1-P_s)}}, \tag{12.16}$$

[6]Note that Lagrangian optimization was already used successfully in other video system applications, such as rate-control optimization or joint source-channel coding.

Table 12.2: Decoded PSNR for packet size optimized at the MAC layer.

p_e	Fixed packet size $L = 500$ bytes Decoded PSNR (dB)	Fixed packet size $L = 1000$ bytes Decoded PSNR (dB)	Optimized scheme L^* determined by MAC Decoded PSNR (dB)
6×10^{-6}	32.16	30.25	30.08 ($L^* = 2249$ bytes)
1×10^{-5}	30.45	28.32	27.90 ($L^* = 1738$ bytes)
3×10^{-5}	28.76	25.56	25.86 ($L^* = 997$ bytes)
5×10^{-5}	25.01	24.09	24.12 ($L^* = 768$ bytes)

where b is the number of bits per symbol and P_s is the probability of symbol error, which will depend on the modulation type and link SNR. However, this packet-size optimization mechanism does not consider either the distortion impact of the various packets or the video delay constraints. Illustrative results, comparing the decoded PSNR obtained with this optimal packet size versus alternative ad hoc schemes with fixed packet sizes, are summarized in Table 12.2 for the *Coast-guard* sequence (at CIF resolution 30 frames per second) that was compressed using a scalable codec [13,14] and for an application-layer delay constraint of 400 ms. Furthermore, the header overhead L^{Header} was 30 bytes (240 bits). In all scenarios, the retransmission limits have been set to 0.

From Table 12.2, it can be clearly concluded that the optimal packet size determined at the MAC layer results in a suboptimal performance in terms of the decoded video quality. This motivates the need for cross-layer optimization involving both channel conditions, but also *explicitly* considering the content characteristics and video encoder features, as well as the delay constraints, when determining the packet sizes and the associated retransmissions.

12.6.2 Formalizing the Joint Cross-Layer Optimization Problem Based on Delay Constraints

We consider a video bit stream that is first organized into separate layers based on the delay deadlines of the various video frames (e.g., I-, P-, and B-frames in conventional MPEG and H.26x predictive structures and L- and H-frames in the temporal pyramids of state-of-the-art wavelet scalable coders). In this way, data from different deadline layers are never jointly packetized, as this can lead to suboptimal scheduling strategies due to inefficient exploitation of the remaining transmission time. To facilitate *real-time adaptive* packetization, scheduling and cross-layer optimization with the lower layers based on instantaneous channel conditions and decoding deadlines (without actually requiring the reorganization of the bit stream) an abstraction layer "multitrack hinting" [45] can be adopted, which is an extension of the MP4 file format hinting mechanism [39]. Multitrack hinting can be used to structure the bit stream into multiple sub-layers with different distortion impacts and delay constraints, as illustrated in Figure 12.9. The

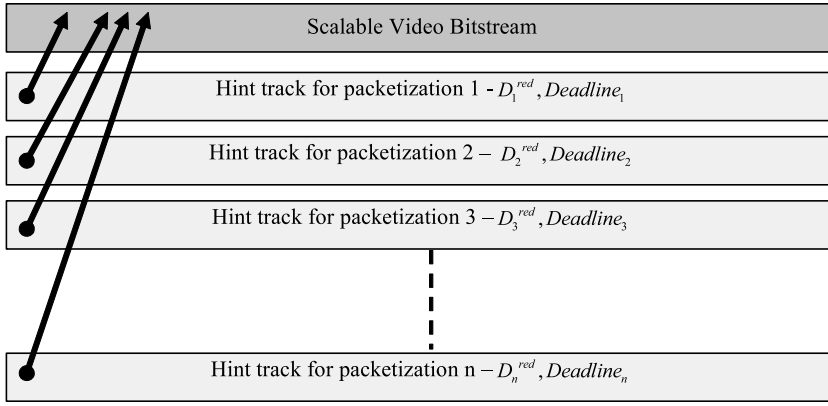


FIGURE 12.9: Deployed multitrack rate–distortion hinting file format.

concept of multitrack hinting was developed in [45] and discussed in more detail in [15].

The multitrack concept is useful for wireless multimedia transmission because it enables (i) real-time adaptation of the packet sizes at transmission time, after the encoding has been performed, (ii) real-time prioritization of different packets based on distortion impacts and changing delay constraints, and (iii) real-time optimized scheduling of video packets based on their deadline and the transmission of the previous packets.

The goal of the cross-layer optimization discussed in this section is to determine the optimal packet size L_j and maximum number of times each packet j can be transmitted, $m_j^{\max}$, such that the expected video distortion is minimized under a given delay constraint. Based on whether packet j is received or lost, the decoded video experiences distortion D_j^{quant} or D_j^{loss}. Hence, when a packet is received successfully, the decoded distortion decreases by an amount D_j^{red}, where $D_j^{red} = D_j^{loss} - D_j^{quant}$. This represents the utility (benefit) of receiving the packet and can be determined using encoding by empirical or analytical rate–distortion models [18]. The goal of the optimization strategy is to maximize the expected utility for a Group of Pictures (GOP),

$$D_{GOP} = \sum_{j=1}^{N_p} D_j^{red} P_j^{succ}, \tag{12.17}$$

with N_p being the number of packets within a GOP and P_j^{succ} being the probability of successfully receiving packet j, given the bit error probability P_e, subject

to a delay constraint,

$$\sum_{k=1}^{j} Time_k \leq Deadline_j, \quad 1 \leq j \leq N_p, \tag{12.18}$$

where $Time_j$ is the actual time it takes to transmit packet j and $Deadline_j$ is the time deadline for the packet to be received at the application layer of the client in order to be decoded and displayed.[7] The deadlines are determined based on the coding dependencies between the video frames (and thus, the encoding structure and parameters) and also include the maximum delay tolerated at the application layer $Delay_{\max}$.

In the wireless video transmission scenario considered in this section, there are two reasons for discarding packets: packet loss from the BER in the wireless link and exceeded packet transmission deadlines. The impact of the buffer overflow or underflow was not considered. We define P_j^{succ} as the probability of successfully receiving the packet given bit errors in the network. With packet size L_j (bits) and bit error probability P_e (controlled by the physical layer, based on the channel SNR, channel coding and modulation strategy used, etc.), the packet loss probability is $P_{L_j} = 1 - (1 - P_e)^{L_j}$. Furthermore, if we assume that the wireless link is a memoryless packet erasure channel [26], such that the packets are dropped independently, the probability of success for packet j with an upper limit on the number of transmissions $m_j^{\max}$ (i.e., a retransmission limit $m_j^{\max} - 1$) can be calculated

$$P_j^{succ} = \sum_{k=1}^{m_j^{\max}} (P_{L_j})^{k-1}(1 - P_{L_j}) = 1 - (P_{L_j})^{m_j^{\max}}. \tag{12.19}$$

The goal of the packetization and retransmission assignment policy is then to solve the delay-constrained optimization problem defined by Eqs. (12.18) and (12.19).

Importantly, two important differences exist between a conventional joint source-channel coding (JSCC) optimization [7–9] and the cross-layer optimization discussed here. First, in JSCC, the optimal channel codes are determined given channel rate constraints, while we are focusing on the delay-constrained transmission scenario for adaptive MAC retransmission and packetization. Second, as discussed in the previous sections, due to the MAC-layer feedback implemented within the 802.11 wireless protocol, we have access to timely informa-

[7] Note that we do not transmit any packets of the GOP beyond the deadline of the last packet within the GOP to avoid any impact on future GOPs, as GOPs are treated as independent entities in the SIV codec.

tion about the lost packets and the actual time that was needed for transmitting a packet $Time_{act}$. Hence, the cross-layer optimization can be performed successfully using online algorithms that combine real-time information with expected packet loss information, unlike in the conventional JSCC schemes that are deployed at the application layer. Furthermore, we rely on the implemented MAC retransmission strategies to consider the actual transmissions for cross-layer optimization rather than considering the current transmission and hypothetical future retransmission to optimize the expectation of a Lagrangian cost function. Hence, rather than modeling the effect of different transmission policies on the properties of a Markov decision process (as in [46,47]) and finding the strategy that maximizes the expectation of the video quality over all possible paths, the features of state-of-the-art wireless LAN protocols can consider determining an optimized cross-layer solution where

i. the feedback is considered to be immediate, such that coding dependencies are guaranteed to be satisfied, thereby avoiding the polynomial optimization objective encountered in [46,47]
ii. the approach is greedy in the sense that all resources can be consumed in transmitting data that has one deadline, before considering the transmission of data with a later deadline.

12.6.3 Packetizing and Retransmitting Data with Common Deadlines

Let us first consider the problem of solving the aforementioned cross-layer optimization for video layers with one common deadline. First, it can be shown that the problem of delay-constrained transmission can be mapped into a rate-constrained transmission. Assume that there are Q deadline layers with a common decoding deadline, which we label *Deadline*. Each deadline layer is partitioned into packets (one or more), and the optimal retransmission strategy is determined for this set of packets. Let one such partitioning lead to a total of $\hat{N}$ packets (for this set of deadline layers). Furthermore, consider that packet j, with packet size L_j,[8] is transmitted m_j times. Then, given the physical layer transmission rate $Rate_{PHY}$, the time to transmit this packet may be computed

$$Time_j = m_j \left(\frac{L_j}{Rate_{PHY}} + Time_O \right), \tag{12.20}$$

where $Time_O$ is the timing overhead for the 802.11 MAC protocol (necessary to send a packet in a practical implementation), which can be approximated based

[8]Given that all bits within a deadline layer have roughly the same importance, we partition each deadline layer into equal parts for packetization. Hence packet sizes L_j are the same for all packets within one deadline layer. Furthermore, we assign the same retry limit to packets from the same deadline layer.

on [1,2,10] and includes the time of waiting for acknowledgments, duration of empty slots, expected back-off delays for transmitting a frame, and so on. Given that all packets have the same deadline, the delay constraint on the packet transmission can be rewritten

$$\sum_{j=1}^{\hat{N}} m_j \left(\frac{L_j}{Rate_{PHY}} + Time_O \right) \leq Deadline, \tag{12.21}$$

or, equivalently,

$$\sum_{j=1}^{\hat{N}} m_j \left(\frac{\hat{L}_j}{Rate_{PHY}} \right) \leq Deadline, \tag{12.22}$$

where the time overhead discussed in previous sections can be included as an equivalent packet length overhead. We can further rewrite this as

$$\sum_{j=1}^{\hat{N}} m_j \hat{L}_j \leq Rate_{PHY} \times Deadline = L_{\max}. \tag{12.23}$$

Hence, the delay constrained optimization becomes

$$\max \left[\sum_{j=1}^{\hat{N}} D_j^{red} P_j^{succ} \right] \quad \text{subject to} \quad \sum_{j=1}^{\hat{N}} m_j \hat{L}_j \leq L_{\max}. \tag{12.24}$$

Note that m_j corresponds to the redundancy rate associated with packet k. However, since the actual value of m_j cannot be determined analytically without actually transmitting the packet (it is a particular instance of an underlying random process), the *expected* redundancy rate, in terms of the expected number of transmissions of the packet, is considered. For packet j with a transmission limit $m_j^{\max}$ the expected number of times the packet is transmitted is

$$\overline{m}_j = \sum_{i=1}^{m_j^{\max}} i(1 - P_{L_j})(P_{L_j})^{i-1} + m_j^{\max}(P_{L_j})^{m_j^{\max}} = \frac{1 - (P_{L_j})^{m_j^{\max}}}{1 - P_{L_j}} = \frac{P_j^{succ}}{1 - P_{L_j}}. \tag{12.25}$$

Using a similar Lagrangian formulation, the optimization functional can be rewritten as

$$F = \sum_{j=1}^{\hat{N}} \left(D_j^{red} P_j^{succ} - \lambda \hat{L}_j \overline{m}_j \right), \tag{12.26}$$

where λ (≥ 0) is the Lagrange multiplier. The goal of the optimization is thus to maximize F. This problem may be further decomposed into a set of $\hat{N}$ independent optimizations for the packets, where the goal is to optimize the individual cost functional

$$F_j = \left(P_j^{succ} - \lambda_j \overline{m}_j\right), \quad \text{with } \lambda_j = \lambda \frac{\hat{L}_j}{D_j^{red}}. \tag{12.27}$$

The optimal solution may be obtained based on the convex hull of the probability of success P_j^{succ} versus the expected redundancy rate $\overline{m}_j$ curve. Note that the actual packet length L_j parameterizes this curve. In particular, the optimal solution $(m_j^{\max,opt}, L_j^{opt})$ is obtained on the curve at the point with the maximum redundancy, where the slope of the curve is larger (or equal) than the parameter λ_j. More details on determining the optimal λ using, for example, a bisection search, and the corresponding optimal retransmission limit and packet size can be obtained in [22].

Comparing the derived expressions for P_j^{succ} and $\overline{m}_j$ for the considered cross-layer optimization, we can clearly see that they have a linear relationship (with a slope $1 - P_{L_j}$). Hence, for such a linear curve, the optimal limit on the number of transmissions $m_j^{\max,opt}$ for packet j will be ∞, when $1 - P_{L_j} \geq \lambda_j$ and 0 otherwise, that is, either transmit a packet until it is received or do not transmit the packet at all. This is an important result, which indicates that the optimum retransmission policy is to retransmit as often as needed ($m_j^{\max,opt} = \infty$) the most important packets corresponding to high distortion impacts and to not transmit the less important packets at all.

Using the aforementioned analysis, a real-time algorithm can be developed to tune the retransmission limit based on the actual number of packet transmissions (instead of the expected redundancy rate). After analytically determining the optimal packet size[9] and the maximum number of packet transmissions (as ∞ or 0), we sort the set of packets in decreasing order of the fraction $(1 - P_{L_j^{opt}})/\lambda_j$. The packets are then transmitted in this order, where no packet is transmitted before all preceding packet transmissions are either completed or terminated. This ensures that coding dependencies between the layers are maintained and that the additive distortion model is not violated. Since in the delay-constrained wireless video transmission the maximum number of times a packet j can be transmitted cannot actually be ∞ and is bounded by the delay deadline *Deadline* (assumed here to be the same for all packets), the limit for each packet is tuned based on the

[9]Note that the packet size is upper bounded by the size of the deadline layers.

actual number of observed transmissions that occurred before it,

$$m_j^{\max,opt} = \left\lfloor \frac{Deadline - \sum_{k=1}^{j-1} m_k (L_k^{opt}/Rate_{PHY} + Time_o)}{L_j^{opt}/Rate_{PHY} + Time_o} \right\rfloor, \qquad (12.28)$$

where $\lfloor \cdot \rfloor$ is the floor operation. One additional advantage of computing this limit in real time is that we can recompute the retransmission limits (and also packet sizes if necessary) when the channel condition P_e or used PHY modulation strategy (that determines $Rate_{PHY}$) changes. Hence, for data belonging to different deadline layers with a common deadline, we can determine the optimal packet size as well as the retransmission limit using the aforementioned analysis. The next section shows how this approach can be extended to the case when we have deadline layers with different decoding deadlines for the real-time transmission scenario.

12.6.4 Real-Time Cross-Layer Algorithm for Wireless Video Streaming

This section extends the previous analysis to include sets of packets with different deadlines, as is the case in typical video streaming scenarios. One approach used to solve this cross-layer optimization is to formulate it as a joint optimization problem (optimization across different deadlines, quality layers, etc.) as performed in [47]. However, the complexity of such an algorithm increases rapidly with the number of different deadlines that need to be considered, especially as each transmission impacts all future transmissions and it is thus not practical. Additionally, such a joint optimization would require that assumptions are made about future channel conditions, modulation strategies employed, and so on.

Instead, a real-time greedy approach can be used, which is based on the analysis in the previous section, but which has the benefits of simplicity, as well as of enabling real time, instantaneous adaptation to varying channel conditions or PHY modulation strategies. In this greedy approach, the optimization problem (to determine the optimal packet size and the retransmission limit) can be solved independently for each set of deadline layers with a common deadline. Note that this approach does not consider the benefits of transmitting packets (deadline layers) with a late deadline before packets with an earlier deadline (which might be advantageous in some cases). However, a major advantage of three-dimensional wavelet video coders, as discussed in Chapter 5, is that packets with the largest distortion impact are mostly transmitted with an early decoding deadline due to the hierarchical temporal structure deployed, and hence the greedy algorithm is likely to perform close to the optimal solution. The real-time greedy algorithm is outlined in more detail in Table 12.3. In Table 12.3, the superscript k corresponds to a group of packets having the same deadline $Deadline^k$.

Table 12.3: Illustrative real-time greedy algorithm for adaptive packetization and MAC retransmission.

Set $Time_{cur} = 0$.
Compute the decoding deadlines for each subband, and hence each code block, based on the encoding parameters, and tolerable application delay. Let there be K separate deadlines (*with values* $Deadline^k$).
Reorganize (*hint*) *the scalable bit stream into deadline layers*.
Sort the deadlines in ascending order.
For $k = 1 : K$
    Gather all deadline layers with deadline ($Deadline^k$).

    Determine instantaneous channel conditions P_e *and PHY modulation strategy* $Rate_{PHY}$.

    Solve the equivalent rate-constrained optimization using the probability of success versus expected redundancy rate curve to determine optimal λ *and determine optimal packet sizes and initial retransmission limits* (*as* ∞ *or* 0).
    Packetize data using these obtained packet sizes $L_j^{opt,k}$.
    Sort the packets in descending order of $\left(1 - P^k_{L_j^{opt,k}}\right)/\lambda_j^k$.
    For $j = 1$: $\hat{N}^k$
    Tune the actual retransmit limit $m_j^{\max,opt,k} = \left\lfloor \frac{Deadline - Time_{cur}}{(L_j^{opt,k}/Rate_{PHY} + Time_o)} \right\rfloor$.
    Transmit the packet to determine the actual number of transmissions m_j^k ($m_j^k \leq m_j^{\max,opt,k}$).
    Set $Time_{cur} \leftarrow Time_{cur} + m_j^k\left(L_j^{opt,k}/Rate_{PHY} + Time_o\right)$.
    If $Time_{cur} + \left(L_{j+1}^{opt,k}/Rate_{PHY} + Time_o\right) > Deadline^k$, *break*.

In summary, the main conclusions of the aforementioned cross-layer optimization case study are threefold. First, an analytical solution can be determined based on existing joint source channel coding research, for a special case of the considered cross-layer optimization problem, that is, when all packets have the same decoding deadline. Under such a scenario, the optimal cross-layer strategy results in retransmitting the most important packets (subbands) as often as needed and discarding the lesser important packets. Second, this analysis can be used to develop a real-time greedy algorithm to perform cross-layer optimization for the case when different sets of data have different decoding deadlines. This discussed greedy algorithm can successfully take advantage of the available feedback at the MAC about the actual number of times previous packets have been transmitted to correctly determine the number of times the current packet can be retransmitted.

Moreover, this algorithm can also successfully adapt on the fly to the changing channel conditions or physical layer modulation strategies. In [22], it was shown that the discussed algorithms can achieve significant improvements of 2 dB or more for a variety of video sequences, transmission bit rates, and delay constraints as opposed to simple algorithms based on only the packets' importance.

12.7 EFFICIENT RESOURCE RESERVATION MECHANISMS FOR WIRELESS MULTIMEDIA TRANSMISSION USING 802.11E

As mentioned in Section 12.1, in the 802.11e standard [3], a new wireless medium access method called Hybrid Coordination Function (HCF) is introduced, which combines the EDCA contention-based channel access mechanism with the polling-based channel access mechanism HCCA. Both EDCA and HCCA operate simultaneously and continuously. HCF enables differentiated treatment of traffic streams and can be tuned to meet QoS requirements of low latency and jitter. As such, its use for wireless multimedia streaming designs is an important cross-layer design issue. However, in order to achieve optimal transport of video over 802.11e HCCA, we need to accommodate application-layer constraints such as bandwidth variations due to variable bit rate (VBR) coding, delay constraints, and selective packet retransmission, as discussed in previous sections.

12.7.1 Background for Video Transmission over HCCA in IEEE 802.11e

The feasibility of the EDCA and HCCA mechanisms of HCF for multimedia transmission was addressed in [48–51]. In an attempt to optimize scheduling for VBR video traffic, in [49] an approach was presented for efficient scheduling by the resource manager (Hybrid Coordinator [HC]) based on the measured queue sizes of each traffic stream. HCCA was used, as it provides significant benefits over EDCF for applications requiring strict QoS. However, it is important to mention that all these approaches perform optimization at either the application layer or the MAC layer, thereby not benefiting from the advantages provided by the joint application-layer and MAC-layer optimization that can improve the overall system performance significantly.

HCCA-Based Admission Control for Video

HCCA is used to provide a parameterized QoS service. With HCCA, there is a negotiation of QoS requirements between the QoS enhanced wireless station and the HC. Once a stream for a WSTA is established, the HC allocates transmission opportunities (TXOPs) via polling to the WSTA in order to guarantee its QoS requirements. The HC enjoys free access to the medium during the contention-free period and uses the highest EDCA priority during the contention period in

order to (1) send polls to allocate TXOPs and (2) send downlink parameterized traffic. It makes use of the priority interframe space to seize and maintain control of the medium. Once the HC has control of the medium, it starts to deliver parameterized downlink traffic to stations and issues QoS contention-free polls (QoS CF-Polls) to those stations that have requested parameterized services. QoS CF-Polls include the TXOP duration granted to the WSTA. If the station being polled has traffic to send, it may transmit several packets for each QoS CF-poll received respecting the TXOP limit specified in the poll. In order to utilize the medium more efficiently, it is possible to piggyback both the acknowledgment (CF-Ack) and the CF-Poll onto data packets. In contrast to the point coordination function of the IEEE 802.11-99 standard, HCCA operates during both the contention-free period and the contention period (see Figure 12.10).

The admission control and scheduling units enable HCCA to guarantee that the QoS requirements are met once a stream has been admitted in the network. Alternatively, EDCA only provides a QoS priority differentiation via a randomly distributed access mechanism.

To ensure user satisfaction, it is essential that, once admitted, a video stream is guaranteed QoS for its lifetime. Thus, there is a need to control how many streams are admitted to the system and what wireless resources should be allocated to each stream in order to obtain the optimal trade-off between a larger number of admitted stations and an acceptable video quality level for the admitted stations. In other words, a scalable admission control and adaptive protection strategy is necessary. Among the parameters defined in the traffic specification (TSPEC) element of IEEE 802.11e, we discuss the subset of parameters that influence the design of an efficient admission control algorithm for video applications. For each video flow i, these parameters are the peak data rate (P_i), the mean data rate (ρ_i), the maximum burst size (σ_i), the maximum permissible delay (d_i), the nominal MAC service data unit (packet) size (L_i), and the minimum physical-layer transmission rate (R_i).

In conventional video streaming mechanisms, P_i, ρ_i, and σ_i are part of a twin leaky bucket mechanism [52] and are supplied to the MAC by the application layer. Based on the twin leaky bucket analysis, the effective bandwidth for each video flow i can be computed[10]

$$g_i = \frac{P_i}{1 + d_i(P_i - \rho_i)\sigma_i^{-1}}. \tag{12.29}$$

[10]For the first part of the analysis presented in this section, we assume that channel or link state analysis is used in order to determine the additional percentage that needs to be reserved for the bandwidth to cover the losses that may occur in the wireless medium. Initially, we assume an ideal channel condition where no errors occur during the HCCA duration. Modifications imposed in the admission control in order to incorporate the effects of video packet retransmission due to channel errors are presented later.

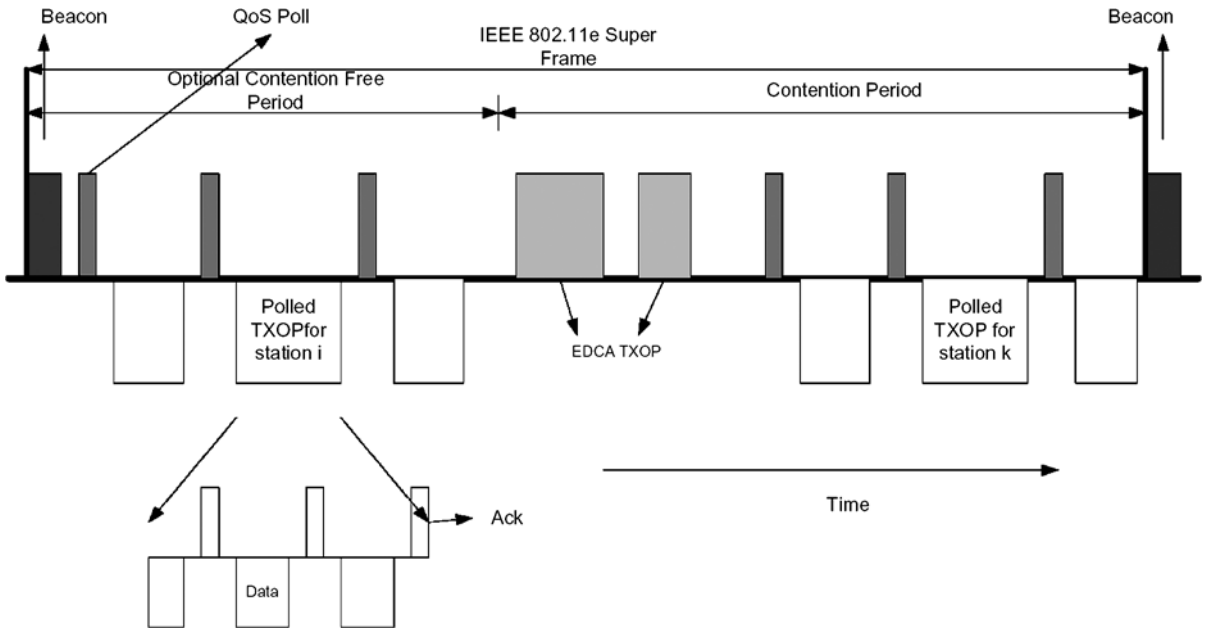


FIGURE 12.10: Operation of the IEEE 802.11e HCF [3].

The previous bandwidth computation is "ideal" in the sense that it does not include overheads. As mentioned in previous sections, for the transmission of each packet there is an overhead in time based on the acknowledgment policy, the PIFS time, MAC-layer and physical-layer headers, and polling overhead. As a result, the scheduling policy has to determine and take into account these overheads, as different scheduling policies determine how many times one has to poll a WSTA in the duration of a service interval (SI), denoted as t_{SI}. Assuming that t_{SI} is known, the number of packets per SI is

$$N_i = \left\lceil \frac{g_i \cdot t_{\mathrm{SI}}}{L_i} \right\rceil \tag{12.30}$$

and the modified guaranteed bandwidth including overheads is

$$g_i' = \frac{N_i(L_i + O_i)}{t_{\mathrm{SI}}}, \tag{12.31}$$

where O_i represents the additional bits due to overheads for the transmission of a packet corresponding to video flow i. As a result, having already $i-1$ admitted flows in the network, the admission control for the ith video flow can be expressed

$$g_i' + \sum_{j=1}^{i-1} g_j' + g_{\mathrm{other}} \leq C, \tag{12.32}$$

where g_{other} represents additional bandwidth allocated to nonvideo traffic (e.g., audio or other QoS-requiring media) and C is the total guaranteed bandwidth of the wireless medium. It is important to mention that a necessary condition for nonviolation of the initially negotiated QoS requirements is that $R_i \geq g_i'$. Based on the readjusted guaranteed bandwidth, the number of packets per SI is recalculated as in (12.30) with g_i replaced by g_i', and for each video flow i we denote the modified value by N_i'. The admission control unit can now calculate the TXOP duration required to service all these packets within t_{SI},

$$t_{\mathrm{TXOP},i} = N_i' \left(\frac{L_i}{R_i} + T_{\mathrm{overhead},i} \right), \tag{12.33}$$

with $T_{\mathrm{overhead},i}$ the required overheads, as explained earlier. Similar to (12.32), we can express the admission control in terms of the TXOP duration for each video flow i:

$$\frac{t_{\mathrm{TXOP},i}}{t_{\mathrm{SI}}} + \sum_{j=1}^{i-1} \frac{t_{\mathrm{TXOP},j}}{t_{\mathrm{SI}}} + t_{\mathrm{TXOP,other}} \leq \frac{T - T_{\mathrm{CP}}}{T}, \tag{12.34}$$

where $t_{\text{TXOP,other}}$ indicates the TXOP allocated to nonvideo traffic, T is the beacon interval illustrated in Figure 12.10, and T_{CP} is the time reserved for the contention period, that is, for EDCA traffic. Importantly, it should be noted that the $T_{\text{overhead},i}$ and $t_{\text{TXOP,other}}$ have a significant impact on the number of admitted stations, as will be shown by the presented results.

The admission control expressed by (12.34) can be used for the construction of a round-robin, standard-compliant scheduler. In particular, the normative behavior set by the IEEE 802.11e standard [3] requires that the HC grants every flow i the negotiated time $t_{\text{TXOP},i}$. Hence, for every video flow, the admission control described by (12.33) and (12.34) can be used. The remaining unknown parameter is t_{SI}, which is typically calculated [23,48] as

$$t_{\text{SI}} = 0.5 \min\{d_1, \ldots, d_n\} \tag{12.35}$$

for a total of n flows to be scheduled. Note that out of the n flows, several can be video flows, audio flows, or other delay-stringent applications. In addition, factor 0.5 is used to accommodate the jitter constraints demanded by the particular applications.

In order to better understand the challenges and limitations associated with deploying the previously described HCCA admission control for video, we consider the transmission of an MCTF scalable video coder,[11] whose particular architecture is outlined in Chapter 5. In typical MCTF-based video compression, the rate allocation for scalable bit stream extraction is performed with a maximum granularity of one GOP. This creates VBR characteristics for the compressed video content across the frames of each GOP. In addition, each decoded frame of every GOP has its own playback deadline determined by the frame rate. Note that, based on the MCTF structure, the decoding frame rate itself can be reduced dyadically by skipping the frames of the finer temporal levels [13]. Frame rate scalability will be useful in the cross-layer adaptation strategy that maximizes the number of admitted stations in the wireless network (see Chapter 5 for more details on how to perform spatio-temporal–SNR trade-offs).

12.7.2 A Scalable Video Admission Control Mechanism over IEEE 802.11e: The Sub-Flow Concept [16]

Implementation of the simple scheduler explained in Section 12.7.1 is easy, but it can be quite inefficient for real-time video streaming applications. This is be-

[11] However, it is important to emphasize that the cross-layer algorithms highlighted can be deployed with any video coding scheme. The essential part of the enhanced admission control mechanism is determination of the frame dependencies of the deployed video coder (and hence the frame/packet delays and traffic characteristics), which can be established based on the encoding parameters and modeled by direct acyclic graphs [47].

cause video traffic varies over time and consists of frames/packets with considerably varying sizes and different delay constraints. Conventionally, the video is considered as a single stream and the TSPEC parameters are set so that the MAC of IEEE 802.11e HCCA would do the admission control and scheduling as outlined previously. To improve the overall system utilization (number of admission stations), as well as the performance of the admitted stations, we introduce the sub-flow concept in which a video flow (bit stream) is divided into several sub-flows based on their delay constraints as well as on the relative priority in terms of the overall distortion of the decoded video. The application layer enables each sub-flow of the video to interface with the MAC as a separate flow. Each sub-flow has a different priority (determined by its distortion impact) and delay constraint. A sub-flow has its own TSPEC parameters and is admitted independently by the resource coordinator.

The goal is to use the sub-flow mechanism to provide a joint application–MAC optimization that maximizes the number of admitted wireless stations while optimizing the video quality for each admitted station. Given the channel conditions, the resource manager (coordinator) and the cooperating wireless stations have to determine for each application the number of sub-flows the application layer can transmit, as well as their protection strategies (e.g., MAC retry limits per sub-flow), while maximizing the number of wireless stations in the network. This section shows how the global flow traffic can be partitioned into sub-flows that are then shaped by multiple token leaky buckets to determine their individual QoS token rates.

For illustration, let us consider one GOP of 16 frames that is encoded using MCTF (see Figure 12.11). Frames with the same playback deadline are grouped

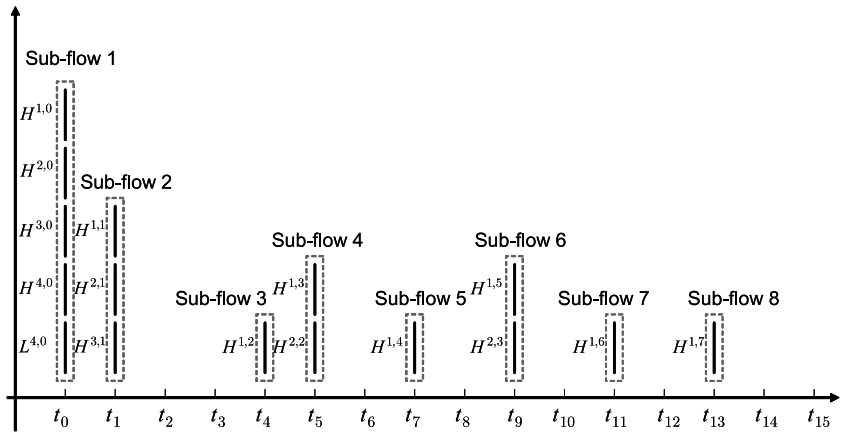


FIGURE 12.11: Example of sub-flow formation.

into the same sub-flow. The number of sub-flows depends on the temporal decomposition levels and the number of reference frames used for motion estimation. If we denote the number of sub-flows of one GOP as N_s we have

$$N_s = 2^{D-1}, \tag{12.36}$$

where D is the total number of temporal decomposition levels. Each sub-flow is regarded as an independent traffic flow passing through a twin leaky bucket to get its own QoS guaranteed bandwidth g_i as expressed by (12.29), with i indicating the sub-flow number, $1 \leq i \leq N_s$, and P_i, ρ_i, σ_i, and d_i the corresponding peak data rate, mean data rate, maximum burst size, and delay constraint of sub-flow i, respectively. As a result, each sub-flow has its own TSPEC parameters and thus there are multiple sets of TSPEC parameters corresponding to one global video flow. A WSTA uses these multiple sets of TSPEC parameters to negotiate with the resource coordinator.

The system performance gain that can be achieved by the discussed sub-flow concept can be quantified theoretically if we introduce the average transmission opportunity duration, $\overline{t_{\text{TXOP}}}$:

$$\overline{t_{\text{TXOP}}} = \frac{1}{N_s} \sum_{i=1}^{N_s} t_{\text{TXOP},i}. \tag{12.37}$$

For a global video flow i, t_{TXOP} is equal to the definition given in (12.33). Following the admission control expressed in (12.34), if we assume only N_{QSTA} video flows for the HCCA transmission intervals, that is, $t_{\text{TXOP,other}} = 0$, by replacing the average transmission opportunity duration for each station by (12.37), we get the maximum number of admitted WSTA carrying video data as

$$N_{\text{QSTA}} = \left\lfloor \frac{t_{\text{SI}}(1 - T_{\text{CP}} \cdot T^{-1})}{\overline{t_{\text{TXOP}}}} \right\rfloor. \tag{12.38}$$

In the following section, we determine the optimal allocated $t_{\text{TXOP},i}$ for each sub-flow i (under predetermined delay constraints) such that the number of admitted stations (N_{QSTA}) is maximized.

12.7.3 Optimization of the Number of Admitted Stations

We discuss a mechanism that maximizes the number of simultaneously admitted wireless stations by optimizing the allocated transmission opportunity duration for each sub-flow. The solution can be obtained using linear programming. Given the allotted TXOP per sub-flow, as shown next, the maximum number of packet

retransmissions can be determined in order to optimize the video quality under the presence of network errors. Subsequently, an algorithm is discussed for dynamic adaptation of packet retransmissions based on this derivation. Finally, we explain how link adaptation can be incorporated in the discussed framework to improve the overall performance for different channel conditions.

12.7.3.1 Optimized Multimedia Admission Control Under Delay Constraints

Although the use of sub-flows may increase the number of admitted stations in the HCCA traffic, if additional delay is permitted in the transmission of each sub-flow traffic, an optimal scheduling algorithm can yield further improvements. A visual example of such a case for one GOP of video data can be seen in Figure 12.12, where each increase in the transmission duration of each sub-flow i, $d_{s,i}$, provides the opportunity for traffic smoothing. In order to accommodate delay requirements, we have $\max\{d_{s,1}, \ldots, d_{s,2^{D-1}}\} \leq d_{\max}$ with $d_{\max}$ set by the chosen streaming scenario.

Each increase in the transmission duration of sub-flow i is reflected by a change in $t_{\text{TXOP},i}$. The optimization goal of maximizing N_{QSTA} given by (12.38) can be equivalently stated as minimizing $\overline{t_{\text{TXOP}}}$ since the other parameters in (12.38) are unaffected by changes in the transmission duration. As a result, if we limit

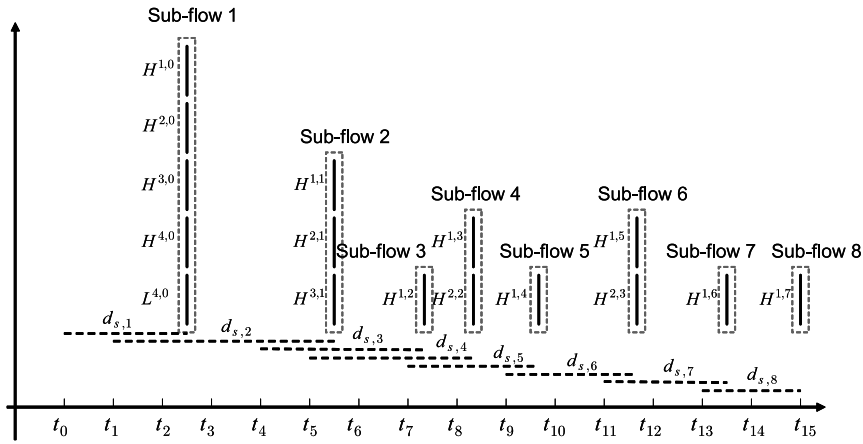


FIGURE 12.12: Sub-flows with different transmission durations due to additional delay permitted. Each $d_{s,i}$, $1 \leq i \leq 2^{D-1}$ corresponds to additional transmission time for sub-flow i. For cases, we assume that an upper bound for the additional delay is set, denoted by $d_{\max}$, and we have $\max\{d_{s,1}, \ldots, d_{s,2^{D-1}}\} \leq d_{\max}$.

optimization to the duration of one GOP (since the video flow traffic is periodic for each GOP) by combining Eqs. (12.30)–(12.33), the minimization problem now becomes

$$\text{Primary problem:} \quad \{t^*_{s,1}, \ldots, t^*_{s,2^{D-1}}\} = \arg\min \sum_{i=1}^{2^{D-1}} g_{s,i}, \tag{12.39}$$

such that $\forall i : 1 \leq i \leq 2^{D-1}$ we have

$$\sum_{j=1}^{i} t^*_{s,j} \leq \sum_{j=1}^{i} (t_{s,j} + d_{\max}). \tag{12.40}$$

In Eqs. (12.39) and (12.40), $\{t^*_{s,1}, \ldots, t^*_{s,2^{D-1}}\}$ are the optimal transmission durations corresponding to sub-flows $1 \leq i \leq 2^{D-1}$, $g_{s,i}$ is the effective bandwidth defined by $g_{s,i} = b_{s,i}/t_{s,i}$, with $b_{s,i}$ the size (in bits) of sub-flow i, and $t_{s,i}$ is the original transmission duration of sub-flow i. Note that this definition of $g_{s,i}$ corresponds to the generic definition of (12.29) if we assume that $P_i = \rho_i$, that is, under the assumption of CBR transmission for the transmission duration of sub-flow i. In order to facilitate the optimization process, the optimization problem stated in (12.39) and (12.40) can be expressed in a dual form [16] as

$$\text{Dual problem:} \quad \{b^*_{s,1}, \ldots, b^*_{s,2^{D-1}}\} = \arg\min \sum_{i=1}^{2^{D-1}} \frac{b_{s,i}}{t^{\max}_{s,i}}, \tag{12.41}$$

such that $\forall i : 1 \leq i \leq 2^{D-1}$ we have

$$\sum_{j=1}^{i} b^*_{s,j} \geq \sum_{j=1}^{i} b_{s,j}. \tag{12.42}$$

In Eqs. (12.41) and (12.42), $\{b^*_{s,1}, \ldots, b^*_{s,2^{D-1}}\}$ are the optimal sub-flow sizes, and $t^{\max}_{s,1} = t_{s,1} + d_{\max}$, $t^{\max}_{s,i} = t_{s,i}$ for $2 \leq i \leq 2^{D-1}$ represent the maximum permissible transmission durations for each sub-flow.

Under CBR transmission for each sub-flow, the Primary and Dual problems stated earlier provide the same solution. For example, by deriving the optimal sub-flow sizes we can establish the optimal transmission duration corresponding to the size of each sub-flow as

$$t^*_{s,i} = t^{\max}_{s,i} \frac{b_{s,i}}{b^*_{s,i}}. \tag{12.43}$$

Nevertheless, one difference of practical significance is that the Dual problem facilitates the application of linear programming techniques, namely the simplex minimization, for establishment of the optimal solution. This ensures optimality with low complexity, as the algorithm converges in a number of steps proportional to the total number of sub-flows, N_s. The simplex optimization scans through all the vertices of the N_s-dimensional simplex in order to establish the point corresponding to the minimum of (12.41) (see [16] for more details). It is important to mention that, in order to formulate a bounded problem for this purpose, we need to impose an upper bound to the maximum number of bits transmitted in the time interval corresponding to one GOP. Hence, we introduced an additional constraint to the problem,

$$\sum_{j=1}^{2^{D-1}-1} b_{s,j}^{*} \leq \sum_{j=1}^{2^{D-1}-1} R_j \cdot t_{s,j}^{\max}, \tag{12.44}$$

which corresponds to the physical constraint that the maximum number of bits transmitted during the duration of one GOP, together with the packetization overhead introduced at the various layers, cannot exceed the mean amount of bits transmitted by the physical layer during this time.

Finally, although the optimization problem is defined and solved for the duration of one GOP, if access to additional sub-flows from consecutive GOPs is possible (e.g., in the case of off-line encoding), they can be included in the optimization problem of (12.41) and (12.42) following the same rationale. Experimental results with real video data utilizing the discussed optimization approach are presented in [16].

12.7.3.2 Packet Scheduling and Retransmissions Under the 802.11e HCCA Admission Control

For the admitted sub-flows of a WSTA, the application and MAC layers can cooperate to improve multimedia quality by adapting the retry limit. The discussions of retry limit adaptation in the previous section are not HCCA enabled and also they do not explicitly consider the delay bound set by the application for the various packets/flows. Here, the goal of packet scheduling and prioritized MAC retransmissions is to minimize the playback distortion for a video streaming session over an 802.11a/e HCCA WLAN, under delay constraints.

Due to limits imposed by link adaptation to different physical layer rates, as well as delay constraints, the retransmission bound for the earlier transmitted packets can be higher than the maximum retransmissions allowed for the remainder set of packets. Hence, under a scheme allowing for unequal video packet retransmissions, a higher probability for correct reception can be provided to the first subsets of video packets. This motivates packet prioritization at the appli-

cation layer depending on the video data significance (incurred distortion due to losing the packet).

The optimal transmission duration for each sub-flow was already established in the previous section by linear programming. In this section, given the set of video packets for each sub-flow i, as well as the transmission duration $t^*_{s,i}$, we establish which subset should be transmitted, as well as the maximum permissible number of video packet retransmissions in case of errors.

Modeling approaches have been proposed for the establishment of substream significance in MCTF-based video compression [17,18]. Most of these models use dynamic computation of the expected distortion using signal statistics or precomputed distortion metadata in conjunction with models for the error propagation across the MCTF decoding structure. Although such solutions result in a model-optimized scheduling with the potential for high accuracy, they can also incur a high computational burden for online processing of many streams. In addition, if we define the number of retransmissions based only on the video packet significance, we will not be able to take advantage of the fact that the MAC layer can provide real-time feedback concerning the correct reception of each individual packet.

Once the ordering is complete, the video packets are placed in packets, which are passed to the MAC layer in the specified order. Although these rules are simply based on the compression architecture and the discussed sub-flow scheduling, the layering principle of fully scalable MCTF-based video coding ensures the optimality of such a scheduling approach. In addition, theoretical studies [18] have shown that the expected distortion–reduction obtained by decoding each video packet is proportional to the temporal and spatial level that the packet belongs to, according to the ordering expressed in the aforementioned rules. We remark that, similar to the previous section, the scheduling algorithm operates independently for each GOP, although extensions to multiple GOPs can be envisaged following similar principles.

Here, the transmission channel is assumed to be an independent, identically distributed error channel. Thus, the channel causes errors independently in each packet and the error probability is the same for all packets with the same length at all times. Let $p_b(m)$ be the bit error probability in physical-layer mode m. Then, the error probability of a packet of size L_i (belonging to sub-flow i) in physical-layer mode m is a function of bit error probability $p_b(m)$ and is defined

$$p_e(m, L_i) = 1 - \big[1 - p_b(m)\big]^{L_i}. \tag{12.45}$$

Let $N^{\max}_{\text{retry}}(j)$ be the maximum number of retries of packet j belonging to sub-flow i. Note that the value of $N^{\max}_{\text{retry}}(j)$ depends on the position of the packet in the transmission queue (derived based on the criteria outlined earlier), as well as on the available transmission duration for the current sub-flow. The probability of

unsuccessful transmission after $N_{\text{retry}}^{\max}(j)$ retransmissions is

$$P_e\big(m, L_i, N_{\text{retry}}^{\max}(j)\big) = \big[p_e(m, L_i)\big]^{N_{\text{retry}}^{\max}(j)+1}. \tag{12.46}$$

In addition, based on $N_{\text{retry}}^{\max}(j)$, the average number of transmissions for the jth packet until the packet is transmitted successfully or the retransmission limit is reached can be found as discussed in the previous sections as

$$N_{\text{average}}(j) = \frac{1 - [p_e(m, L)]^{N_{\text{retry}}^{\max}(j)+1}}{1 - p_e(m, L)}. \tag{12.47}$$

The corresponding average time to transmit the packet using the guaranteed channel rate g_i' for sub-flow i is

$$T_{\text{average}} = N_{\text{average}}(j)\left(\frac{L_i}{g_i'} + T_{\text{ACK}}\right), \tag{12.48}$$

where T_{ACK} is the overhead for the transmission of the acknowledgment frame. Assuming that the maximum time before the packet expires is $T_{\max}$, we have

$$T_{\text{average}} \leq T_{\max}. \tag{12.49}$$

Due to CBR transmission for the duration of each sub-flow, the packets are distributed evenly with an interval α_i (i.e., packet arrival interval for sub-flow i). Assuming that the transmission duration for sub-flow i is $t_{s,i}^*$ (estimated by the optimization of Section 12.7.3.1), for the jth packet of that sub-flow we have

$$T_{\max} = t_{s,i}^* - \alpha_i \sum_{k=1}^{j-1} N_{\text{retry}}^{\text{actual}}(k), \tag{12.50}$$

where $N_{\text{retry}}^{\text{actual}}(k)$, $0 \leq N_{\text{retry}}^{\text{actual}}(k) \leq N_{\text{retry}}^{\max}(k)$, is the actual number of retries for each packet k that precedes packet j until an acknowledgment has been received, or the maximum number of retries has been performed. Note that $N_{\text{retry}}^{\text{actual}}(k)$ can be determined dynamically based on feedback from the MAC layer. The last equation can be used in conjunction with Eqs. (12.46) and (12.49) to establish the bound for the maximum-allowable number of retries for the current packet j:

$$N_{\text{retry}}^{\max}(j) \leq \log_{p_e(m,L_i)}\Bigg[\big(1 - p_e(m, L_i)\big)\left(\frac{L_i}{g_i'} + T_{\text{ACK}}\right)^{-1} \times \left(t_{s,i}^* - \alpha_i \sum_{k=1}^{j-1} N_{\text{retry}}^{\text{actual}}(k)\right)\Bigg] - 1. \tag{12.51}$$

Note that the estimated maximum number of retries determined by (12.51) can be negative, depending on whether we exceeded the available bandwidth for sub-flow i or not. In such a case, the remaining packets of the current sub-flow are simply discarded.

12.7.3.3 Scalable Sub-Flow Transmission with Dynamic Adaptation

We outline the steps performed during the actual streaming process for each sub-flow i in Table 12.4. Some of the last packets of each sub-flow will not be transmitted whenever the channel condition deteriorates, as the transmission duration (deadline) determined by the simplex optimization of the previous section does not take into account the retransmissions that will occur based on the algorithm of Table 12.4. This is checked in Step 2 of the algorithm of Table 12.4. Nevertheless, use of a scalable video coding and the prioritization rules for the transmission of the video packets specified before ensure that near-optimal adaptation of the video quality will occur based on the instantaneous channel capacity, as packets with the most important video data will be transmitted first.

An alternative design can be formulated by a priori calculating the maximum number of retransmissions for each packet j based on $p_b(m)$ and using (12.45) and (12.50) with the setting of $N_{\text{retry}}^{\text{actual}}(k) = N_{\text{retry}}^{\max}(k)$ for every $1 \leq k < j$. Then the sub-flow sizes can be readjusted to include the estimated number of retransmissions. This allows for the optimization algorithm of Section 12.7.3.1 to derive optimal transmission durations that include the (worst-case) expected number of retransmissions for packets of the sub-flow. Overall, the latter case is expected to

Table 12.4: Transmission of packets of each sub-flow i.

- Initialization: Establish $p_b(m)$ based on the utilized physical-layer mode. Calculate $p_e(m, L_i)$ from (12.45).
- For each packet j:
  1. Establish $T_{\max}$ based on (12.50). Calculate $N_{\text{retry}}^{\max}(j)$ based on (12.47)–(12.49). Set current_retries $= 0$.
  2. If $N_{\text{retry}}^{\max}(j) \geq 0$
     - Set current_ACK = FALSE; go to Step 3.

     else
     - Discard the current packet as well as the remaining sub-flow packets with the same deadline.
  3. While current_ACK = FALSE AND current_retries $\leq N_{\text{retry}}^{\max}$,
     - Transmit the current packet. Set: current_retries $\leftarrow$ current_retries $+ 1$.
     - Set current_ACK to TRUE or FALSE depending on MAC-layer feedback.
  4. Set $N_{\text{retry}}^{\text{actual}}(j) =$ current_retries.

overprovision bandwidth for each sub-flow, whereas the previous case can lead to some of the least-significant video packets being dropped, depending on the channel condition.

12.7.3.4 QoS Token Rate Adaptation for Link Adaptation

As mentioned earlier, IEEE 802.11a supports eight physical-layer rates from 6 to 54 Mbit/s. Link adaptation selects one appropriate physical-layer mode based on link conditions in order to improve the system goodput and throughput. WSTAs may adapt their physical-layer modulation and coding strategies depending on the link conditions. In particular, the physical-layer rate will be lowered dynamically when the link condition of one WSTA gets worse, that is, when the signal-to-interference noise ratio drops. The TXOP durations calculated by (12.33) will not take into account the new rate when the WSTA switches its default physical-layer rate mode and, as a result, the resource coordinator may deny the traffic stream of the WSTA whose physical rate turns out to be lower than the prenegotiated minimum rate.

In order to keep the number of admitted stations fixed and have graceful quality degradation, we can utilize the packet scheduling algorithm of Section 12.7.3.2 in order to drop packets containing less important video data such that the precalculated TXOP duration can still guarantee the QoS when the physical-layer mode is changed. For this purpose, we need to determine the new effective bandwidth for each sub-flow i, g_i'', under a change in the physical-layer transmission rate. If we assume that the modified rate for the duration of the sub-flow transmission is R_i', from (12.33) we have

$$N_i' = \frac{t_{\text{TXOP},i}}{L_i \cdot (R')^{-1} + T_{\text{overhead},i}}. \tag{12.52}$$

Then, from (12.30) we get

$$g_i' = \frac{N_i \cdot L_i}{t_{\text{SI}}}, \tag{12.53}$$

and because CBR transmission occurs for the duration of the sub-flow transmission, $t_{s,i}^*$, we can calculate the modified sub-flow size, $b_{s,i}'$, using (12.52) and (12.53) as

$$\rho_i' = \frac{b_{s,i}'}{t_{s,i}^*} = g_i' \Rightarrow b_{s,i}' = \frac{t_{s,i}^* \cdot t_{\text{TXOP},i} \cdot L_i}{[L_i \cdot (R')^{-1} + T_{\text{overhead},i}] \cdot t_{\text{SI}}}. \tag{12.54}$$

Note that in the cases where the link adaptation may change the physical layer rate more than once during the sub-flow transmission interval $t_{s,i}^*$, R_i' can be calculated

based on the weighted sum of the different rates,

$$R' = \frac{1}{t^*_{s,i}} \sum_{k=1}^{w} \left[R_{\text{phy}}(k) \cdot t_{\text{phy}}(k) \right], \tag{12.55}$$

where $R_{\text{phy}}(k)$ and $t_{\text{phy}}(k)$ represent the rate and duration, respectively, corresponding to the kth link adaptation during time interval $t^*_{s,i}$ (out of w total adaptations).

The modified sub-flow size estimated by (12.54) may be used to restrict the number of video packets of each sub-flow; depending on the adaptive retransmission scheme of Table 12.4, once the amount of video packets sent reaches $b'_{s,i}$, the remaining packets in the prioritized transmission queue are discarded. Hence, similar to the case of Section 12.7.3.2, the prioritization mechanism ensures that the most significant packets receive the highest priority under link adaptation at the physical layer. An interesting extension of the link adaptation algorithm would be to optimize the chosen packet length depending on the chosen physical layer rate. This should be done having the application-layer packetization restrictions in mind in order not to affect the decoding dependencies.

12.7.4 Examples of Sub-Flow Transmission

In this section, several simple illustrative examples for video transmission over 802.11e are presented based on [16]. First, we highlight the importance of the (nonoptimized) sub-flow concept versus the conventional global flow scheduling. The experiment of Table 12.5 used a typical CIF video sequence—"Foreman," encoded at 30 frames per second (fps), although similar results have been obtained

Table 12.5: Sub-flow QoS token rates and $t_{\text{TXOP},i}$ with $d_{\max} = 200$ ms for the CIF-resolution sequence "Foreman" (2048 kbps, 30 fps).

Traffic name	Components	QoS token rate ρ_i (kbps)	t_{TXOP} (ms)[a]
Sub-flow $_1$	$H^{1,0}$, $H^{2,0}$, $H^{3,0}$, $H^{4,0}$, $L^{4,0}$	10,032	13.89
Sub-flow $_2$	$H^{1,1}$, $H^{2,1}$, $H^{3,1}$	2840	3.96
Sub-flow $_3$	$H^{1,2}$	184	0.26
Sub-flow $_4$	$H^{1,3}$, $H^{2,2}$	1064	1.46
Sub-flow $_5$	$H^{1,4}$	264	0.37
Sub-flow $_6$	$H^{1,5}$, $H^{2,3}$	728	1.00
Sub-flow $_7$	$H^{1,6}$	392	0.54
Sub-flow $_8$	$H^{1,7}$	440	0.61

[a] $\overline{t_{\text{TXOP}}} = 2.76$ msec, $N_{\text{QSTA}} = 7$.

with a variety of video content. The results of this section have been generated with the settings $T = 100$ ms, $T_{CP} = 60$ ms, and $t_{SI} = 50$ ms. The token rates reported in Table 12.5 were calculated based on a simulation with a twin leaky bucket traffic smoothing system [16], and the delay deadline was extended equally for all sub-flows, such that $d_{max} = 200$ ms. For the case of sub-flow scheduling we have $\overline{t_{TXOP}} = 2.76$ msec and, from (12.38), $N_{QSTA} = 7$. Similarly, for the global flow case we get $\overline{t_{TXOP}} = 13.89$ msec and $N_{QSTA} = 1$. The number of admitted stations can be increased if the optimization framework of Section 12.7.3.1 is used. This is shown by the results of Table 12.6, where the number of stations in the sub-flow case is increased to $N_{QSTA} = 10$. In addition, based on the priorities shown in Table 12.6, we can increase the number of admitted stations if the least-significant sub-flows are discarded. This is illustrated in Figure 12.13, where the number of admitted stations is plotted against the number of utilized sub-flows. Figure 12.13 demonstrates that under a progressive decrease in frame rate, resulting from the removal (drop) of the least-significant sub-flows (with the significance indicated in Table 12.6), the number of admitted stations can be increased further. In a collaborative framework, multiple stations may opt to decrease the video frame rate in order to allow for additional stations (or additional video flows) to utilize the wireless medium under HCCA. Also, the desired number of admitted sub-flows, as well as how these sub-flows are prioritized at the application layer, can be determined based on the channel resources, specific video application, and user preferences. For instance, different spatio-temporal resolutions (and corresponding sub-flows) should be selected for the best percep-

Table 12.6: Sub-flow QoS token rates and $t_{TXOP,i}$ with $d_{max} = 200$ ms for the CIF-resolution sequence "Foreman" (2048 kbps, 30 fps) with the optimization framework of Section 12.7.3.1. The "priority" indicates the importance (4, highest; 1, lowest) of each sub-flow in terms of incurred distortion at the receiver.

Traffic name	Components	QoS token rate ρ_i (kbps)	Priority	t_{TXOP} (ms)[a]
Sub-flow $_1$	$H^{1,0}$, $H^{2,0}$, $H^{3,0}$, $H^{4,0}$, $L^{4,0}$	4664	4	6.46
Sub-flow $_2$	$H^{1,1}$, $H^{2,1}$, $H^{3,1}$	2768	3	3.82
Sub-flow $_3$	$H^{1,2}$	216	1	0.31
Sub-flow $_4$	$H^{1,3}$, $H^{2,2}$	1376	2	1.90
Sub-flow $_5$	$H^{1,4}$	344	1	0.46
Sub-flow $_6$	$H^{1,5}$, $H^{2,3}$	880	2	1.24
Sub-flow $_7$	$H^{1,6}$	392	1	0.54
Sub-flow $_8$	$H^{1,7}$	440	1	0.61

[a] $\overline{t_{TXOP}} = 1.92$ msec, $N_{QSTA} = 10$.

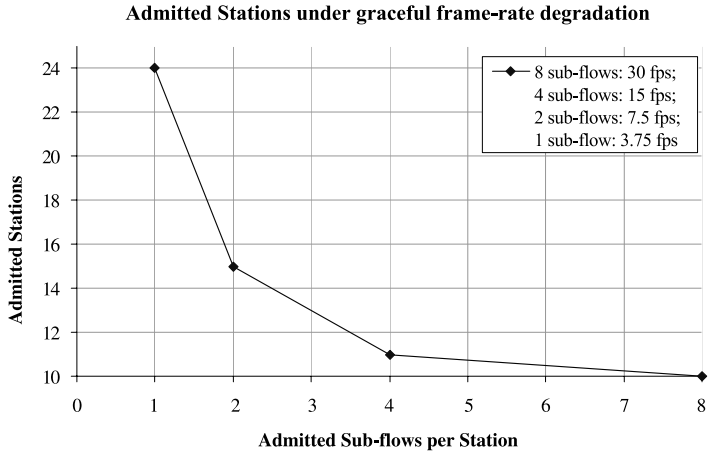


FIGURE 12.13: A reduction of the number of admitted sub-flows results in a dyadically reduced frame rate. However, the number of admitted stations increases. The utilized video sequences were encoded at 2048 kbps.

tual video quality for different channel conditions. This flexibility can be easily provided using the sub-flow concept.

In summary, we observe that a higher number of stations can be admitted given the same channel condition if the sub-flow case is used, as compared to the global flow case. Note that the same video bit streams are transmitted in both cases and that no losses are incurred due to the use of sub-flows.

For more details on optimized resource management for video transmission over 802.11e, the interested reader is referred to [3,16,23].

12.8 SIMPLIFYING THE REAL-TIME CROSS-LAYER OPTIMIZATION PROBLEM USING CLASSIFICATION

In previous sections, several joint optimizations across the various layers of the protocol stack have been discussed for improving the performance of real-time video transmission over wireless networks. However, the complexity associated with performing the cross-layer optimization in real time is very high. Thus, low complexity systems are required for determining the optimal cross-layer strategies in real time whenever a packet needs to be transmitted. Next, we discuss such a possible approach based on classification and machine learning techniques [19].

For illustration purposes, this section focuses on determining the optimal MAC retry limit for each video packet given the maximum available bit rate ($R_{\max}$),

the maximum tolerable delay $Delay_{\max}$, and the experienced bit error rate (P_e), similar to the problem investigated in Sections 12.5 and 12.6. However, classification techniques could also be used successfully to simplify other cross-layer optimizations.

12.8.1 Classification System for Cross-Layer Optimization

The classification-based wireless video transmission system is depicted in Figure 12.14. It consists of an off-line training module followed by online processing. The former includes modules for class definition and classifier learning, whereas the latter mainly involves classification and real-time cross-layer strategy prediction for video packets. The major steps in the approach are as follow.

Step 1: Generate ground truth (off-line) First, a set of packets from a variety of video sequences is collected under different representative channel conditions and the entire set of cross-layer strategies available at the wireless station is identified. For each packet in this training set and collection of encoding parameters, the compressed-domain content features (CF) and packet types (PT) determined based on the specific encoder configuration/parameter set are extracted [32]. Used feature sets should also include the wireless channel conditions (WCC): $R_{\max}$[12] and P_e. Subsequently, the optimal strategy s^{opt} resulting in the best quality for the

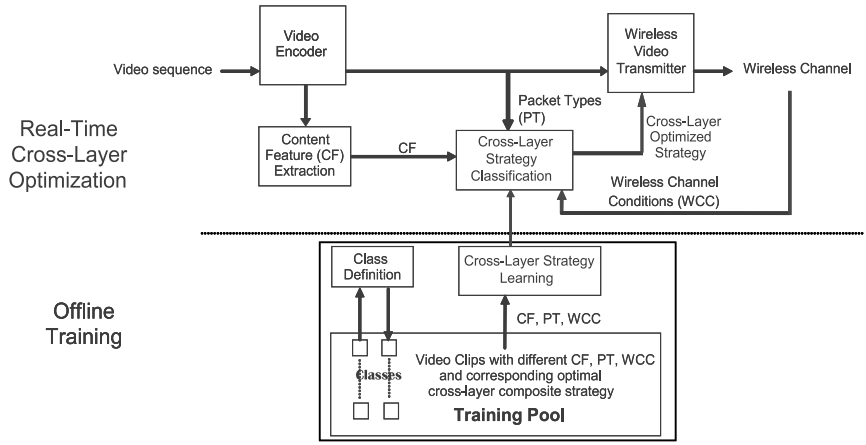


FIGURE 12.14: Classification-based cross-layer system for wireless video.

[12] $R_{\max}$ can be determined based on the video encoding rate, delay constraint, and the PHY rate used for transmission (determined based on the modulation strategy, etc.) [26].

different training sequences, packet types, and channel conditions is determined using dynamic programming (see Section 12.6).

Step 2: Train classifier (off-line) The key is to determine, for each packet j, a mapping from the composite feature vector $\mathbf{f}(j)$ to class label l_j, corresponding to a specific optimal strategy s^{opt}. During training, supervised clustering methods are used to map the composite features to the corresponding class label. Two different classification strategies aimed at minimizing the probability of misclassification and minimizing the cost of misclassification (in terms of video distortion), respectively, are discussed, for illustration purposes.

Step 3: Real-time strategy selection based on classification The optimal strategy s^{opt} for a sequence of incoming video packets given the instantaneous wireless channel conditions/characteristics can be then determined, by the trained classifier, on a packet-by-packet basis using the composite feature vectors. The selected strategy is used to determine the optimized parameters and configurations of the wireless multimedia system.

The various steps just outlined are described in more detail in subsequent sections.

12.8.2 Feature Selection

For video packet j, a suitable feature vector $\mathbf{f}(j)$ needs to be identified that can predict the optimal decision with low complexity (i.e., with features that can be computed/extracted easily at run time). The WCC features P_e (equivalently P_L) and $R_{\max}$ can be determined in real time based on information that can be extracted easily from the wireless card driver. For example, the transmitter can use the received signal strength indicator (RSSI) of previously received MAC acknowledgment frames, as well as MAC acknowledgment reports to determine these features [22]. A more detailed description of the various features that can be extracted in real time from the lower layers of the transmission system can be found in Chapter 13.

Among CF, the packet energy can be selected, which may be used to distinguish among sequences with different levels of spatio-temporal detail. The energy for packet j is calculated by summing up the squared wavelet coefficients *coeff* belonging to the packet,

$$E_j = \sum_{i \in W_j} \big(coeff(i)\big)^2, \tag{12.56}$$

where W_j is the set of decoded coefficients (collected from all the decoding units within the packet) belonging to packet j. At encoding time, the energy of each decoding unit is computed for the various quality layers. During transmission,

the packet energy can simply be computed by aggregating the relevant energies corresponding to the target bit rate.

The codec-specific (PT) features include the spatial and temporal level of the data in the packet. This is because packets belonging to distinct spatiotemporal bands have a different impact on the overall distortion and require different protection: the retry limit of the packet decreases with an increasing spatiotemporal level. Most ad hoc cross-layer strategies are based on this simple classification criterion for selecting the retry limit, that is, the spatiotemporal level (or frame type for conventional video coders). However, these schemes do not use either the content characteristics or the channel conditions, which directly impact the optimal retry limit. In order to test the suitability of the CF and PT features,[13] the correlation coefficient is computed between them and the optimal decision sequence (i.e., the choice of optimal retry limit per packet). If feature i from packet j is called $\mathbf{f}_i(j)$ and the optimal retry limit (decision) for this packet is $T^{opt}(j)$, then the correlation coefficient between the feature sequence and the decision sequence may be defined

$$\rho_i = \frac{\sum_{j=1}^{N_p} \mathbf{f}_i(j) T^{opt}(j)}{\sqrt{\sum_{j=1}^{N_p} (\mathbf{f}_i(j))^2 \sum_{j=1}^{N_p} (T^{opt}(j))^2}}, \tag{12.57}$$

where N_P is the number of packets in the GOP.

Table 12.7 shows the correlation coefficients for these different features with the optimal retry limit for the *Mobile* sequence. Similar results were obtained for other video sequences. The large values (close to 1) of the coefficients ρ_i in Table 12.7 show that for given channel conditions, the selected features are well correlated with the optimal decision sequence.

Table 12.7: *Mobile*: Correlation coefficients ρ_i.

Feature	$R_{\max} = 512$ kps			
	$P_L = 1\%$	$P_L = 3\%$	$P_L = 5\%$	$P_L = 10\%$
Packet energy	0.79	0.82	0.82	0.83
Temporal level	0.76	0.86	0.90	0.93
Spatial level	0.62	0.73	0.75	0.80
Feature	$R_{\max} = 1024$ kps			
	$P_L = 1\%$	$P_L = 3\%$	$P_L = 5\%$	$P_L = 10\%$
Packet energy	0.75	0.80	0.81	0.81
Temporal level	0.72	0.82	0.85	0.89
Spatial level	0.60	0.67	0.73	0.76

[13] We exclude the WCC features as they are common to all the packets.

Table 12.8: Pair-wise MI for the chosen feature set.

	Packet energy	Temporal level	Spatial level	$R_{\max}$	P_L
Packet energy	3.4	1.37	1.34	1.3	1.19
Temporal level	1.37	1.90	0.07	0.03	0
Spatial level	1.34	0.07	2.12	0.01	0
$R_{\max}$	1.30	0.03	0.01	1.57	0
P_L	1.19	0	0	0	2

Table 12.9: Accuracy of the classifier based on a single feature.

	Packet energy	Temporal level	Spatial level	P_L	$R_{\max}$
Percentage of accuracy	52%	58%	48%	52%	48%

In order to examine the redundancy in the feature set, metrics such as the mutual information[14] (MI) between pairs of features can be computed. An illustrative example is presented in Table 12.8. We can see from Table 12.8 that while there is some redundancy among the features, especially between packet energy and the rest of the features, each feature contains nonredundant information (for a majority of cases, the MI is significantly lower than the feature entropy). Allied with this is the fact that none of these features are computationally complex to determine, and hence we use the complete set of features in the system.

Finally, in order to validate these features for the actual classification task, we can also examine the classifier performance with each of these individual features. Table 12.9 shows the classifier accuracy results with each individual feature.

The temporal level feature leads to the best classification performance, whereas the video rate and the spatial level have the worst classification performance. This knowledge can be used to design an ad hoc strategy (similar to that used in [11, 33,34] but for different video coders) to determine the packet retransmission limits. Finally, since these features are used jointly, the classifier accuracy increases to ∼83%. While additional features can be used (e.g., the motion vectors, available bits per frame, and number of bit planes per frame at various bit rates), these will increase the complexity of the real-time system with only limited possible improvement in the classification performance. See [19] for an extensive study.

In summarizing, the feature extraction step needs to be kept at a low complexity because it is also performed online. Consequently, content and encoder-specific features can be selected that are already computed during the encoding process (i.e., no additional complexity is needed for feature extraction). These features can be prestored in metadata files together with the video bit streams. Hence, at

[14]The MI between two random variables X and Y with distributions $p(x)$ and $p(y)$ and joint distribution $p(x, y)$ is defined as $MI(X, Y) = \sum_x \sum_y p(x, y) \log \frac{p(x,y)}{p(x)p(y)}$.

transmission time, only the channel features need to be determined based on the RSSI and MAC acknowledgment frames. These values can be accessed readily from device drivers of existing wireless cards (e.g., Intel PRO/Wireless 2915ABG Network Connection and Intel PRO/Wireless 2200BG Network Connection mini PCI adapters). A more detailed discussion on determining and monitoring the channel quality can be found in Chapter 13.

12.8.3 Classifier Design

The cross-layer optimization problem involves assigning a retry limit to each packet such that the expected overall decoded distortion is jointly minimized. Let us assume that there are M available retry limits $\{X_1, \ldots, X_M\}$ (i.e., an M-class classification problem) and N_t packets in the training set. From data within each packet j, we extract an F-dimensional feature vector $\mathbf{f}(j) \in \mathbb{R}^F$ (in this case $F = 5$). The classifier is then provided with feature vectors $\mathbf{f}(j)$, $1 \leq j \leq N_t$, and the associated optimal retry limit $T^{opt}(j) \in \{X_1, \ldots, X_M\}$ for each packet. The classifier then partitions the feature space $\mathbb{R}^F$ into M-nonoverlapping regions $G_1, \ldots, G_M$, with region G_i associated with a unique optimal retry limit X_i, such that the error in classification (i.e., probability of misclassification) on training data is minimized. This may be written

$$\{G_1, \ldots, G_M\}^{opt} = \underset{G_1,\ldots,G_M}{\arg\min} \sum_{j=1}^{N_t} \left[1 - B\left(\mathbf{f}(j) \in G_i \mid T^{opt}(j) = X_i\right)\right], \quad (12.58)$$

where $B(\mathbf{f}(j) \in G_i \mid T^{opt}(j) = X_i)$ is a binary-valued function that takes value 1 when vector $\mathbf{f}(j)$ is classified correctly, that is, if $\mathbf{f}(j)$, with optimal retry $T^{opt}(j)(= X_i)$ is inside region G_i, and zero otherwise.

While minimizing the previously defined classification error, the classifier views all feature vectors in the training set equivalently. However, in reality, the feature vectors do have different importance because misclassifying different feature vectors can lead to different penalties in the total distortion. Hence, since minimizing the decoded distortion is the final goal of the optimization, the classifier needs to be modified to take the distortion impact into account. Let the importance of packet j with feature vector $\mathbf{f}(j)$ be determined by the cost of misclassifying it $C(j)(\geq 0)$. We will discuss this cost in more detail later. Then, the classifier design problem may be written

$$\{G_1, \ldots, G_M\}^{opt} = \underset{G_1,\ldots,G_M}{\arg\min} \sum_{j=1}^{N_t} C(j)\left[1 - B\left(\mathbf{f}(j) \in G_i \mid T^{opt}(j) = X_i\right)\right] \quad (12.59)$$

or, alternatively,

$$\{G_1, \ldots, G_M\}^{opt} = \arg\min_{G_1,\ldots,G_M} \sum_{j=1}^{N_t} \sum_{k=1}^{C(j)} \big[1 - B\big(\mathbf{f}(j) \in G_i \mid T^{opt}(j) = X_i\big)\big]. \tag{12.60}$$

This optimization has the same form as the one before, where instead of providing the classifier with vector $\mathbf{f}(j)$, we provide it vector $\mathbf{f}(j)$ repeated $C(j)$ times.[15] Hence, by modifying the training set in such a manner, the minimized classification error classifier can be used to minimize the cost of misclassification.

The cost of misclassification $C(j)$ in the cross-layer problem needs to be defined in terms of the increase in distortion when packet j is assigned the wrong retry limit. Hence, when instead of this optimal retry limit, a different, suboptimal, retry limit X_k is assigned, the corresponding increase in the incurred distortion is $(\bar{D}(X_k, j) - \bar{D}(T^{opt}(j), j)) \geq 0$. Hence, the total cost $C(j)$ of misclassifying the packet, in terms of distortion, may be computed

$$C(j) = \sum_{\substack{k=1 \\ X_k \neq T^{opt}(j)}}^{M} \big(\bar{D}(X_k, j) - \bar{D}\big(T^{opt}(j), j\big)\big). \tag{12.61}$$

For the classification, a supervised nonparametric classification technique, such as support vector machines, can be adopted.

12.8.4 Validation Experiments

In [19], the efficiency of the discussed classification-based system was validated using a real wireless streaming test bed. The performance of the classification-based cross-layer strategy is compared against the optimal exhaustive strategy for these real wireless channel traces in Table 12.10. Results demonstrate that under varying SNR, the cross-layer mechanism leads to a decrease in PSNR of $\sim$0.7 dB as compared with the optimal strategy. The obtained classification-based results outperformed by 3–5 dB current ad hoc retransmission strategies available in the wireless card. A thorough validation study can be found in [19]. Also, a thorough description of how such real-time middleware systems can be designed and implemented can be found in Chapter 13.

[15] In general it is not necessary that all the costs $C(j)$ are integers; however, without loss of generality we can scale them appropriately to make them integers.

Table 12.10: Decoded PSNR for real wireless packet loss traces.

Measured channel SNR	Foreman PSNR (dB)		Mobile PSNR (dB)	
	Classification	Exhaustive	Classification	Exhaustive
Poor channel conditions (12–15 dB)	33.81	34.29	26.31	26.50
Average channel conditions (15–20 dB)	35.90	36.06	28.48	29.26
Very good channel conditions (20–25 dB)	38.64	38.66	31.82	32.01

12.9 DYNAMIC AND FAIR MULTIUSER WIRELESS TRANSMISSION

In the previous sections, the time allocation among the various WSTAs was statically performed, that is, once for the entire duration of the flows (cf. Section 12.7, where the TSPEC negotiation was only performed initially). This static resource (transmission opportunities) management is inefficient because it does not scale with the number of users, channel conditions, video characteristics, and so on. Alternatively, a dynamic resource allocation enables the time allocation to be performed repeatedly, every SI or group of SIs depending on the channel condition, cross-layer strategy, and used fairness policy. Moreover, as a result of the static TXOP allocation, until now, we only considered the problem of cross-layer optimization in isolation, at each WSTA. However, in wireless multimedia transmission systems, the cross-layer strategies adopted by the various WSTAs impact other competing stations. If a WSTA is adapting its transmission strategy, the delay and throughput of the competing stations are affected and, as a consequence, they may need to adjust their own strategies. (See [12] for several such examples.) Hence, the cross-layer strategies adopted by a station should not be optimized in isolation, but should also consider the system-wide availability of resources and "fairness" issues.

12.9.1 Why Are Current Fairness Strategies Not Suitable for Cross-Layer-Optimized Multimedia Transmission?

The objective of fair scheduling is to provide multimedia applications with different amounts of "work" (resources) proportional to their requirements in terms of bandwidth, delay, and packet-loss rates. Usually, "work" is measured by the amount of data transmitted (either in number of bytes or in packets/frames) during a certain period of time. Let $W_i(t_1, t_2)$ be the amount of video flow i's traffic served in a time interval (t_1, t_2) and ϕ_i be its corresponding weight based on its requirements. Then, an ideal fair scheduler (i.e., the Generalized Processor Sched-

uler [27]) for N WSTAs (and their flows) can be defined

$$\frac{W_i(t_1,t_2)}{W_j(t_1,t_2)} \geq \frac{\phi_i}{\phi_j}, \quad j = 1,2,\ldots,N \tag{12.62}$$

for any multimedia flow i that is continuously backlogged during (t_1,t_2). [Backlogged means that flow i has frames in its buffer during the specified time interval (t_1,t_2).] If all multimedia flows are transmitted at a fixed rate, we can obtain from (12.62),

$$\frac{W_i(t_1,t_2)}{t_2-t_1} \geq \frac{\phi_i}{\sum_j \phi_j} r, \tag{12.63}$$

where r is the physical transmission rate or the total channel capacity. Thus, each multimedia flow i is guaranteed to have the throughput given by (12.63) regardless of the states of the queues and frame arrivals of the other flows. However, the advantages of using GPS, such as the guaranteed throughput and independent service, cannot be preserved if the flows are deploying different cross-layer optimization, resulting in different transmission rates. Depending on the channel condition, or their distance from the access point, WSTAs may choose, as discussed in prior sections, different cross-layer transmission strategies (PHY modes, retry limits, packet sizes, etc.) to ensure optimized multimedia quality. Determining a "*fair share of resource*" among WSTAs in such a transmission scenario is a very challenging problem because serving an equal amount of traffic from individual stations deploying different strategies requires allocation of various amounts of airtime and results in different impacts on the multimedia quality.

12.9.2 Time Fairness

To obtain a better allocation of resources that explicitly considers the various deployed cross-layer strategies, time fairness was proposed in [12,28]. For this, the total throughput degradation due to WSTAs deploying different cross-layer strategies (e.g., different PHY rates) in the WLAN network can be computed. Given n WSTAs (with all stations having the same frame size), with $n_i (\sum_{i=1}^{8} n_i = n)$ operating at, for example, PHY mode i $(= 1,\ldots,8)$, the throughput degradation can be determined

$$\textit{Throughput} = \frac{1}{(1/n)\left(\sum_{j=1}^{8} n_j/R_j\right)}. \tag{12.64}$$

WSTAs having different transmission rates R_j due to the different PHY modes or other deployed cross-layer optimization strategies cause this unwanted degradation. Time fairness tries to alleviate this problem by allocating each WSTA a fair

share of time (i.e., a percentage of the SI), which is proportional to the requirements mentioned in their TSPEC (see Section 12.7), rather than guaranteeing a specific bandwidth (rate requirement). This proportional time allocation (e.g., allocated to a stream at admission time) removes part of the unfairness resulting from the deployment of different cross-layer strategies by the various WSTAs. Equation (12.62) can be thus rewritten to provide time fairness

$$\frac{T_i(t_1, t_2)}{T_j(t_1, t_2)} \geq \frac{\phi_i}{\phi_j}, \quad j = 1, 2, \ldots, N, \tag{12.65}$$

where T_i, and T_j represent the time allocated to the streams i and j, respectively.

The advantages of this airtime fair scheduler (AFS) as opposed to the conventional weighted fair queuing for multimedia transmission are analyzed in detail in [12]. AFS isolates the channel and differential transmission rates of the various WSTAs, thus guaranteeing a better multimedia performance across all participating stations.

12.9.3 Multimedia Quality Fairness

While time fairness is efficient for a variety of applications, multimedia users do require a different type of fairness, which quantifies the resulting resource allocation in terms of the utility impact rather than the consumed time resources. For instance, a resource manager (access point) can decide to implement a policy where the users derive either the same video quality or the same quality penalty, independent of the experienced channel conditions or deployed cross-layer strategies.

For this, bargaining solutions from game theory were deployed successfully in [25,57] to allocate utilities to selfish WSTAs fairly and optimally by *directly* considering the relationships in terms of utility resulting from various resource allocations according to different fairness policies. The Kalai–Smorodinsky bargaining solution, Egalitarian bargaining solution, and Nash bargaining solution were used in [25,57] in order to enforce different fairness policies among users.

Unlike traditional fairness approaches, such as the proportional fairness introduced by Kelly [58] and Kelly *et al.* [59] and the time fairness and GPS fairness solutions discussed earlier where the resulting relationships between the users' utilities cannot be guaranteed, utility-based bargaining solutions do ensure that certain relationships in terms of utilities are satisfied. For instance, as was shown in [25,57], the Kalai–Smorodinsky bargaining solution ensures that the participating WSTA incur the same drop in multimedia quality (PSNR drop) as compared to a maximum desirable video quality (see Figure 12.15 for a simple illustration of the feasible utility set of two WSTA and the resulting Kalai–Smorodinsky bargaining solution), whereas the Egalitarian bargaining solution guarantees that the

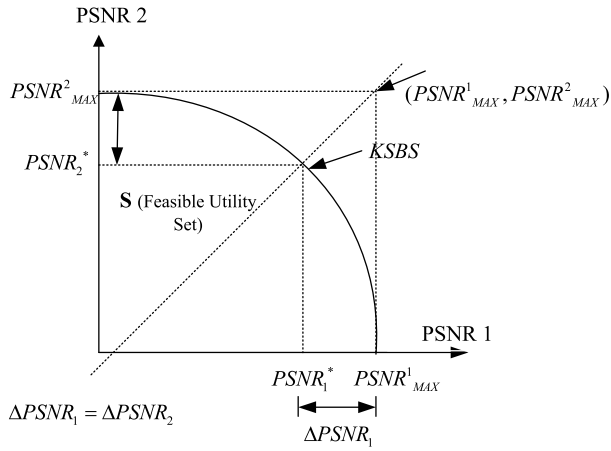


FIGURE 12.15: Utility-based fairness based on the Kalai–Smorodinsky bargaining solution.

users will have the same video quality independent of their cross-layer strategies or channel conditions. After a socially fair allocation in terms of utilities is derived using the bargaining solution, the corresponding resources (i.e., time opportunities) are determined and the user can start the actual transmission. The resource manager can also assign different bargaining powers to the various WSTAs.

Moreover, unlike conventional optimization solutions, the various bargaining solutions can be differentiated based on the axioms (properties) that they fulfill. These axioms are essential for a fair resource allocation among multimedia users. For instance, as shown in [25,57], the unique axiom of *individual monotonicity* of the Kalai–Smorodinsky bargaining solution guarantees that increasing the maximum achievable utility in a direction favorable to a WSTA (e.g., by deploying a more sophisticated cross-layer strategy) always benefits that WSTA. This property implicitly means that the Kalai–Smorodinsky bargaining solution encourages each WSTA to maximize its achievable utility and then allocates the resources based on its maximum achievable utility. This is especially useful for modeling the selfish user behavior of the WSTAs transmitting delay-sensitive multimedia.

12.9.4 Game-Theoretic Dynamic Resource Management

The static resource allocation discussed in Sections 12.4 and 12.7 represents the current, conventional approach for resource allocation. However, since the channel conditions, video characteristics, number of participating WSTAs, or even the user desired utility varies over time, the conventional resource allocation (e.g., 802.11e) does not exploit the network resources efficiently and does not provide

adequate QoS support for multimedia transmission, especially when the network is congested. Also, importantly, the WSTA can untruthfully declare (exaggerate) its resource requirements during the initialization stage in order to obtain a longer transmission time t_i. Thus, in existing wireless networks, there is no mechanism available to prevent the WSTA from lying about the required t_i.

To eliminate the aforementioned limitations for multiuser wireless multimedia transmission, in [55,56] WSTAs were enabled to dynamically acquire wireless resources depending on the desired utility, their available cross-layer strategies, and private information. Specifically, in [55,56], the multiuser wireless communication is modeled as a noncollaborative resource management game regulated by the access point, referred to here as the Central Spectrum Moderator (CSM), where the WSTAs are allowed to dynamically compete for the available TXOPs by jointly adapting their cross-layer strategies and their willingness-to-pay and risk attitude. In this noncollaborative game, WSTAs are considered selfish (autonomous) users that solely aim at maximizing their own utilities by gathering as many resources as possible.

To prevent WSTAs from misusing the available resources, the CSM adopts a tool from mechanism design, referred to as *transfer*, to penalize WSTAs from exaggerating their resource requirements. Specifically, in [55,56], the Vickrey–Clarke–Groves (VCG) mechanism was used to implement and enforce the "rules" of the resource allocation game. In the VCG mechanism, the resource allocation is based on a "social decision," which maximizes the aggregated multiuser wireless system utility. To encourage the WSTAs to work in this social optimal way, the CSM charges WSTA a transfer corresponding to the inconvenience it causes to other WSTAs. In the noncollaborative wireless network of [55,56], the inconvenience caused by a WSTA is quantified as the utility penalty (drop) that the competing WSTAs incur due to the participation (resource usage) of that WSTA in the resource management game. In the formulation, the performance of each WSTA will depend on the private information, the adopted cross-layer strategy, and the WSTA willingness to pay for resources. The willingness to pay, denoted as $\boldsymbol{w}_i$, will affect the ability of a WSTA i to transmit more or less video data during the current SI by accepting to pay a larger/lower transfer. Details of how the willingness-to-pay $\boldsymbol{w}_i$ affects the strategy with which the WSTA plays the resource game and its derived utility and incurred transfer can be found in [55,56], as well as the details of the VCG mechanism deployed at the CSM side.

Implementation of the resource allocation game is depicted pictorially in Figure 12.16. In the resource game, a *joint strategy* is defined for WSTA i that consists of selecting an *expected cross-layer strategy* $\bar{s}_i \in \boldsymbol{\mathcal{S}}_i$ and a *revealing strategy* $\mu_i \in \boldsymbol{\mathcal{V}}_i$, where $\boldsymbol{\mathcal{V}}_i$ is the set of revealing strategies available to WSTA i. We denote the joint strategy as $\kappa_i = (\bar{s}_i, \mu_i), \kappa_i \in \boldsymbol{\mathcal{S}}_i \times \boldsymbol{\mathcal{V}}_i$. The purpose of the expected cross-layer strategy and the revealing strategy is outlined in subsequent paragraphs.

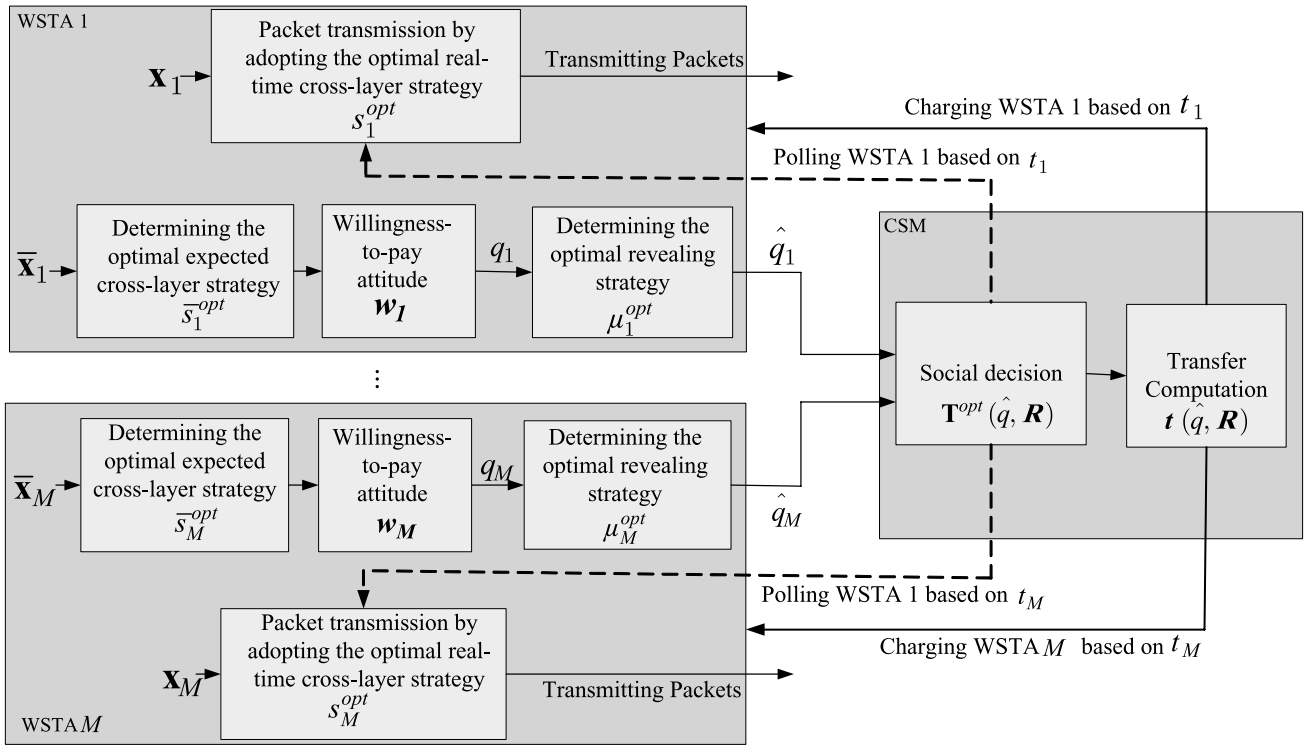


FIGURE 12.16: Mechanism design framework for the multiuser wireless video resource allocation game.

The expected cross-layer strategy $\bar{s}_i$ is computed by the WSTA i *prior* to the transmission time in order to determine the *expected* benefit in terms of utility that it can derive by acquiring available resource during the upcoming SI. Note that the expected cross-layer strategy $\bar{s}_i$ is proactively decided at the beginning of every SI and will not be exactly the same as the actual real-time strategy s_i adopted at transmission time. The reason for this is that the strategy for playing the game also depends on the WSTA's private information $\mathbf{x}_i$. Unlike the real-time cross-layer strategy, which has precise information about $\mathbf{x}_i$, the expected cross-layer strategy will need to determine the modulation mode at the PHY layer, the number of retransmissions per packet at the MAC layer, the packet prioritization and scheduling at APP layer, and so on based on the expected private information $\bar{\mathbf{x}}_i$.

To play the resource management game, each WSTA i needs to announce its "type," denoted as $\theta_i(\bar{s}_i, \bar{\mathbf{x}}_i, \boldsymbol{w}_i)$, which represents the utility that can be derived from the potentially allocated resources (TXOPs). Based on the announced types, the CSM will determine the resources allocation and transfers for the participating WSTAs. We refer to the set of possible types available to WSTA i as Θ_i. The type is defined as a nominal vector that encapsulates the expected private information $\bar{\mathbf{x}}_i$, the expected cross-layer strategy $\bar{s}_i$, and the willingness-to-pay $\boldsymbol{w}_i$ for resources (transfers). The type profile for all WSTAs is defined as $\theta = (\theta_1, \ldots, \theta_M)$, with $\theta \in \boldsymbol{\Theta}$, $\boldsymbol{\Theta} = \Theta_1 \times \cdots \times \Theta_M$. For more details on this, the reader is referred to [55,56].

A revealing strategy μ_i is adopted by the WSTA i to determine which type should be declared to the CSM based on the derived real type θ_i. The type of WSTA i revealed to the CSM (referred to as announced type) can be computed as $\hat{\theta}_i = \mu_i(\theta_i)$. The announced type profile for all WSTAs is denoted as $\hat{\theta} = (\hat{\theta}_1, \ldots, \hat{\theta}_M)$. In other words, the joint strategy κ_i adopted by WSTA i determines the announced type $\hat{\theta}_i$, that is, $\hat{\theta}_i = \kappa_i(\bar{\mathbf{x}}_i, \boldsymbol{w}_i) = \mu_i(\theta_i(\bar{s}_i, \bar{\mathbf{x}}_i, \boldsymbol{w}_i))$.

For the dynamic resource allocation game, the outcome is denoted as $\mathbf{T}(\hat{\theta}, \boldsymbol{\mathcal{R}})$, where $\mathbf{T} : \boldsymbol{\Theta} \times \mathbb{R}_+ \to \mathbb{R}_+^M$ is a function of both the announced type profile $\hat{\theta}$ and the available resource $\boldsymbol{\mathcal{R}}$. Thus, $\mathbf{T}(\hat{\theta}, \boldsymbol{\mathcal{R}}) = [t_1, \ldots, t_M]$, where t_i denotes the allocated time to WSTA i within the current SI and $\sum_{i=1}^{M} t_i \leq t_{SI}$. Based on the dynamic resource allocation t_i and its derived type θ_i, WSTA i can derive utility $u_i(t_i, \theta_i)$. However, the utility computed at the CSM side for WSTA i is $u_i(t_i, \hat{\theta}_i)$, as this is determined based on the announced type $\hat{\theta}_i$. Note that t_i is decided by the CSM, which is a function of the announced type profile $\hat{\theta}$ and the available resource $\boldsymbol{\mathcal{R}}$. Hence, note that the "real" utility derived by a WSTA and the utility that a CSM believes that the WSTA is obtaining can differ, as the CSM solely relies on the information announced by the WSTA. In the resource management game, the utility is computed not only based on the expected received video quality such as in the conventional cross-layer design, but also on the willingness to pay for resources of a WSTA, $\boldsymbol{w}_i$. The transfer computed by the CSM is represented by $\boldsymbol{\tau}(\hat{\theta}, \boldsymbol{\mathcal{R}})$, where $\boldsymbol{\tau} : \boldsymbol{\Theta} \times \mathbb{R}_+ \to \mathbb{R}_-^M$ is a function of both the announced

type profile $\hat{\theta}$ and the available resource $\boldsymbol{\mathcal{R}}$, and $\boldsymbol{\tau}(\hat{\theta}, \boldsymbol{\mathcal{R}}) = [\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_M]$, where $\boldsymbol{\tau}_i$ denotes the transfer that WSTA i needs to pay during the current SI. By participating in the resource allocation game, WSTA i gains the "payoff" $\upsilon_i(\hat{\theta}, \theta_i, \boldsymbol{\mathcal{R}}) = u_i(t_i, \theta_i) + \tau_i$, which is always nonnegative in the VCG mechanism.

In summary, the following dynamic, game-theoretic resource allocation at the CSM side can be implemented during each SI.

1. **Social decision**: After receiving the announced type profile $\hat{\theta} = (\hat{\theta}_1, \ldots, \hat{\theta}_M)$ from the WSTAs, the CSM decides the resource allocation $\mathbf{T}(\hat{\theta}, \boldsymbol{\mathcal{R}})$ such that the multiuser wireless system utility (i.e., the sum of utilities of all WSTAs) is maximized.
2. **Transfer computation**: Next, it computes the transfers $\boldsymbol{\tau}(\hat{\theta}, \boldsymbol{\mathcal{R}})$ associated with this resource allocation to enforce the WSTA to reveal their real type truthfully.
3. **Polling WSTAs**: The CSM polls the WSTAs for packet transmission according to the allocated time.

At the WSTAs side, the subsequent steps are performed by WSTA i in order to play the resource management game.

1. **Private information estimation**: Each WSTA i estimates the expected private information $\bar{\mathbf{x}}_i$, which includes the expected video source characteristics $\bar{\xi}_i$ and channel conditions in terms of $\overline{SNR}_i$.
2. **Selection of optimal joint strategy and corresponding "type"**: Based on the private information, WSTA i determines the optimal joint strategy to play the resource allocation game, that is,

$$\begin{aligned} \kappa_i^{opt} = \left(\bar{s}_i^{opt}, \mu_i^{opt}\right) &= \underset{\kappa_i = (\bar{s}_i, \mu_i) \in \boldsymbol{\mathcal{S}}_i \times \boldsymbol{\mathcal{V}}_i}{\arg\max} \upsilon_i\left(\hat{\theta}, \theta_i, \boldsymbol{\mathcal{R}}\right) \\ &= \underset{\kappa_i = (\bar{s}_i, \mu_i) \in \boldsymbol{\mathcal{S}}_i \times \boldsymbol{\mathcal{V}}_i}{\arg\max} \left\{u_i(t_i, \theta_i) + \tau_i\right\} \\ &\quad \text{s.t.} \quad Delay(t_i, \theta_i) \le Delay_i^{\max}. \end{aligned} \tag{12.66}$$

Note that the WSTA i cannot explicitly solve the optimization problem just given because both the resource allocation t_i and the transfer τ_i depend on the announced types of the other WSTAs, which are not known by this station. However, in [56] it was proven that whenever the VCG mechanism is used, *the optimal joint strategy* can be simply determined by first proactively selecting *the optimal expected cross-layer strategy* $\bar{s}_i^{opt}$ that maximizes the expected received video quality without considering the impact of the other WSTAs. Then, based on this, *the optimal revealing strategy* μ_i^{opt} through which the real (truthful) type (including willingness-to-pay attitude) is revealed, that is, $\hat{\theta}_i = \mu_i^{opt}(\theta_i) = \theta_i$. Details of the expected cross-layer strategy, revealing strategy and type computation, are presented in [56].

1. **Reveal the type to CSM**: The determined type $\hat{\theta}_i$ is declared by each WSTA to the CSM.
2. **Transmit video packets**: When polled by the CSM, each WSTA i determines and deploys the *optimal real-time cross-layer strategy* s_i^{opt} for video transmission that maximizes the expected received video quality. This cross-layer strategy is determined as discussed earlier.

Note that while the transfers are computed for each WSTA during every SI, the CSM can communicate and charge the WSTA the incurred (cumulative) transfer every couple of SIs. Various charging mechanisms and protocols can be used for this purpose.

In summarizing, to play the dynamic resource management game, WSTAs deploy three different types of strategies at different stages of the transmission: the optimal expected cross-layer strategies and the revealing strategies (prior to the actual transmission, in order to determine the announced type) and the optimal real-time cross-layer strategy (in real time, during the actual transmission). Hence, the cross-layer-optimized transmission strategies become the "smartness" with which WSTAs play the competitive, dynamic, resource management game.

12.10 SUMMARY AND FURTHER READING

Most existing wireless transmission algorithms and standards have been designed in an application agnostic fashion [1–3], as this guarantees their durability, adaptability, and generality. Research in cross-layer optimization has shown that the performance of existing wireless protocols can be improved by jointly optimizing the various layers; importantly, it was also shown that cross-layer optimization could catalyze the development of new protocols, with enhanced support for multimedia applications, while preserving these properties (e.g., 802.11e [3]).

This chapter showed that establishing communication mechanisms between OSI layers to convey application-layer information to lower layers (e.g., packet sizes, relative importance, different interrelationships, arrival rates, and delay constraints) and channel condition information to the application layer, as well as to enable resource and information exchanges among stations, can lead to an important improvement in the system efficiency and individual quality of the participating stations. We focused on multimedia applications that can operate at multiple quality levels and have different delay requirements, thereby enabling the study of different communication trade-offs. The cross-layer optimization problem was formulated, and several solutions based on queuing theory, Lagrange optimization, and classification were discussed. Moreover, the benefits in terms of multimedia quality of employing a cross-layer-optimized framework for different multimedia applications with different delay sensitivities and loss tolerances

were quantified. However, the described cross-layer-optimized wireless multimedia paradigm is only recently emerging, and a variety of research topics need still to be addressed. A summary of such topics is presented together with a list of possible future reading.

Realistic integrated models for the delay, multimedia quality, and consumed power of various transmission strategies/protocols need to be developed. An example of such work can be found in [54].

An important extension of the cross-layer design principles, discussed in this chapter for the case of single-hop wireless transmission, is the extension to multiple hops. Video transmission over multihop wireless networks became recently of increasing interest, as such networks provide a flexible, low-cost infrastructure for the deployment of multimedia applications [20,21,24,38,43].

Another important topic of consideration when performing cross-layer optimization and considering its trade-offs is the resulting power. The interested reader is referred to [23,35–37] for several relevant works on this topic. Importantly, the design of the various cross-layer algorithms discussed in this chapter needs to be implemented into flexible, integrated middleware architectures. For more details on the principles, requirements, and solutions that guide such designs, the interested reader is referred to Chapter 13, which presents various designs for middle architectures.

We have also identified a new fairness paradigm for wireless multimedia transmission based on game-theoretic bargaining solutions, which can result in an improved utilization of wireless resources, as well as an enhanced multimedia performance by the participating stations. The interaction between various wireless stations and their cross-layer optimization strategies can be further analyzed based on economics principles such as bargaining and mechanism design. This is achieved by remodeling existing passive resource allocation problems as economics-driven interactions among selfish users competing for a common network resource "market" [25,55]. The outcome of various interactions among selfish users can be analyzed in terms of both dynamics and steady-state equilibrium(s), and mechanisms can be synthesized that achieve new measures of optimality, rationality, and fairness for multiuser communication systems [25,55]. Game-theoretic principles and tools (mechanism design, bargaining theory, equilibrium analysis, competitive analysis, and other microeconomic methods) can be used to model, analyze, and modify such interactions. However, this work is only at its inception and significant research still remains ahead.

ACKNOWLEDGMENTS

The author of this chapter thanks her previous colleagues at Philips Research—Dr. Deepak Turaga (IBM Research), Dr. Sai Shankar (Qualcomm), and Dr. Qiong

Li (Bayer)—and Dr. Yiannis Andreopoulos (postdoc, UCLA) and Mr. Fangwen Fu (Ph.D. student, UCLA) from her research group for their contributions to this chapter. Also, the author acknowledges the NSF Career Award, the Intel Research IT Grant, and, in particular, Dr. Dilip Krishnaswamy (Intel) for their support of the cross-layer optimized multimedia streaming research performed by the author and her students.

REFERENCES

[1] IEEE Std. 802.11-1999, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, Reference number ISO/IEC 8802-11:1999(E), IEEE Std. 802.11, 1999 edition, 1999.

[2] IEEE Std. 802.11b, Supplement to Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-speed Physical Layer Extension in the 2.4 GHz Band, IEEE Std. 802.11b-1999, 1999.

[3] IEEE 802.11e/D8.0, Draft Supplement to Part 11: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Medium Access Control (MAC) Enhancements for Quality of Service (QoS), November 2003.

[4] V. Kawadia and P. R. Kumar. "A Cautionary Perspective on Cross Layer Design," *IEEE Wireless Communication Magazine*, July 2003.

[5] B. Girod, M. Kalman, Y. Liang, and R. Zhang. "Advances in Channel-Adaptive Video Streaming," *Wireless Communications and Mobile Computing*, vol. 2, no. 6, pp. 549–552, September 2002. (Invited)

[6] E. Setton, T. Yoo, X. Zhu, A. Goldsmith, and B. Girod. "Cross-Layer Design of Ad-Hoc Networks for Realtime Video Streaming," *IEEE Wireless Communication Magazine*, pp. 59–65, August 2005.

[7] P. Frossard. "FEC Performances in Multimedia Streaming," *IEEE Communications Letters*, vol. 5, no. 3, pp. 122–124, March 2001.

[8] P. Frossard and O. Verscheure. "Joint Source/FEC Rate Selection for Quality-Optimal MPEG-2 Video Delivery," *IEEE Transactions on Image Processing*, vol. 10, no. 12, pp. 1815–1825, December 2001.

[9] A. Nosratinia, J. Lu, and B. Aazhang. "Source-Channel Rate Allocation for Progressive Transmission of Images," *IEEE Transactions on Communications*, pp. 186–196, February 2003.

[10] M. van der Schaar, S. Krishnamachari, S. Choi, and X. Xu. "Adaptive Cross-Layer Protection Strategies for Robust Scalable Video Transmission over 802.11 WLANs," *IEEE Journal on Selected Areas of Communications (JSAC)*, vol. 21, no. 10, pp. 1752–1763, December 2003.

[11] Q. Li and M. van der Schaar. "Providing Adaptive QoS to Layered Video over Wireless Local Area Networks through Real-Time Retry Limit Adaptation," *IEEE Trans. on Multimedia*, vol. 6, no. 2, pp. 278–290, April 2004.

[12] M. van der Schaar and S. Shankar. "Cross-layer Wireless Multimedia Transmission: Challenges, Principles and New Paradigms," *IEEE Wireless Communications Magazine*, vol. 12, no. 4, pp. 50–58, August 2005.

[13] D. Turaga, M. van der Schaar, Y. Andreopoulos, A. Munteanu, and P. Schelkens. "Unconstrained Motion Compensated Temporal Filtering (UMCTF) for Efficient and Flexible Interframe Wavelet Video Coding," *EURASIP Signal Processing: Image Communication*, vol. 20, no. 1, pp. 1–19, January 2005.

[14] J. R. Ohm, M. van der Schaar, and J. Woods. "Interframe Wavelet Coding: Motion Picture Representation for Universal Scalability," *EURASIP Signal Processing: Image Communication*, Special issue on Digital Cinema, vol. 19, no. 9, pp. 877–908, October 2004.

[15] M. van der Schaar and Y. Andreopoulos. "Rate–Distortion-Complexity Modeling for Network and Receiver Aware Adaptation," *IEEE Trans. on Multimedia*, vol. 7, no. 3, pp. 471–479, June 2005.

[16] M. van der Schaar, Y. Andreopoulos, and Z. Hu. "Optimized Scalable Video Streaming over IEEE 802.11a/e HCCA Wireless Networks under Delay Constraints," *IEEE Trans. on Mobile Computing*, vol. 5, no. 6, pp. 755–768, June 2006.

[17] M. Wang and M. van der Schaar. "Model-Based Joint Source Channel Coding for Subband Video," *IEEE Signal Proc. Letters*, vol. 13, no. 6, pp. 341–344, June 2006.

[18] M. Wang and M. van der Schaar. "Operational Rate-Distortion Modeling for Wavelet Video Coders," *IEEE Trans. on Signal Processing,* vol. 54, no. 9, pp. 3505–3517, September 2006.

[19] M. van der Schaar, D. Turaga, and R. S. Wong. "Classification-Based System for Cross-Layer Optimized Wireless Video Transmission," *IEEE Trans. on Multimedia*, vol. 8, no. 5, pp. 1082–1095, October 2006.

[20] Y. Andreopoulos, R. Kelarapura, M. van der Schaar, and C. N. Chuah. "Failure-Aware, Open-Loop, Adaptive Video Streaming with Packet-Level Optimized Redundancy," *IEEE Trans. on Multimedia*, vol. 8, no. 6, pp. 1274–1290, December 2006.

[21] Y. Andreopoulos, N. Mastronade, and M. van der Schaar. "Cross-layer Optimized Video Streaming over Wireless Multi-hop Mesh Networks," *IEEE JSAC, special issue on "Multi-hop wireless mesh networks*," vol. 24, no. 11, pp. 2104–1215, November 2006.

[22] M. van der Schaar and D. Turaga. "Cross-layer Packetization and Retransmission Strategies for Delay-Sensitive Wireless Multimedia Transmission," *IEEE Trans. on Multimedia*, January 2007.

[23] S. Shankar and M. van der Schaar. "Performance Analysis of Video Transmission over IEEE 802.11a/e WLANs," *IEEE Trans. on Vehicular Technolog*, 2007.

[24] N. Mastronade, D. Turaga, and M. van der Schaar. "Collaborative Resource Exchanges for Peer-to-Peer Video Streaming over Wireless Mesh Networks," *IEEE JSAC*, Special issue on Peer-to-Peer communications and Applications, 2007.

[25] H. Park and M. van der Schaar. "Bargaining Strategies for Networked Multimedia Resource Management," *IEEE Trans. on Signal Proc.*, 2007.

[26] Daji Qiao, Sunghyun Choi, and Kang G. Shin. "Goodput Analysis and Link Adaptation for IEEE 802.11a Wireless LAN," *Proc. of IEEE Trans. on Mobile Computing*, vol. 1, no. 4, 2002.

[27] A. K. Parekh and R. G. Gallager. "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case," *Proc. of IEEE/ACM Trans. on Networ.*, vol. 1, June 1993.

[28] M. van der Schaar and S. Shankar. "New Fairness Paradigms for Wireless Multimedia Communication," *Proc. IEEE ICIP*, Vol. 3, pp. 704–707, September 2005.

[29] Q. Zhang, W. Zhu, and Y. Zhang. "End-to-End QoS for Video Delivery over Wireless Internet," *Proc. IEEE*, 2005.

[30] C. Luna, L. Kondi, and A. K. Katsaggelos, "Maximizing User Utility in Video Streaming Applications," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, no. 2, pp. 141–148, February 2003.

[31] G. Cheung and A. Zakhor. "Bit Allocation for Joint Source/Channel Coding of Scalable Video," *IEEE Trans. on Image Processing*, vol. 9, no. 3, pp. 340–357, March 2000.

[32] W. Tan and A. Zakhor. "Packet Classification Schemes for Streaming MPEG Video over Delay and Loss Differentiated Networks," Proc. Packet Video Workshop, 2001.

[33] Y. Shan and A. Zakhor. "Cross Layer Techniques for Adaptive Video Streaming over Wireless Networks," *IEEE ICME*, vol. 1, pp. 277–280, 2002.

[34] A. Majumdar, R. Puri, K. Ramchandran, and I. Kozintsev. "Robust Video Multicast under Rate and Channel Variability with Application to Wireless LANs," IEEE International Symposium on Circuits and Systems (ISCAS), Scottsdale, AZ, May 2002.

[35] F. Zhai, C.E. Luna, Y. Eisenberg, T.N. Pappas, R. Berry, and A.K. Katsaggelos. "Joint Source Coding and Packet Classification for Real-Time Video Streaming over Differentiated Services Networks," *IEEE Trans. Multimedia*, vol. 7, pp. 716–726, August 2005.

[36] A. K. Katsaggelos, Y. Eisenberg, F. Zhai, R. Berry, and T. N. Pappas. "Advances in Efficient Resource Allocation for Packet-Switched Video Transmission," *Proc. IEEE*, special issue on "Advances in Video Coding and Delivery," vol. 93, pp. 135–147, January 2005.

[37] Y. Eisenberg, C. Luna, T. N. Pappas, R. Berry, and A. K. Katsaggelos. "Joint Source Coding and Transmission Power Management for Energy Efficient Wireless Video Communications," *IEEE Tr. Circ. Sys. Video Techn.*, special issue on Wireless Video, vol. 12, pp. 411–424, June 2002.

[38] W. Wei and A. Zakhor. "Multipath Unicast and Multicast Video Communication over Wireless Ad Hoc Networks," *Proc. Int. Conf. Broadband Networks*, Broadnets, pp. 496–505, 2002.

[39] MPEG4IP: Open Source, Open Standards, Open Streaming [Online]. Available: http://mpeg4ip.net

[40] B. Awerbuch and T. Leighton. "Improved Approximation Algorithms for the Multicommodity Flow Problem and Local Competitive Routing in Dynamic Networks," *Proc. 26th ACM Symposium on Theory of Computing*, May 1994.

[41] S. Toumpis and A. J. Goldsmith. "Capacity Regions for Wireless Ad Hoc Networks," *IEEE Trans. Wireless Commun.*, vol. 2, no. 4, pp. 736–748, July 2003.

[42] A. Butala and L. Tong. "Cross-layer Design for Medium Access Control in CDMA Ad-Hoc Networks," *EURASIP Journal on Applied Signal Processing*, vol. 2005, no. 2, pp. 129–143, 2005.

[43] Y. Wu, P. A. Chou, Q. Zhang, K. Jain, W. Zhu, and S. Y. Kung. "Network Planning in Wireless Ad Hoc Networks: A Cross-Layer Approach," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 1, pp. 136–150, January 2005.

[44] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. "A High Throughput Path Metric for Multi-hop Wireless Routing," *Proc. ACM Conf. Mob. Computing and Networking*, MOBICOM, pp. 134–146, 2003.

[45] Q. Li and M. van der Schaar. "A Flexible Streaming Architecture for Efficient Scalable Coded Video Transmission over IP Networks," in *ISO/IEC JTC 1/SC 29/WG 11/M8944*, October 2002.

[46] J. Thie and D. Taubman. "Optimal Erasure Protection Assignment for Scalable Compressed Data with Small Packets and Short Channel Codewords," *EURASIP Journal on Applied Signal Processing: Special issue on Multimedia over IP and Wireless Networks*, no. 2, pp. 207–219, February 2004.

[47] P. A. Chou and Z. Miao. "Rate–Distortion Optimized Streaming of Packetized Media," *Microsoft Research Technical Report MSR-TR-2001-35*, February 2001.

[48] A. Grilo, M. Macedo, and M. Nunes. "A Scheduling Algorithm for QoS Support in IEEE 802.11E Networks," *IEEE Wireless Commun. Mag.*, vol. 10, no. 3, pp. 36–43, June 2003.

[49] P. Ansel, Q. Ni, and T. Turletti. "An Efficient Scheduling Scheme for IEEE 802.11E," *Proc. IEEE Workshop on Model. and Opt. in Mob., Ad-Hoc and Wireless Net.* (WiOpt 2004), Cambridge, UK, March 2004.

[50] S. Mangold, S. Choi, G. Hiertz, O. Klein, and B. Walker. "Analysis of IEEE 802.11E for QoS Support in Wireless LANs," *IEEE Wireless Commun. Mag.*, vol. 10, no. 6, pp. 40–50, December 2003.

[51] P. Garg, R. Doshi, R. Greene, M. Baker, M. Malek, and X. Cheng. "Using IEEE 802.11E MAC for QoS over Wireless," *Proc. IEEE Internat. Conf. on Perform. Comp. and Commun.*, IPCCC 2003, vol. 1, pp. 537–542, April 2003.

[52] B. V. Patel and C. C. Bisdikian. "End-Station Performance over Leaky Bucket Traffic Shaping," *IEEE Network Mag.*, vol. 10, no. 5, pp. 40–47, September 1996.

[53] JPEG2000: Image Compression Fundamentals, Standards and Practice, D. Taubman and M. Marcellin, Kluwer Academic, 2002.

[54] G. Bianchi. "Performance Analysis of the IEEE 802.11 Distributed Coordination Function," *IEEE Journal on Selected Area in Comm.*, vol. 18, No. 3, March 2000.

[55] A. Fattahi, F. Fu, and M. van der Schaar, "Mechanism-Based Resource Allocation for Multimedia Transmission over Spectrum Agile Wireless Networks," *IEEE JSAC*, Special issue on Adaptive, Spectrum Agile and Cognitive Wireless Networks, 2007.

[56] F. Fu and M. van der Schaar. "Proactive Cross Layer Design for Optimized Resource Exchanges Using Mechanism Design," *IEEE Trans. On Multimedia*, accepted for publication.

[57] H. Park and M. van der Schaar. "Utility-Based Fairness for Multi-user Wireless Multimedia Resource Allocation Using Bargaining," submitted to *IEEE Trans. on Signal Processing*.

[58] F. Kelly. "Charging and Rate Control for Elastic Traffic," *Eur. Trans. Telecommun. "Focus on Elastic Services over ATM Networks*," vol. 8, no. 1, pp. 33–37, 1997.

[59] F. Kelly, A. Maulloo, and D. Tan. "Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability," *Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, March 1998.

[60] H. P. Shiang and M. van der Schaar. "Multi-User Video Streaming over Multi-Hop Wireless Networks: A Distributed, Cross-Layer Approach Based on Priority Queuing," *IEEE Journal on Selected Area in Comm.*, May 2007.

[61] S. Shakkottai, T. S. Rappaport, and P. C. Karlsson. "Cross-layer Design for Wireless Networks," *IEEE Communications Magazine*, October 2003.

[62] A. Maharshi, L. Tong, and A. Swami. "Cross-layer Designs of Multichannel Reservation MAC under Rayleigh Fading," *IEEE Trans. Signal Processing*, vol. 51, no. 8, pp. 2054–2067, August 2003.

[63] I. Aad and C. Castelluccia. "Differentiation Mechanisms for IEEE 802.11," *Proceedings IEEE INFOCOM*, April 2001.

[64] C. Papadopoulos and G. Parulkar. "Retransmission Based Error Control for Continuous Media Applications," *Proceedings of the Sixth International Workshop on Network and Operating System Support for Digital Audio and Video*, 1996.

[65] R. Bhaskaran, P. Bhagwat, and S. Seshan. "Arguments for Cross-Layer Optimizations in Bluetooth Scatternets," *Proc. of Symposium on Applications and the Internet*, 2001.

[66] R. Kapoor, M. Cesana, and M. Gerla. "Link Layer Support for Streaming MPEG Video over Wireless Links," Conference on Computer Communications and Networks ICCCN'03, Dallas, TX, October 20–22, 2003.

[67] G. Pau, D. Maniezzo, S. Das, Y. Lim, J. Pyon, H. Yu, and M. Gerla. "A Cross-Layer Framework for Wireless LAN QoS Support," IEEE International Conference on Information Technology Research and Education, ITRE 2003, Newark, NJ, August 10–13, 2003.

[68] A. P. Butala and L. Tong. "Dynamic Channel Allocation and Optimal Detection for MAC in CDMA Ad Hoc Networks," in *Proc. 36th Asilomar Conf. Signals, Syst., Comput.*, Pacific Grove, CA, November 2002.

[69] R. Pan, C. Nair, B. Yang, and B. Prabhakar. "Packet Dropping Schemes, Some Examples and Analysis," *Proceedings of the 39th Annual Allerton Conference on Communication, Control and Computing*, pp. 563–572, October 2001.

[70] T. Yoo, R. Lavery, A. Goldsmith, and D. Goodman. "Throughput Optimization Using Adaptive Techniques," submitted for publication.

[71] G. Bianchi and A. Campbell. "A Programmable MAC Framework for Utility-based Adaptive Quality of Service Support," *IEEE Journal of Selected Areas in Communications*, Special Issue on Intelligent Techniques in High Speed Networks, vol. 18, no. 2, pp. 244–256, February 2000.

[72] S. Shankar and M. van der Schaar. "Multimedia Transmission over WLANs Using Cross Layer Design—Challenges, Principles and Standards," half-day tutorial at *IEEE Globecom 2003*.

This page intentionally left blank

13

Quality of Service Support in Multimedia Wireless Environments

Klara Nahrstedt, Wanghong Yuan, Samarth Shah, Yuan Xue, and Kai Chen

13.1 INTRODUCTION

Over recent years, there has been a strong proliferation in the use of multimedia wireless technology all over the world, creating new research and business opportunities for producers and consumers of these technologies. There are several reasons for this fast proliferation. First, wireless networking technologies such as cellular networks, wireless local area networking (WLAN), and Bluetooth are becoming an integral part of our communication environment. Second, new wireless devices such as cellular phones, PDAs, and laptops are emerging to assist people in their lives. Third, multimedia applications became popular, first in the Internet environment and now in wireless environments, due to (a) standardization of digital multimedia formats such as MPEG-2, MPEG-4, H.263, and others and (b) understanding of multimedia applications and user behaviors under different networking conditions. This allows service providers to build large-scale multimedia services such as video conferencing and video-on-demand and to offer them to the population at large. Fourth, new hardware opportunities are appearing such as multifrequency energy-efficient processors, allowing for more efficient use of energy in mobile devices.

However, these new opportunities bring with them also various challenges. We will concentrate on addressing two major challenges. First, mobile devices running distributed multimedia applications and communicating over wireless networks must deal with scarce and variable resources such as battery power, processor speed, memory, and wireless bandwidth. Hence the resource management

problem for support of Quality of Service (QoS) must be solved. Second, multimedia applications running over wireless networks must achieve some level of performance QoS guarantees. Hence modeling of application QoS, QoS management and its connectivity to underlying resource management must be addressed.

In this chapter, we aim to answer these application QoS and resource management challenges and to describe some of the solutions that may contribute to solving these challenges. Since these two challenges are still very broad, we narrow their scope to address the following problems:

- The topic of multimedia applications and QoS is very broad and there is an extensive pool of solutions in the literature. We concentrate on modeling of conversational applications with strict delay requirements such as Voice over IP and retrieval applications with sensitive throughput requirements such as multimedia on demand using mobile multimedia devices. We consider three QoS metrics for multimedia distributed services: throughput, end-to-end delay, and application lifetime.
- The topic of resource management in wireless networks is also very broad and there are multiple techniques that optimize different resource usage. Furthermore, resource management is required for all types of wireless networks such as cellular networks, wireless LANs, mobile ad hoc networks, and sensor networks. In this chapter we concentrate on resource management schemes meant only for networks based on, or compatible with, the widely used network standard IEEE 802.11. Also, we consider four major resources to deliver application QoS: wireless network bandwidth, CPU bandwidth, memory, and energy. We provide algorithms, services, and protocols at the operating system and middleware layer with cross-layered access to selected information in lower level network solutions. We consider resource management in single mobile devices, in mobile devices connected via single hop ad hoc networks, and in mobile devices connected via access-point-based networks.

To design solutions that address the end-to-end QoS issues and corresponding resource management in 802.11 wireless single hop environments, we take the top-down approach in this chapter. First, we decide on multimedia applications and their models that will run in these environments. Section 13.2 discusses the modeling of these applications and their QoS requirements, especially the application task, connection and QoS (quality) models, and the cross-layer application-OS-network models that drive correct resource allocations in mobile nodes. Second, once it is clear what applications are primarily running in the 802.11 wireless environments, resource management techniques need to be chosen that execute according to QoS requirements. These resource management techniques must span within individual mobile nodes via their operating systems and across mobile nodes via the distributed network management. Therefore, the rest of the

chapter addresses (a) operating-system-internal resource management techniques that help delivering application QoS requirements inside an individual mobile node and (b) network-specific resource management techniques to deliver QoS in end-to-end fashion from the sender(s) to the receiver(s).

The operating-system-internal techniques can be found in Section 13.3 and the network-specific techniques can be found in Section 13.4 as follows. Section 13.3 concentrates on the energy-efficient operating system (EOS) at mobile end points. The Linux-based EOS includes an integrated and cross-layer-optimized CPU/energy resource management to guarantee node delay and application lifetime QoS requirements. This end-point resource management must be addressed to achieve true end-to-end quality guarantees for any multimedia application [42]. Section 13.4 addresses the cross-layer-optimized network resource management to guarantee end-to-end delay and bandwidth QoS requirements in single hop wireless networks. The reason for concentrating on single hop wireless networks is that in commercial applications such as music on demand or phone conversations we believe there will be only a few hops before the multimedia stream reaches the wired infrastructure through which the information will be transported. Hence, what we need to ensure in wireless networks for these types of applications is that the multimedia data be transmitted over the first/last wireless mile in a quality-aware manner.

We have built multiple cross-layer QoS-aware systems that utilize techniques in Sections 13.3 and 13.4. The design principles and overall lessons learned from their design and development are summarized in Section 13.5. The chapter concludes with possible future directions with respect to wireless multimedia and the corresponding QoS support in Section 13.6.

13.2 APPLICATION MODELING

Wireless multimedia applications on various mobile devices are becoming an integral part of our life. Examples are music on demand using the Apple iPOD devices, short video clips on demand using cell phones, DVD players on laptops, and voice over IP using laptops and PDAs. We will first specify the common model of these applications so that we can then address them more easily in the resource management and design solutions that will serve these applications. We consider computational and communication requirements of multimedia applications to have comprehensive and expressive models for OS and network resource management and their support for QoS guarantees.

During their lifetimes, distributed multimedia applications use computational and communication resources on their mobile nodes. Hence when modeling multimedia applications, we need to consider requirements that these applications have on both resources and to include them into the overall application model.

Furthermore, we need to consider the overall quality goal of the end-to-end application. Therefore, the application model will consist of two parts: (a) application task, connection, and quality models and (b) cross-layer application model.

13.2.1 Application Task, Connection, and Quality Models

We consider multimedia distributed applications (video, voice, or music) as periodic tasks, running distributed application functions over single or multiple network connections between sender(s) and receiver(s). Each task consumes CPU time, energy, and network bandwidth resources and provides an output quality. Multimedia applications are adaptive tasks, which means that from the computational point of view they are *soft real-time tasks* that can operate at multiple application QoS (quality) levels. For example, a QoS level may correspond to a video frame rate in a video task. Each task i supports a discrete set of QoS levels, $q_{i1}, \ldots, q_{im}$ [7,43]. Each task can provide different quality levels, trading off quality with resource consumption or trading off consumption between different resources [37]. We aggregate all *best effort (nonmultimedia) applications* into one logical adaptive task. This logical task delivers either average (in lightly loaded environment) or no (in a heavily loaded environment) quality guarantees to individual best effort tasks.

Each connection connects multiple tasks to form a transmission medium between sender(s) and receiver(s) to exchange multimedia data and control information among mobile nodes. Also, each connection consumes through its distributed tasks CPU time, energy, and bandwidth resources, and based on the shared resource availability, especially the wireless channel, it provides an output quality. It is important to stress that from the 802.11 wireless networking point of view, multimedia distributed applications must be adaptive. This means that the end-to-end connections can yield only soft end-to-end guarantees and, in many cases, only statistical or best effort guarantees.

13.2.2 Cross-Layer Application-OS-Network Model

Each wireless multimedia application must have a strong relation to the underlying computing and communication layers that allocate resources to provide QoS guarantees q and utility $u(q)$. We will consider two layers where multimedia applications will interface to: (1) process management (representing the operating system and its access to processor hardware) with its soft real-time task scheduling and (2) middleware layer (representing entrance to the network protocol stack) with its connections/packets scheduling and bandwidth management.

Each application QoS level q has a utility $u(q)$, which measures the perceptual quality at a QoS level from the user's point of view and consumes $C(q)$ cycles and $B(q)$ network bytes per period $P(q)$. Furthermore, we assume that for

each QoS level, the task has probability distribution of its cycle demand; that is, $F(x) = \mathcal{P}r(X \leq x)$ is the probability that the task demands no more than x cycles for each job. This distribution can be obtained with our previously developed kernel-based profiler [49,50]. Specifically, the operating system uses a profiling window to keep track of the number of CPU cycles each task has consumed for its recent jobs. The operating system then builds a histogram based on the result in the profiling window. The histogram estimates the probability distribution of the cycle demand of the task for each job. With respect to network connections, we assume two network models: the integrated service ("IntServ") model and the differentiated service ("DiffServ") model that determine the network bandwidth allocation in 802.11 wireless networks and will be discussed in detail in Section 13.4.1.

In the hardware layer, a multimedia application uses two adaptive resources: CPU and wireless network interface card (WNIC). The CPU can operate at multiple speeds (frequencies/voltages), $\{f_1, \ldots, f_{\max}\}$, trading off performance for energy. The power consumption of the CPU is $p(f)$ at speed f. The lower the speed is, the lower the power is. We assume that the overhead for adapting CPU speed is negligible.

The WNIC supports three operation modes: *active*, *idle*, and *sleep*, where the sleep mode has much less power. Power consumptions at the aforementioned states are p_{act}, p_{idl}, and p_{slp}, respectively. The overhead for switching the WNIC into sleep and from sleep is t_{slp}, which is not negligible (e.g., around 40 ms for the Lucent WaveLan card).

13.3 QoS SUPPORT IN MOBILE OPERATING SYSTEMS

In mobile wireless environments, QoS-aware operating system support for battery-powered mobile nodes is crucial in order to run multimedia applications. Such multimedia-enabled mobile systems need to save energy while supporting multimedia QoS requirements. There is a conflict in the design goals for QoS provisioning and energy saving. For QoS provisioning, system resources often need to provide high performance, typically resulting in high energy consumption. For energy saving, system resources should consume low energy. As a result, the operating system of mobile devices needs to manage resources in a QoS- and energy-aware manner and provides the flexibility to trade off QoS and energy based on the user's preferences.

Recently, a number of soft real-time operating systems has been proposed to support QoS for multimedia applications. These operating systems typically integrate predictable CPU allocation (such as proportional sharing [8,31] and reservation [18,36]) and real-time scheduling algorithms, such as earliest deadline first (EDF) and rate monotonic [21]. Energy management is also an im-

portant part of the operating system. For example, ECOSystem [52] and Nemesis [29] manage energy as a first-class OS resource. Vertigo [16] saves energy by monitoring application CPU usage and adapting the CPU speed correspondingly. More recently, there is some work on QoS and energy-aware cross-layer adaptation [27,33,34,50,51]. Pereira *et al.* [33] proposed a power-aware application programming interface that exchanges the information on energy and performance among the hardware, OS, and applications. Mohapatra *et al.* [27] proposed an approach that uses a middleware to coordinate the adaptation of hardware and applications at coarse time granularity (e.g., at the time of admission control). EQoS [34] is an energy-aware QoS adaptation framework, which formulates energy-aware QoS adaptation as a constrained optimization problem. GRACE [49–51] coordinates the adaptation of the CPU speed in the hardware layer, CPU scheduling in the OS layer, and multimedia quality in the application layer in response to system changes at both fine and coarse time granularity.

We next introduce the design of our operating system, which is a part of the GRACE project [50,51].

13.3.1 Design and Algorithm

The goal of the operating system is to maximize multimedia quality q of all concurrent tasks in the mobile device under the constraints of CPU, network bandwidth, and battery energy. Figure 13.1 shows the architecture of the operating system, which includes four major components: a coordinator, a soft real-time CPU scheduler, a CPU adapter, and a WNIC adapter. The coordinator coordinates tasks and the CPU and WNIC resources to determine the quality level and CPU allocation for each task and the average power consumption for the CPU and WNIC. The CPU scheduler enforces the coordinated allocation to support the coordinated QoS levels of individual tasks. Finally, the CPU and WNIC adapters dynamically adapt the CPU and network card to minimize their power consumption. We next describe each component in turn.

13.3.1.1 Coordination

The goal of the coordination is to maximize the aggregate utility of all concurrent tasks in the device subject to the constraints of CPU, network, and energy in the device. More formally, let's assume that (1) there are n tasks concurrently running in the device. Each task has multiple QoS levels, $\{q_{i1}, \ldots, q_{im}\}$. Each QoS level has a utility $u(q_{ij})$, consumes $C(q_{ij})$ cycles and $B(q_{ij})$ network bytes per period $P(q_{ij})$; (2) the remaining battery energy in the device is E; (3) the estimated operating time of the device is T; and (4) the available network bandwidth is BW. The coordination needs to determine a QoS level for each task, the CPU speed f and power $p(f)$, and the network power p_{net}. Intuitively, when tasks operate at a

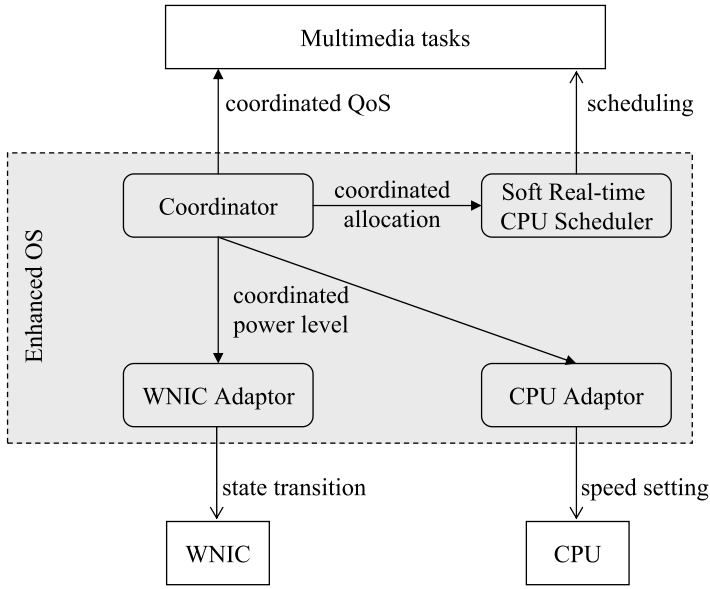


FIGURE 13.1: The architecture of the OS.

higher QoS level, they demand more CPU and network resources; consequently, the CPU and WNIC perform at higher performance and hence consume more energy.

The coordination problem can be formulated as follows:

$$\text{maximize} \quad \sum_{i=1}^{n} u(q_{ij}) \qquad \text{(total utility)} \tag{13.1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \frac{C(q_{ij})/f}{P(q_{ij})} \leq 1 \qquad \text{(CPU constraint)}, \tag{13.2}$$

$$\sum_{i=1}^{n} \frac{B(q_{ij})}{P(q_{ij})} \leq BW \qquad \text{(network constraint)}, \tag{13.3}$$

$$\big(p(f) + p_{\text{net}}\big) * T \leq E \qquad \text{(energy constraint)}, \tag{13.4}$$

$$q_{ij} \in \{q_{i1}, \ldots, q_{im_i}\} \qquad i = 1, \ldots, n \text{ (QoS levels)}, \tag{13.5}$$

$$f \in \{f_1, \ldots, f_{\max}\} \qquad \text{(CPU speeds)}. \tag{13.6}$$

The CPU power $p(f)$ is directly determined by the speed f. We determine the network power as follows: If the transmission speed for the WNIC is S, the WNIC needs to be in the active state for B/S and in idle state for $1 - B/S$ for every second, where B is the aggregate bandwidth requirement of all tasks, that is, $\sum_{i=1}^{n}(B(q_{ij})/P(q_{ij}))$. Then the network power is

$$p_{\text{net}} = p_{\text{act}} * \frac{B}{S} + p_{\text{slp}} * \left(1 - \frac{B}{S}\right). \tag{13.7}$$

Note that in (13.7), we switch the WNIC into sleep when it is idle.

The aforementioned constraint optimization happens at coarse time granularity, for example, when a task joins or leaves the system. The coordination problem is NP hard, since we can prove that the NP-hard Knapsack problem is an instance of the aforementioned constraint optimization problem. We therefore use the dynamically programming algorithm [28] that provides a heuristic solution. As a result of this solution, we determine the QoS level and CPU allocation for each task as well as the average CPU power and network power.

13.3.1.2 Soft Real-Time CPU Scheduling

Soft real-time scheduling is a common mechanism to support timing requirements of multimedia applications [8,11,30]. Here, we focus on the CPU scheduling. Previous soft real-time scheduling algorithms, however, often assume that the CPU runs at a constant speed. This assumption does not hold for our target mobile devices with a variable-speed CPU. As a result, we cannot directly use existing scheduling algorithms in our system. We therefore extend traditional real-time scheduling algorithms by adding another dimension—*speed*. That is, the scheduler also sets the CPU speed when executing a task and hence enforces the CPU allocation on a variable-speed CPU [48].

The operating system uses an energy-aware EDF scheduling algorithm, which enforces the globally coordinated CPU allocation on a variable-speed CPU [48]. Specifically, in this scheduling algorithm, each task has a deadline and a cycle budget:

- The deadline of the task equals the end of its current period. That is, when a task begins a new period, its deadline is postponed by the period.
- The budget of a task is recharged periodically. In particular, when a task begins a new period, its budget is recharged to the coordinated number of cycles.

The scheduler schedules all tasks based on their deadline and budget. In particular, the scheduler always dispatches the task that has the earliest deadline and a positive budget. As the task is executed, its budget is decreased by the number

of cycles it consumes. When the budget of a task is decreased to 0, the task is preempted to run in best-effort mode until its budget is replenished again at the next period.

This preemption provides temporal and hence performance isolation among tasks; that is, a task's performance is not affected by the behavior of other tasks [11,18,30].

13.3.1.3 CPU Energy Saving

As the coordination problem just given shows, the coordinated CPU power consumption is $p(f)$, where $f = \sum_{i=1}^{n}(C(q_{ij})/P(q_{ij}))$. In other words, we expect the CPU to execute at a uniform speed for all concurrent tasks. If each task uses exactly $C(q_{ij})$ cycles per period $P(q_{ij})$, this uniform speed technique would consume minimum energy due to the convex nature of the CPU speed–power function [17]. However, the instantaneous cycle demand of multimedia tasks often varies greatly. In particular, a task may, and often does, complete a job before using up its allocated cycles. Such early completion often results in CPU idle time, thereby wasting energy. To avoid this energy waste, we dynamically adapt the CPU speed during each task's execution.

However, we cannot lower the speed too much, as the task may miss its deadline or cause other tasks to miss their deadlines. To do this, we allocate the task a time as follows: If there are n concurrent tasks and each task is allocated C_i cycles per period P_i, then the scheduler allocates the ith task CPU time $T_i = C_i/\sum_{i=1}^{n}(C_i/P_i)$ every period P_i. The reason for time allocation (in addition to cycle allocation) is to guarantee that each task executes for up to its allocated cycles within its allocated time, regardless of speed changes.

Without loss of generality, we focus on the speed adaptation for an individual task, which is allocated C cycles and T time per period and has a probability distribution of its cycle demand $F(x) = \mathcal{P}r(X \le x), 1 \le X \le C$. Our goal is to minimize the expected energy consumption of each job of the task. To do this, we find a speed for each of the allocated cycles of this task, such that the total energy consumption of these allocated cycles is minimized while their total execution time is no more than the allocated time. More formally, if a cycle x executes at speed f_x, its execution time is $1/f_x$ and its expected energy consumption is $(1 - F(x)) \times 1/f_x \times p(f_x)$ [9]. We can then formulate the speed adaptation schedule problem as follows:

$$\min: \underbrace{\sum_{x=1}^{C}\big(1 - F(x)\big)\frac{1}{f_x}p(f_x)}_{\text{busy energy}} + \underbrace{\left(T - \sum_{x=1}^{C}\big(1 - F(x)\big)\frac{1}{f_x}\right)p_{\text{idle}}}_{\text{idle energy}} \qquad (13.8)$$

subject to:

$$\sum_{x=1}^{C} \frac{1}{f_x} \leq T, \tag{13.9}$$

$$f_x \in \{f_1, \ldots, f_{\max}\}, \tag{13.10}$$

where p_{idle} is the CPU idle power at the lowest speed. Note that the energy consists of two parts: The first part is the energy consumed when executing all allocated cycles. The second part is the energy consumed during the residual time (i.e., the time budget minus the expected execution time of all allocated cycles). During this residual time, the CPU is often idle since the process needs to wait until the next job is available. During this idle time, we set the CPU to the lowest speed during the idle slack.

We refer to the aforementioned optimization as a statistical Dynamic Voltage Scaling (DVS) approach. This optimization happens at fine time granularity, for example, within a multimedia frame execution. The optimization problem is NP hard. To provide an approximate solution, we develop a dynamic programming algorithm, based on the algorithm proposed by Pisinger [35]. Specifically, we first divide the allocated cycles into groups and find a speed for each group, rather than for each cycle. We then consider the combinations of all speed options for all cycle groups and sort them in the nondecreasing order of a *slope* that is defined as the ration of the increased energy to the decreased time by increasing a group's speed to the next higher speed. We initially set all cycle groups to the lowest speed and then visit the sorted slope list. For the currently visited slope, we try to increase the speed of its associated cycle group to the next higher speed. We finish the visit when the total execution time of all cycle groups is no more than its allocated time.

Each task has its own speed schedule and its speed schedule applies to all its jobs. In other words, the OS changes the CPU speed in three cases (Figure 13.2):

- **Context switch**. After a context switch, the OS sets the CPU speed based on the speed schedule of the switched-in task. This provides isolation of speed scaling among different tasks.
- **New job**. When the current task releases a new job, its execution speed is reset to the speed of its first cycle group.
- **Job progress**. The OS also monitors the progress of each job execution and changes the CPU speed when the job reaches its next cycle group.

13.3.1.4 Network Energy Saving

Dynamic power management (DPM) is a common technique used to save network energy by switching the WNIC into sleep when it is idle. DPM, however,

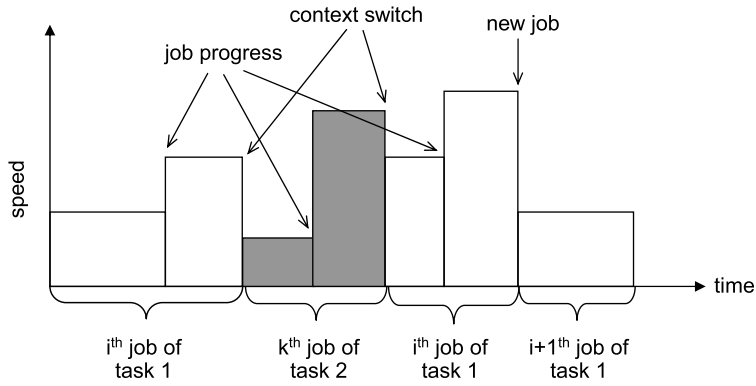


FIGURE 13.2: The OS changes the CPU speed during job execution and at context switch.

cannot be directly applied in our target multimedia systems for the following reason. Multimedia applications are periodic and need to transmit or receive data in each period. Consequently, the idle interval of the WNIC is often shorter than the period. Since the period is often shorter than the DPM overhead, the WNIC cannot enter the lower-power sleep mode. To save network energy, we use a *buffering* approach. In this approach, each task still performs computation every period in a timely fashion, but delays the transmission by buffering frames and sending them in bursts at longer intervals (Figure 13.3).

Specifically, let's assume that the buffer size is k frames and each frame needs to transmit for t_{act} time. In each period P, the task processes a frame and stores it in the buffer. When the buffer has k frames (i.e., every k periods), the OS sends all buffered frames in batch. The buffering approach combines short WNIC idle intervals with length $(P - t_{\text{act}})$ into longer ones with length $k(P - t_{\text{act}})$. Such aggregate idle intervals are larger than the DPM overhead so the WNIC can enter sleep. The buffering approach saves more energy. That is, the network power in the k period is

$$\begin{aligned} p_{\text{net}} &= \frac{p_{\text{act}} \times k t_{\text{act}} + p_{\text{slp}} \times k(P - t_{\text{act}})}{kP} \\ &= p_{\text{act}} \times \frac{t_{\text{act}}}{P} + p_{\text{slp}} \times \left(1 - \frac{t_{\text{act}}}{P}\right), \end{aligned} \tag{13.11}$$

which is equivalent to (13.7). That is, we enable the WNIC to consume the coordinated power.

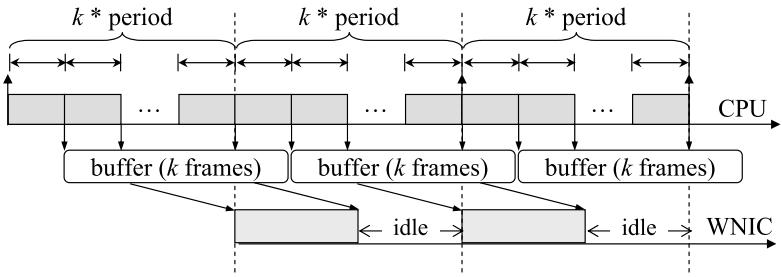


FIGURE 13.3: The buffering approach.

13.3.2 Experimental Results

We have implemented a prototype of the OS. The hardware platform for our implementation is the HP Pavilion N5470 laptop with a single AMD Athlon 4 processor, which supports six different frequencies, 300, 500, 600, 700, 800, and 1000 MHz. The laptop has a Cisco Aironet 350 wireless card. The coordinator, scheduler, and CPU adapter are implemented as a set of patches and modules that hook into the Linux kernel 2.6.5. The WNIC adapter is implemented as a user-level process that switches the WNIC into the power-saving mode (PSM) when it is idle and into the continuous access mode (CAM) when it is active.

We next evaluate the OS prototype. Since the coordination requires the utility function for each task, which is application specific, we focus on our evaluation on energy saving. To measure energy, we remove the battery from the laptop and let it use the power from the AC adapter. The power consumption is the product of the input voltage and input current from the AC adapter. We use the Agilent 54621A oscilloscope to record the measurement. The sampling rate of the oscilloscope is 5 kHz, that is, making a sample every 200 μs. Figure 13.4 shows the setup for power measurement.

First, we analyze the impact of the CPU adaptation and WNIC adaptation together. To do this, we use an H263 encoder that encodes local raw images into frames in real time and sends the encoded frames to a receiver through a wireless network. The input pictures are `paris.cif`. The H263 encoder can process two or three frames per second in real time. We measure average energy consumption characteristics for three different system scenarios:

- *no adapt*: the CPU always runs at the highest speed and the WNIC always runs at the CAM mode.
- *CPU only*: the CPU runs at a uniform speed that meets the total average demand of applications.
- *CPU+NW*: The CPU and WNIC both adapt.

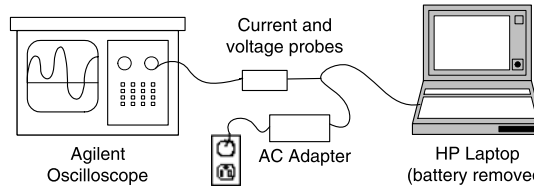


FIGURE 13.4: Setup for power measurement.

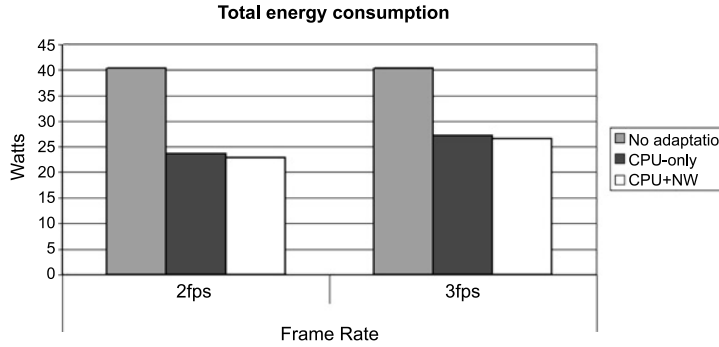


FIGURE 13.5: Benefits of CPU adaptation and WNIC adaptation.

This gives us an idea of the energy saving resulting from CPU speed adaptation and wireless card mode changing.

Figure 13.5 shows the energy consumption of the laptop. We note that the CPU speed adaptation reduces base energy consumption of the laptop by about 34% at 3 fps, and the network card mode-changing reduces the energy consumption of the network interface by about 42% at the same frame rate. However we find that the total energy saving in the CPU+NW case over the CPU-only case is only 2–3% with the HP Pavilion laptop. The reason is that the WNIC consumes much less energy than the CPU in the HP laptop.

Second, we evaluate the benefits of our proposed CPU energy-saving technique. To do this, we disable the wireless connection and let the H263 encode three frames per second in real time and store the encoded frames in a local file. We run the stand-alone H263 encoder under the following DVS techniques:

- *No DVS*. This is the baseline technique that always runs the CPU at the highest speed.
- *Uniform DVS*. It sets the CPU speed based on the average CPU demand of the H263 encoder.

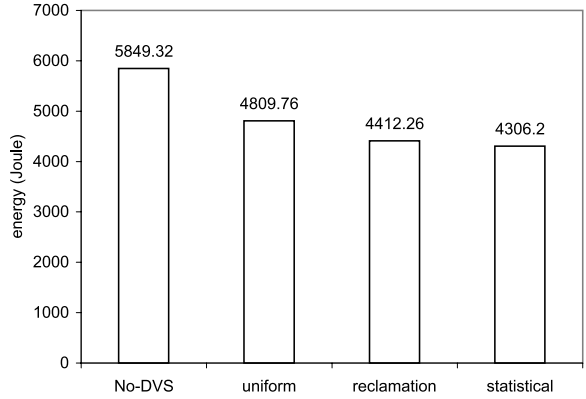


FIGURE 13.6: Benefits of statistical DVS compared to other DVS techniques.

- *Reclamation DVS*. It first sets a uniform speed and sets to the lowest speed when the H263 encoder completes a job early.
- *Statistical DVS*. It adjusts the execution speed for each task job based on its demand distribution.

Figure 13.6 shows the energy results. Compared to the baseline algorithm without DVS, all DVS techniques save energy significantly. In particular, the statistical DVS reduces energy by 26.4% compared to the no-DVS approach. The reason is that the CPU does not need to always run at the highest speed. This clearly shows the benefits of energy saving by dynamically adapting the CPU speed. Compared to other DVS techniques, the statistical DVS further reduces the total energy by 2 to 10%. This clearly shows the benefits of adapting the CPU speed based on demand distribution of tasks.

13.4 QoS SUPPORT IN MOBILE WIRELESS NETWORKS

To achieve end-to-end QoS guarantees in multimedia wireless networks, a strong QoS-aware cross-layer networking system support for wireless multimedia applications must be present. We will present a complete cross-layer networking system support for a *single-hop ad hoc network* based on the IEEE 802.11 MAC layer. In this network, all the nodes are within one-hop transmission range of each other. They are able to talk to each other in a peer-to-peer fashion. They all share the same wireless medium and hence need to cooperate with each other in satisfying their QoS needs.

The nodes in our network use the IEEE 802.11 MAC protocol's Distributed Coordination Function (DCF) mode for communication. The IEEE 802.11 standard specifies two operating modes: Point Coordination Function (PCF) and DCF. The former requires a single coordinator to arbitrate access to the shared wireless channel. The latter mode allows peers to arbitrate channel access without any centralized coordinator using a CSMA/CA protocol. Wireless nodes using the DCF mode carrier sense the medium. If the channel is busy, transmissions are deferred. When the channel is clear, nodes back off for collision avoidance. A node that captures the channel for transmission uses a RTS-CTS-DATA-ACK cycle to transmit a MAC frame. The RTS/CTS handshake is used mainly to deal with the hidden terminal effect.

Multimedia applications running over this 802.11 DCF network need to pay attention to the following conditions: (1) interference of wireless communications between different flows within the network and (2) dynamics of the network environment where resource usage patterns and wireless signals may vary with time. As a result, multimedia applications need to adapt to these conditions with proper system support.

To date, most of the existing work has been proposed within the context of an individual layer, such as the routing and MAC layer. Much less progress has been made in addressing the overall system support for running multimedia applications over wireless networks. One solution for overall system support for wireless multimedia applications is to adopt a *cross-layer* system architecture among MAC, transport, middleware and application layers. All these layers communicate and coordinate with each other to support QoS for multimedia applications.

13.4.1 QoS Models

Before discussing details of our cross-layer networking system architecture, we first revisit the QoS models proposed in the Internet and review their applicability in the wireless environment. There are two different QoS models: (1) the integrated service ("IntServ") model and (2) the differentiated service ("DiffServ") model.

The IntServ QoS model defines two types of services: *guaranteed* and *best effort*. In guaranteed service, each *flow* can request a certain level of QoS from the network, such as minimum bandwidth and maximum delay. Over the Internet, IntServ is usually implemented by per-flow resource reservation in the routers. To apply the IntServ model to a wireless network, admission control must be designed to work with imprecise and time-varying resources information. Furthermore, the reserved resources of a flow may have to change in response to wireless resource fluctuations. As a result, in wireless networks, multimedia applications often specify their QoS requirements over a *range*, for example, minimum and

maximum bandwidths, and the granted resource can be a QoS level within that range.

In the DiffServ model, flows are aggregated into multiple traffic *classes*. A router needs to provide certain per-hop forwarding behavior for each class of packets. In particular, we are interested in the *relative DiffServ* model [12], which assures the relative quality ordering between different classes. *No* guarantee is provided for any of those classes. This QoS model is appealing, especially in wireless networks because it does not need to provide any bandwidth guarantees for any class of packets. Instead, it relies on the end-host's adaptation behavior to dynamically select an appropriate service class for each of its applications.

These two QoS models address different needs of multimedia applications. IntServ is more stringent in resource provisioning. An application has a better level of QoS guarantee but at the same time there is a higher probability that the QoS request may be rejected in admission or terminated due to resource fluctuations. The relative DiffServ model has less guarantee for each application, but each application is always allowed to send out packets, although with different levels of QoS.

In the following subsections, we discuss in detail our design of two cross-layer architectures that realize the QoS models mentioned earlier. There have been several other cross-layer architectures for dynamic bandwidth management and adaptation, such as INSIGNIA [20], SWAN [2], TIMELY [6], dRSVP [26], and, most recently, MPARC [47] and PBRA [46]. These cross-layer architectures assume different QoS models and network topologies, but the underlying mechanisms (subtasks) implemented in order to manage bandwidth are, with some exceptions, similar: available bandwidth monitoring at the MAC, soft state reservation, application adaptation to network variations, and fair bandwidth allocation. QoS research for wireless networks has also addressed fair scheduling at the MAC layer [4,15,19,22–24,44] and new transport mechanisms [3,25,40,46] to improve application performance.

13.4.2 IntServ: Bandwidth Management

13.4.2.1 Bandwidth Management Architecture

Bandwidth Management architecture [38] arbitrates the bandwidth requests of all the flows in a single-hop wireless network. In this architecture, every host in the network monitors its MAC layer transmissions to observe wireless channel fading and interference effects. These observations are fed into a *central* network arbiter, which takes the bandwidth requirements and channel effects pertaining to each multimedia stream in the network into account to decide how much *channel time* each stream gets to access the network to ensure that its requirements are met.

An overview of the architecture is shown in Figure 13.7. It consists of a middleware agent for each host, which obtains channel quality updates from the MAC

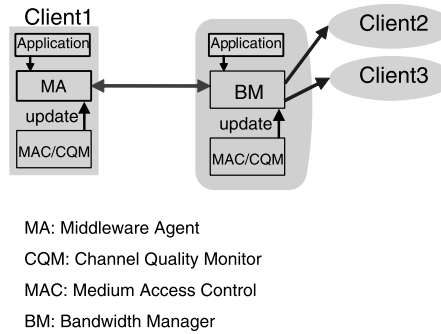


FIGURE 13.7: Bandwidth management architecture: overview.

layer monitor and application throughput requirements from each application, including the media application. It translates the throughput requirements into *channel time* requirements, using channel quality. The channel time requirement represents the fraction of unit time that the media stream must have access to the wireless channel of the observed quality in order to satisfy its throughput requirement. The channel time required thus depends on the throughput required as well as the channel quality observed. The middleware agent feeds these channel time fractions required by a particular stream to a central network arbiter, called the *Bandwidth Manager* (BM), which resides on one of the hosts in the network. The BM allocates the unit channel time resource among the various media streams in the network. It can be configured with any logical policy to distribute the resource among the streams, taking into account their requirements, for example, by using a fair, utility-based or price-based policy.

The BM returns to the middleware agent at each wireless host the *channel fraction* allocated to each application running on the host. When there is some change in application throughput requirement or channel quality observed, the BM must reallocate resources. This may involve *revoking* partially the resources previously allocated to an application and reallocating them based on the new network conditions.

Each host also has a *rate-control system* (i.e., traffic shaper) comprising leaky-bucket queues. It is configured to ensure that each application injects no more traffic than can occupy the channel for the fraction of time the application was allotted. The sequence of events within each wireless host is shown in Figure 13.8.

The centralized architecture shown in Figure 13.7 is flexible enough to work with any single-hop wireless topology. It can be used for single-hop peer-to-peer ad hoc networks and for access-point (AP)-based networks. Furthermore, it can be extended to a network consisting of multiple APs that cover a large area with overlapping frequency bands. In such networks, the BM must also keep track

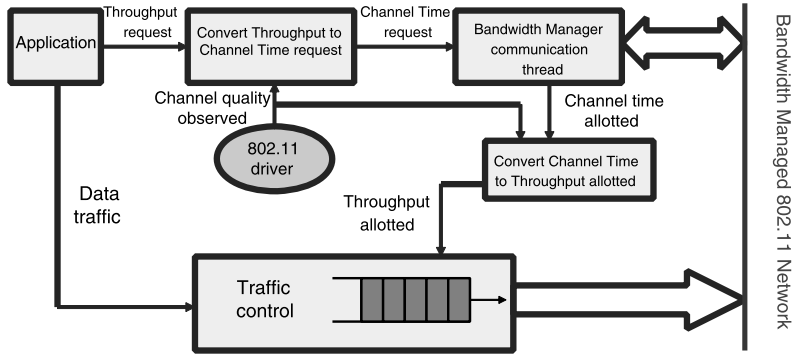


FIGURE 13.8: Bandwidth management architecture: host.

of spatial reuse of the channel resource, apart from tracking the channel time requirements [39]. The channel quality monitor that we describe later is powerful enough to help a host using one AP to detect the presence of interference from a host using a different one.

13.4.2.2 Channel Quality Monitoring

In the BM architecture, a key component is the monitoring of the channel quality. We monitor the channel quality at the MAC layer, that is, we observe how fading and interference phenomena affect MAC frame transmissions. We observe the delay in MAC frame transmission and observe the loss rate of MAC frames. We explain in this section how fading and interference phenomena are manifested in the MAC frame delay and loss rate. We do not change the IEEE 802.11 protocol in any way in constructing the channel quality monitor. We merely observe the MAC layer transmissions of data packets in drawing our inferences.

Interference on the network can be estimated by the amount of time a transmitting host senses the channel busy, and must hence back-off and wait before being able to transmit its RTS or DATA frame. Thus the delay $t_r - t_s$ in Figure 13.9 (top) reflects the interference levels in the network. Signal fading effects cause bit errors in individual frame transmissions, thus requiring the frame to be retransmitted. If ultimately the RTS-CTS-DATA-ACK cycle is successfully completed, then the interval $t_r - t_s$ also measures signal fading effects, since delays due to retransmissions are also accounted for in the interval. In case the RTS or DATA retransmission limit is exceeded, then the frame is dropped at the MAC layer, despite a time T_w wasted in trying to send it. Thus measuring this time wasted T_w in Figure 13.9 (bottom) and the frame loss rate is also crucial in estimating signal fading effects.

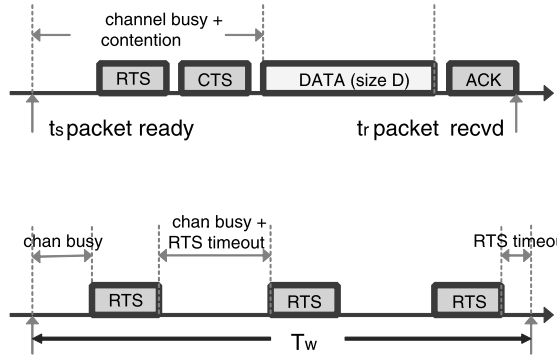


FIGURE 13.9: Successful and unsuccessful transmissions in 802.11.

Our channel quality monitoring scheme also accounts for hidden-terminal effects that might occur in networks spread out over a larger area. Hidden terminal effects cause the CTS frame to be suppressed. This is because the transmitter of the RTS does not know about the transmissions in the receiver's neighborhood. But these transmissions prevent the intended receiver from responding with a CTS. This may result in multiple RTS retransmissions, as would be the case if there were bit errors in the individual frames, and even result in the RTS retransmission limit being exceeded. Both of these scenarios are accounted for when we measure the delay in Figure 13.9 (top) and the time wasted and frame loss rate in Figure 13.9 (bottom).

The channel quality monitoring mechanism measures, over a time interval T, the number of frames successfully transmitted, the delay $t_r - t_s$ incurred in transmitting them, the number of frames lost, and the time wasted in attempting to transmit them T_w. These four measures comprise our channel quality metric. They give us an indication of how many higher layer packets can be transmitted in unit time and how many will be lost in the process. The channel fraction required for a media stream to obtain its required throughput depends on this information.

Note that we have described earlier only the principle behind our channel quality monitoring mechanism. We have omitted the details pertaining to how different higher layer packet sizes affect the monitor and also considerations pertaining to packet-header overhead consuming some channel fraction. Details on dealing with these issues can be found in [38].

13.4.2.3 Illustrative Example

We now demonstrate using the network simulator ns-2 the performance of our channel quality monitoring and rate-control schemes that together constitute our bandwidth management solution. We assume a network topology shown in

Figure 13.10 with two APs, node mobility, handoff, and hidden node effects. The transmission range of a wireless node is 250 m and the carrier-sense range is 550 m.

Note that there is no spatial reuse of the channel, that is, two transmissions are not simultaneously possible on the wireless channel. In order to create the hidden node effect, and illustrate how our scheme deals with it, we assume both APs use the same wireless frequency, although in practice adjacent 802.11 APs tend to use noninterfering frequencies in the 2.4-GHz band. The BM is located in the backbone distribution system and is not shown in Figure 13.11.

Figure 13.11 shows the observed throughput in the absence of any bandwidth management. Figure 13.12 shows the observed throughput when using our bandwidth management architecture. Our scheme provides weighted fairness in throughput to each flow accessing the shared channel. Note that we are able to

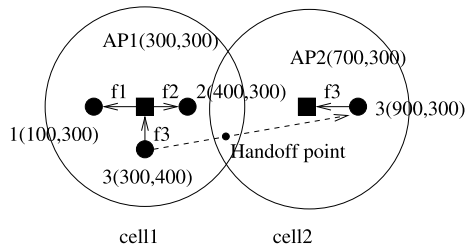


FIGURE 13.10: Illustrative example: no spatial reuse.

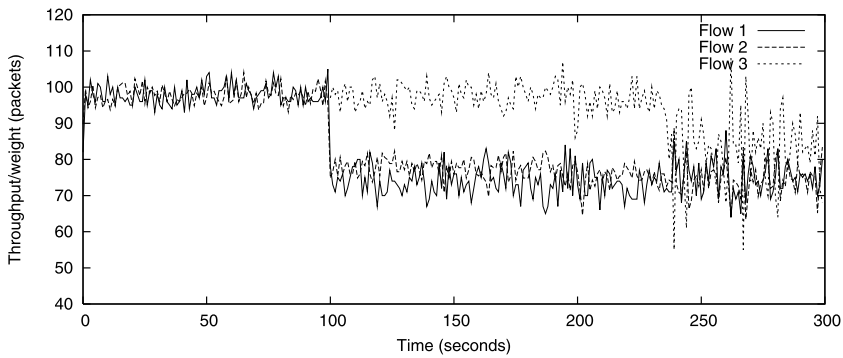


FIGURE 13.11: Observed weighted throughput without bandwidth management.

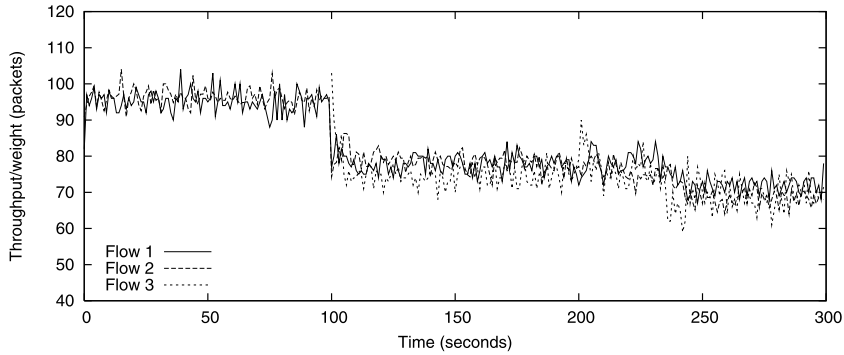


FIGURE 13.12: Observed weighted throughput with bandwidth management.

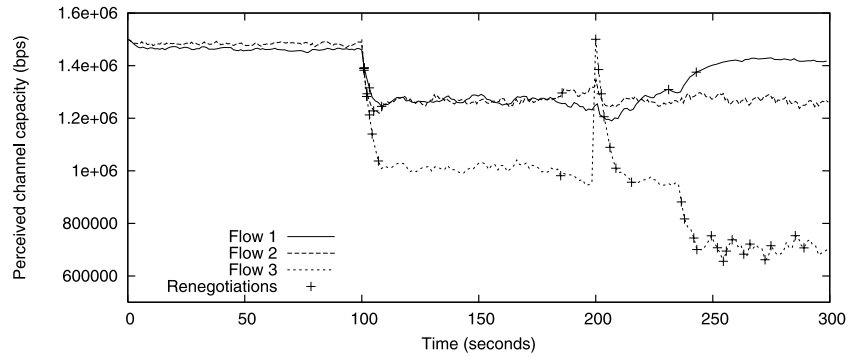


FIGURE 13.13: Perceived channel capacity for each flow.

provide weighted fairness in an extended LAN with multiple APs *without* changing the MAC protocol in any way. Figure 13.13 shows the channel capacity perceived (which takes into account time wasted) by each flow in the network. Figure 13.14 shows the fraction of unit time each flow is permitted to be active on the wireless channel. Flows with lower perceived channel capacity (i.e., worse channel quality) are allowed to spend more time on the channel, and vice versa. The accuracy of our channel quality estimation, even in the presence of hidden node effects, is illustrated by the fact that the allotted channel fractions exactly compensate channel quality variations and the result is a high degree of fairness in throughput among flows.

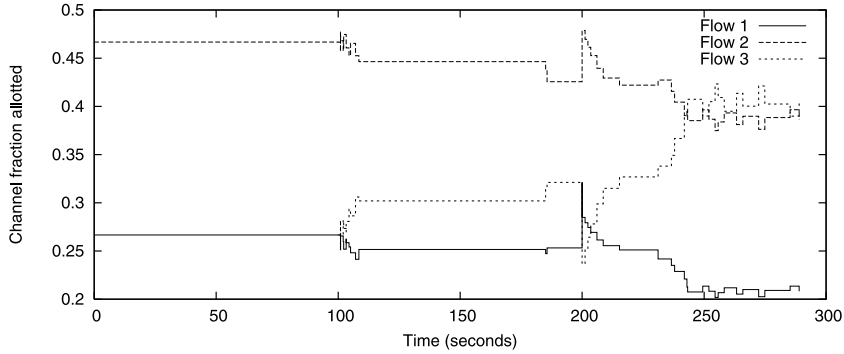


FIGURE 13.14: Channel fraction allotted to each flow.

Of course, throughput fairness is only one notion of fairness. Another notion of fairness is channel-time fairness, wherein all flows get equal access to the channel, and flows with better channel quality end up transmitting more packets successfully. Since we provide flows with worse channel quality more access to the channel, we use the channel less efficiently as a result. (In the ideal case, for maximum efficiency, only one flow should transmit on a completely clear channel, but obviously this starves all other flows and is hence not a practical solution.) In our scenario given earlier, we observe up to 15% drop in overall channel efficiency as compared to the baseline case without bandwidth management.

In cases with spatial reuse of the channel, the BM must arbiter *multiple* resources. It must identify the flows in a particular region that shares the wireless channel in that region of the network, and arbiter the channel among them. In another area of the network, a different set of nodes shares the wireless channel, and channel arbitration must be performed separately for that set of nodes. Details on identifying the flow sets that shares the channel, and on the bandwidth management in a scenario with spatial reuse, can be found in [39].

13.4.3 DiffServ: Proportional Delay Differentiation

13.4.3.1 Delay Management Architecture

Our second cross-layer design is a DiffServ QoS architecture that provides different delays for packets in different service classes [45]. The cross-layer architecture in Figure 13.15 operates from the MAC layer up to the application layer. At the network level, packets from different service classes are processed differently via per-hop forwarding mechanisms (e.g., packet scheduling and queue manage-

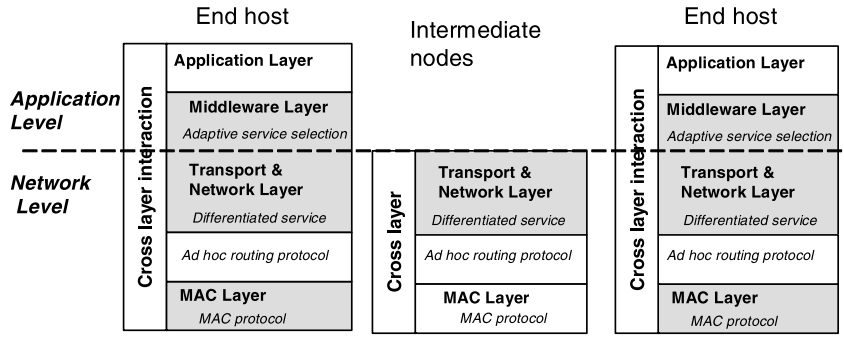


FIGURE 13.15: Delay management QoS architecture: overview.

ment). At the middleware level, a monitor component monitors the performances of applications. Based on the monitored results, it performs appropriate service class adaptation so that different applications are able to meet their required QoS specifications.

A detailed diagram of our delay management QoS architecture, which shows the key components of this architecture, is illustrated in Figure 13.16. In order to provide QoS support in the wireless networking environment, these components interact in the following way.

1. *At application level in the end hosts*:
    - The application notifies the *Adapter* in the middleware that it wishes to set up a flow between two end hosts. It also provides its QoS specification and adaptation policy to the *Adapter* in the middleware layer.
2. *At middleware level in the end hosts*:
    - Based on the previous performance of the service classes and the QoS specifications of the applications, the *Adapter* decides the appropriate service class for each application and notifies the *Classifier*. Adaptation is an application-specific process. Based on application-specific adaptation policy, actions are taken to adapt the application's service class.
    - The packets from applications are delivered through the middleware layer, where the *Classifier* marks the packets with their corresponding service class.
    - The *Monitor* monitors the performance of each service class and notifies the *Adapter* of the observed changes and QoS violations.

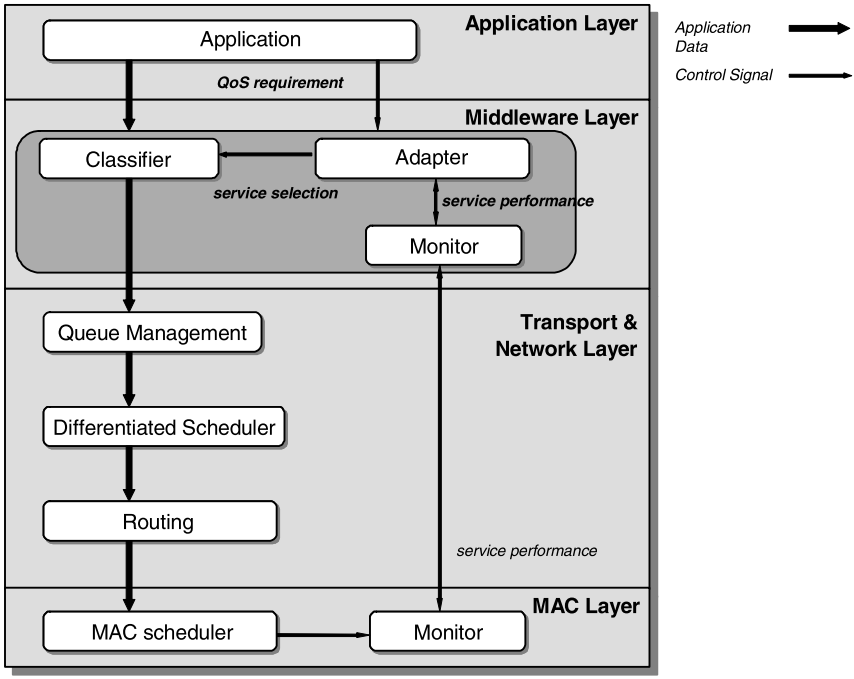


FIGURE 13.16: Delay management QoS architecture: details.

3. *At network level in routing nodes*:

   - The *Queue Management* component allocates buffer spaces and marks or drops packets. It deals with packet loss rate differentiation.
   - The *Differentiated Scheduler* selects a packet to transmit. It performs packet-level QoS enforcement, allocates bandwidth for different flows, and provides delay differentiation.

This architecture balances very well between architectural flexibility and scalability. At the network level, the service differentiation mechanisms work to bring scalability with per-class packet scheduling and queue management. At the middleware level, the individual QoS requirement of each application is met via the application-specific adaptation process.

In the following, we discuss details of the cross-layer proportional delay differentiation scheduler and the adaptation service at the middleware layer.

13.4.3.2 Cross-Layer Proportional Delay Differentiation Scheduler

The model of the proportional service differentiation was first introduced as a per-hop-behavior (PHB) for DiffServ in wireline networks [12]. It states that certain class performance metrics should be proportional to the differentiation parameters. In particular, if we consider the case of delay differentiation in a network with C service classes, the proportional delay differentiation model imposes the following constraints for all pairs of classes:

$$\frac{\bar{d}_i(t, t+\tau)}{\bar{d}_j(t, t+\tau)} = \frac{\delta_j}{\delta_i}, \quad \text{for all } i \neq j \text{ and } i, j \in \{1, 2, \ldots, C\}, \tag{13.12}$$

where δ_i is the service differentiation parameter for class i and $\bar{d}_i(t, t+\tau)$ is the average delay for class i, $(i = 1, 2, \ldots, C)$ in the time interval $(t, t+\tau)$, where τ is the monitoring timescale.

The basic idea of proportional differentiation is that even though the actual quality level of each class may vary with traffic loads, the quality ratio between classes should remain constant in various timescales. In addition, such a quality ratio can be controlled by setting the service differentiation parameters, which provide flexible class provisioning and management. Under certain conditions (i.e., the network is well provisioned), applications with absolute delay requirements can select appropriate service classes to meet their requirements [13], even though the network offers only relative differentiation.

One of the packet scheduling algorithms that can realize the proportional delay differentiation model in a short timescale is the *waiting time priority* (WTP) scheduler [14]. In this algorithm, a packet is assigned with a weight, which increases proportionally to the packet's waiting time. Service classes with higher differentiation parameters have larger weight-increase factors. The packet with the largest weight is served first in nonpreemptive order. Formally, if $wt_{pkt}(t)$ is the waiting time of a packet pkt of class i at time t, define its *normalized waiting time* $\hat{wt}_{pkt}(t, i)$ at time t to be

$$\hat{wt}_{pkt}(t, i) = wt_{pkt}(t) \cdot \delta_i. \tag{13.13}$$

The normalized waiting time is then used as the weight for scheduling. The packet with the largest weight is then selected by the WTP scheduler for transmission. Formally, at time t it will transmit the packet pkt that satisfies

$$pkt = \arg \max_{pkt \in \mathcal{P}} \hat{wt}_{pkt}(t, i), \tag{13.14}$$

where $\mathcal{P}$ is the set of backlogged packets. It is shown that the WTP scheduler is able to approximate the proportional delay differentiation model in wireline networks under heavy traffic condition [14].

Here we introduce the *proportional service differentiation model* into the domain of wireless LANs. In contrast to wireline networks, in which flows through a router contend with each other on the outgoing link, in wireless LANs, not only do flows originating from a node contend with each other, but they also contend with flows originating from other nodes. To extend the concept of proportional service differentiation to wireless LANs, flows originating from different nodes must be considered. To address this, our proportional delay differentiation model for wireless LANs states that the relation (13.12) holds for all flows within the wireless LAN no matter whether they originate from the same node or not.

As a result of the distributed medium sharing, packet scheduling needs cooperation among all the nodes. This is in contrast to wireline networks where packets that need to be scheduled originate from the same router, and hence the packet scheduling decision can be made by the router itself only considering its own packets. We argue that delay differentiation in wireless LANs can only be achieved through a *joint packet scheduling* at the network layer and distributed coordination at the MAC layer. Therefore, we present a *cross-layer waiting time priority scheduling (CWTP) algorithm* that is able to achieve proportional delay differentiation in wireless LANs.

The CWTP algorithm divides the scheduling task into two parts, which are performed at two layers in the network stack. At the network layer, *intra-node* scheduling at node n selects a packet pkt_n^* with the longest normalized waiting time, that is, a packet pkt_n^* that satisfies

$$pkt_n^* = \arg \max_{pkt \in \mathcal{P}_n} \hat{wt}_{pkt}(t, i), \tag{13.15}$$

where $\mathcal{P}_n$ is the set of all backlogged packets at node n. At the MAC layer, *inter-node* scheduling selects a packet pkt^* among the packets pkt_n^*, which satisfies

$$pkt^* = \arg \max_{pkt_n^* \; n \in \mathcal{N}} \hat{wt}_{pkt_n^*}(t, i), \tag{13.16}$$

where $\mathcal{N}$ is the set of wireless nodes.

Such an intra- and inter-node scheduling algorithm can fit well the environment of wireless LANs. In particular, the intra-node scheduling can be implemented via network layer packet scheduling at each individual node and the inter-node scheduling can be implemented via medium access control that coordinates packet transmissions among nodes. Figure 13.17 illustrates such a cross-layer scheduling architecture. In this architecture, the packet scheduler at the network layer

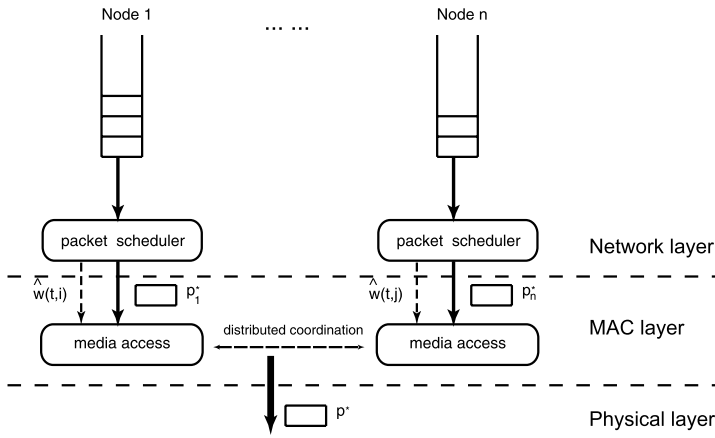


FIGURE 13.17: Cross-layer architecture.

and the distributed coordination function at the MAC layer are coordinated using normalized packet waiting time $\hat{wt}$ as a cross-layer signal.

At the MAC layer, in order to transmit the packet with the largest normalized waiting time before ones with smaller normalized waiting times, we map the normalized waiting time $\hat{wt}$ to the backoff time b via function $b = \Phi(\hat{wt})$. In [45], we present two mapping schemes, namely linear mapping and piecewise linear mapping to implement the function $\Phi(\hat{wt})$.

In the linear mapping scheme, the normalized waiting time of a packet is mapped to its MAC layer backoff time via a linear function. Formally, let us consider a linear function $\phi(x) : \Re^+ \rightarrow \Re$,

$$\phi(x) = \beta - \alpha \cdot x, \tag{13.17}$$

where $\alpha, \beta > 0$ are parameters of this linear function. To ensure it is a nonnegative integer, the backoff time b (in numbers of time slots) of a packet with normalized waiting time $\hat{wt}$ is chosen as

$$b = \Phi(\hat{wt}) = \lceil [\phi(\hat{wt})]^+ \rceil, \tag{13.18}$$

where $[x]^+ = \max(0, x)$ and $\lceil \cdot \rceil$ is the ceiling operation. These two operations round up the value of $\phi(\hat{wt})$ to a nonnegative integer. It is obvious that α and β determine the effectiveness of the mapping function, and thus the performance of the cross-layer scheduling algorithm. We present a dynamic tuning algorithm for α and β. Let $\overline{cw}$ be the expected value of the contention window under

IEEE 802.11 DCF without differentiation. The backoff time b is uniformly chosen from $[0, \overline{cw})$. Let $\hat{wt}_{\max}$ and $\hat{wt}_{\min}$ be the maximum and minimum normalized waiting times, respectively. Preferably, the maximum normalized waiting time $\hat{wt}_{\max}$ can be mapped to the smallest backoff time (0) for efficient channel utilization, and $\hat{wt}_{\min}$ can be mapped to $\overline{cw}$ for similar contention behavior as IEEE 802.11 without differentiation.

The linear mapping scheme neglects the fact that the distribution of the normalized waiting time can be nonuniform. If there is a higher density over a certain interval of time, then it will increase the possibility of packets with different normalized waiting times being mapped into the same backoff time. It can also increase the possibility of packet collision at the MAC layer. To address these problems, we present a piecewise linear mapping scheme that considers the effect of the normalized waiting time distribution. In the piecewise linear mapping scheme, the normalized waiting times $\hat{wt}$ are divided into L intervals of equal lengths defined by points $\hat{wt}_{\min} = \hat{wt}_0, \hat{wt}_1, \hat{wt}_2, \ldots, \hat{wt}_L = \hat{wt}_{\max}$. During each interval, a function $\Phi_i(\hat{wt}) = \lceil [\beta_i - \alpha_i \cdot \hat{wt}]^+ \rceil$ will be used for the mapping. Figure 13.18 compares these two mapping algorithms.

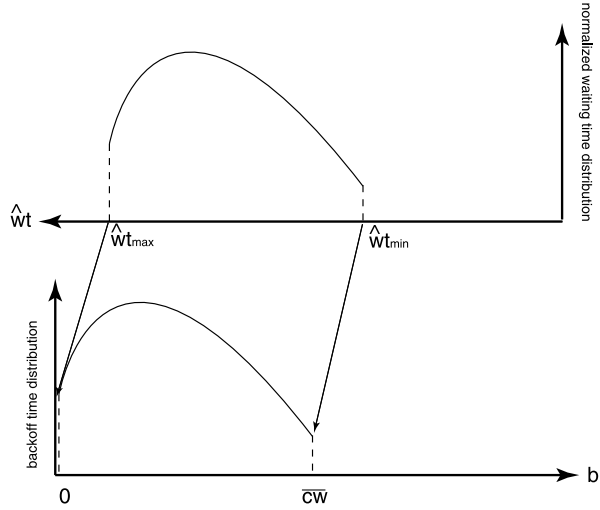
We simulate the CWTP algorithm under both linear mapping and piecewise linear mapping schemes on a variety of network settings in ns-2 [41]. In the simulation, the number of nodes (N) is a parameter to show how CWTP scales to the network size. Each node in the wireless LAN sets up a connection. The transmission rate of each flow is configured to give the network an aggregated load of about 1500 Kbps.

We first show the impact of network size on the CWTP algorithm. In this experiment, two service classes with $\delta_2/\delta_1 = 2$ are supported in the network. Figure 13.19 shows the differentiation index I with different numbers of nodes in the network. The differentiation index (I) is defined as the ratio of the average delay of the two service classes. That is,

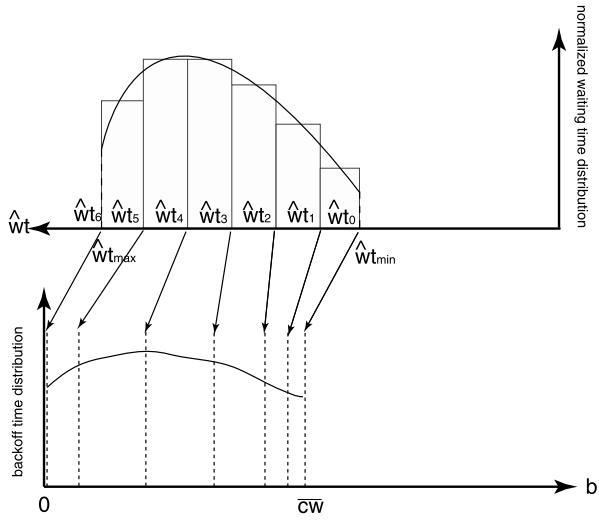
$$I = \frac{\bar{d}_1}{\bar{d}_2}, \tag{13.19}$$

where $\bar{d}_i$ is the expected packet delay of service class i. This metric shows the effectiveness of the service differentiation—how close the differentiation result matches the differentiation goal. Ideally, in these experiments $I = 2$. We observe that both linear mapping and piecewise linear mapping schemes can lead the CWTP scheduling algorithm to achieve a delay differentiation index very close to the target value, when the network size is relatively small (the number of nodes $N < 20$). When the network size is large (e.g., $N = 50$), the piecewise linear mapping scheme performs much better than the linear mapping scheme.

In Figure 13.20, we show the instantaneous delay behaviors under these two schemes when $N = 10$. From these results, we observe that the piecewise linear



(a) Linear mapping



(b) Piecewise linear mapping

FIGURE 13.18: Linear mapping and piecewise linear mapping: a comparison.

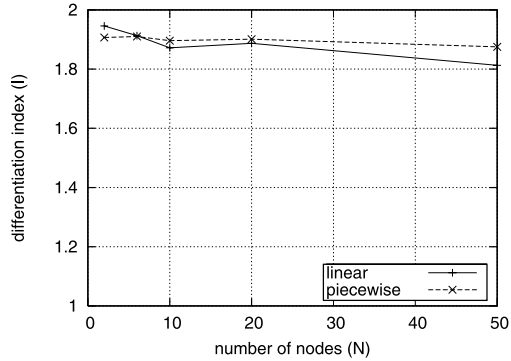
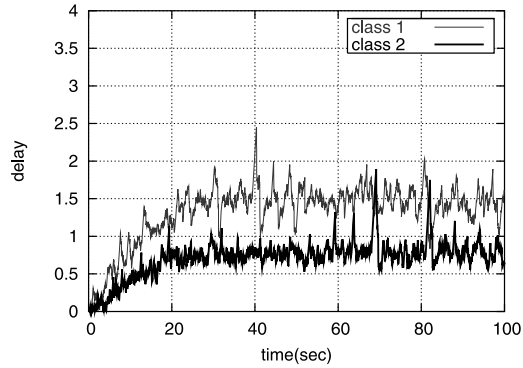
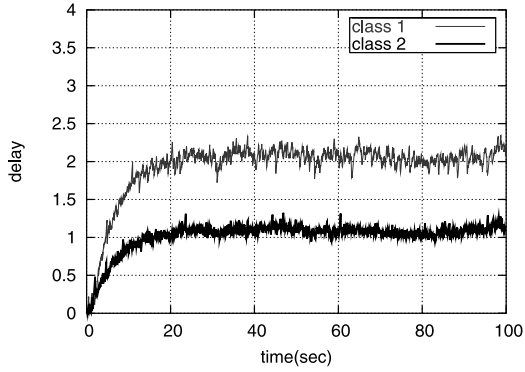


FIGURE 13.19: Differentiation index.



(a) Linear mapping



(b) Piecewise linear mapping

FIGURE 13.20: Instantaneous delay behavior.

mapping scheme gives much more consistent and smooth delay behavior than the linear mapping scheme. This is because with the consideration of the normalized waiting time distribution, piecewise linear mapping significantly reduces the possibility of packet collision at the MAC layer.

13.4.3.3 Middleware-Based Adaptation Services

Now we describe the adaptation services to be provided by the middleware framework. The adaptation services work with the network level service differentiation mechanism to provide an *absolute* QoS level for applications. At the network level, service differentiation provides differentiated quality for packets from different classes. However, applications usually require a QoS level with an absolute value; hence the middleware is responsible for mapping the required QoS level to the correct service class. Our middleware achieves this goal by continually monitoring the performances of the applications and adaptively adjusting their service classes to meet their required QoS levels.

The design of our middleware adaptation framework is based on a task control model as shown in Figure 13.21a. Within the middleware control framework, the *Adaptation Task* and the *Observation Task* are represented in two respective components: the *Adapter* and the *Monitor*. The *Target System* is the differentiated network, represented by the *Classifier* in the middleware layer, as shown in Figure 13.21b. The *Control Action* is the service class selection, and *Task States* are the end-to-end performance of the multimedia application. In particular, the *Adapter* takes the end-to-end delay observed by the *Monitor* as its input, makes the service class selection decision based on the input values, and sets the service class at the classifier as its output. It is controlled by a set of conditional statements in the form of if-then rules. In [32], we presented the detailed design of rules. An example rule is illustrated as follows, where d is the current observed delay of the application, d^* is its delay bound, and $\delta(t)$ is its service differentiation parameter

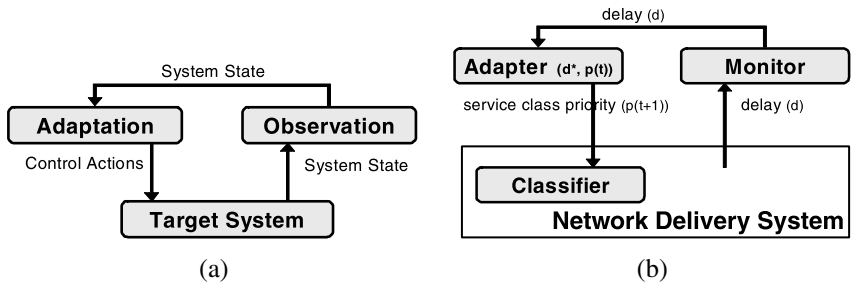


FIGURE 13.21: Middleware control framework.

at time t:

$$\text{If} \quad (d > 2.5d^*) \tag{13.20}$$

$$\text{then} \quad \delta(t+1) = 2\delta(t). \tag{13.21}$$

We show the performance of the adaptation service integrated with the delay differentiation service over an IEEE 802.11-based wireless ad hoc test bed implementation. In the experiment, we first start an audio application that has a QoS requirement in terms of maximum packet delivery delay. Then background UDP traffic with 15,000 Bytes/s is started. From the results in Figure 13.22, we see that the average delay increases quickly from 70 to 800 ms without service differentiation and adaptation. Using the service adaptation policy in the example and the underlying delay differentiation support, we observe that the average delay for the audio application was successfully bounded to < 150 ms.

13.4.4 Comparison of QoS Architectures

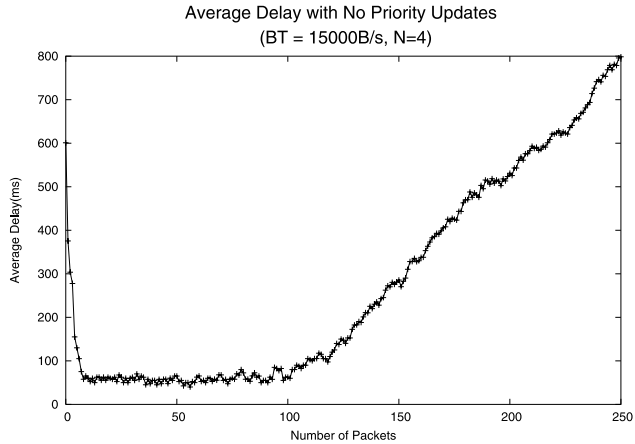
The two QoS architectures described earlier support different QoS models. BM supports the IntServ model by admission control, bandwidth reservation, traffic shaping, and bandwidth renegotiations. Proportional delay differentiation supports the DiffServ model by a special per-hop forwarding behavior that relies on a joint scheduling algorithm at the MAC and network layers.

Each of these QoS architectures has its own strength and weakness. For example, it is convenient for BM to provide a per-flow "soft" bandwidth guarantee, but a flow may be rejected in admission or terminated during transmission. In the delay differentiation architecture, every flow can always send out packets, but the quality protection between different classes of packets is only "relative." Each application takes the risk and burden of choosing an appropriate service class to meet its own needs. Therefore, BM is more suitable for a small number of concurrent flows with stringent QoS requirements, whereas delay differentiation is better for a large number of flows where a few of them are QoS sensitive while the rest are not.

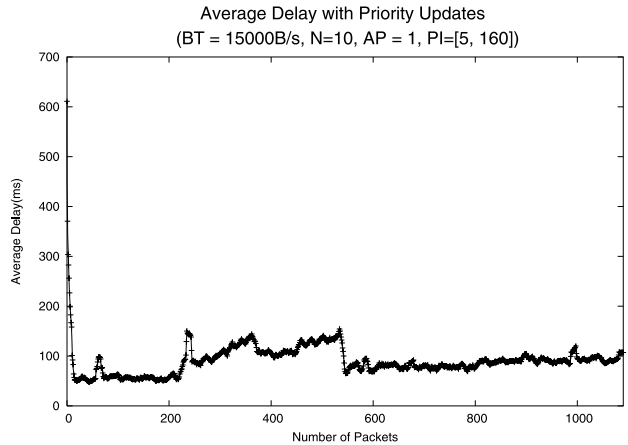
Despite differences in their QoS models, there is a common trait in these two architectures, which is the *cross-layer* design principle. In both architectures, there is a close interaction among application, middleware, network, and MAC layers. Together they provide an agile adaptation framework for QoS applications in wireless networks.

13.4.5 Beyond Single-Hop Wireless Networks

The two QoS architectures discussed earlier assume a single-hop ad hoc network (or wireless LAN) where each node can talk to each other directly. In this section



(a) Without service adaptation and differentiation



(b) With service adaptation and differentiation

FIGURE 13.22: Performance comparison.

we discuss how to support multimedia applications in a *multi-hop* ad hoc network (or "MANET").

Running multimedia applications over a MANET has even more challenges: (1) the network topology is dynamic, which often results in route breakage and rerouting and (2) wireless resource usage is very dynamic and complex due to location-dependent wireless contention and spatial reuse. Examples of QoS support architectures in this network include INSIGNIA [20] and SWAN [2].

INSIGNIA supports the IntServ model by reserving bandwidth over a multi-hop path and continually renegotiating the reservations via signaling. SWAN supports the DiffServ model by differentiating two classes of traffic: real time and best effort. Real-time traffic needs to go through a distributed admission control process at a flow's start-up and needs to monitor the available bandwidth of the path continuously.

Due to the dynamic nature of the multi-hop ad hoc network, robust QoS support is very difficult. Hence we ask another interesting question: how can we better support multimedia flows as part of the *best-effort* traffic in MANET? We are not concerned about any QoS model, but we are interested in how flow control at the transport layer can facilitate the transmission of multimedia traffic. Traditional flow control such as TCP relies on "probing" the network until packet lost is observed. This is certainly not an appealing method to carry multimedia traffic because frequent and large rate fluctuations are inevitable, especially in a wireless environment.

To this end, we study a special *explicit* flow control scheme called "EXACT" [10] where the transport layer gives explicit rate signals to the application layer. Its design rationale is as follows:

- *Router-Assisted Flow Control*: In our framework, the router explicitly gives rate signals to the flows that are currently passing it, since routers are in a better position to react to network bandwidth variations and route changes in MANET.
- *Rate-Based Transmission*: In our framework, the sender follows the rate information set by the routers, and hence the packet transmission is rate based.
- *Feasibility in MANET*: Our framework incurs additional complexity and overhead at the routers. It is *not* targeted for the large-scale Internet (where core routers have to process huge numbers of concurrent flows), but rather as a solution for the smaller scale MANET environment.

13.4.5.1 Overview

An overview of the EXACT framework is shown in Figure 13.23a. Each data packet carries a special IP header, called a *flow control header*, which is modified by the intermediate routers to signal the flow's allowed sending rate. When the packet reaches the destination, the explicit rate information is returned to the sender in a feedback packet. As a result, any bandwidth variation along the path will be returned to the sender within one RTT.

In the event of rerouting (Figure 13.23b), the first data packet traveling through the new path (R_1, R_2, R_3') collects the new allowed rate of the flow. As a result, the sender learns the exact sending rate after only one RTT of delay after rerouting, without having to go through the additive probing phase of TCP.

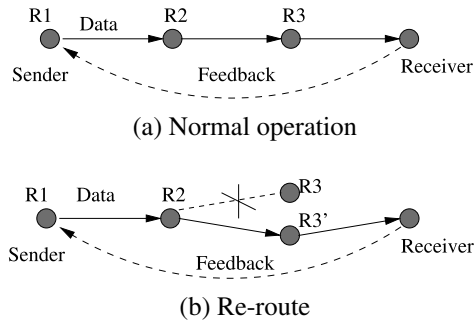


FIGURE 13.23: Overview of the EXACT flow control scheme.

A packet's flow control header includes two fields: *Explicit Rate* (ER) and *Current Rate* (CR). ER is the allowed sending rate of a flow. It is initially set at the sender as its maximum requested rate and is subsequently reduced by the intermediate routers to signal its allowed data rate. CR is initially set at the sender as its current sending rate and is modified by the intermediate routers to signal possible rate reduction along the path. Each router remembers the CR of the current flows in its *flow table* in order to compute each flow's fair share of bandwidth.

13.4.5.2 Router's Behavior

A router plays the central role in EXACT. A router has four major tasks: (1) keep track of current flows and their sending rates; (2) measure the current bandwidth of the outgoing wireless links; (3) compute rates for the current flows; and (4) update the header of each passing data packet.

The core part of each router is its rate computation algorithm to allocate sending rates for the competing flows. The rate computation, performed locally, is based on the current measured bandwidths of the outgoing links, as well as the current rates of the flows going through the router. Efficiency is achieved by making sure that the flows can fully occupy the outgoing wireless links. Fairness can be achieved by allocating the bandwidth "fairly" to each flow. A common fairness criterion is *max–min fairness* [5]. In max–min fairness, flows with minimum requests are granted their requests first; the remaining bandwidth resource is then divided evenly among the higher demanding flows.

Here we propose to maintain fairness among competing flows according to their *channel time* demands to access the wireless channel. The wireless link's bandwidth at the MAC layer is measured using the monitor as described earlier. To represent a flow's resource request, we normalize a flow's requested rate to its next-hop link's bandwidth as $TF_i = r_i/b_i$, where r_i is the flow's data rate and b_i is the current bandwidth of the link. The max–min allocation is then performed

on top of the requests of the flows: TF_i, $i = 1$ to N. Since each flow obtains a throughput proportional to its next-hop link's bandwidth, we call it *bandwidth-proportional max–min fair*. For details of rate computation, interested readers are referred to [10].

13.4.5.3 Multimedia Streaming Using EXACT

EXACT provides explicit rate signals for the flows, but these rate signals may be fluctuating. In order to support multimedia streaming on top of EXACT, our framework supports *split-level* adaptations. At the transport layer, EXACT provides explicit rate signals to the upper applications. It serves as the upper bound of the application's sending rate. Within this upper bound, each application may adjust its own sending rate based on its adaptation policies, for example, to maintain smooth rate changes for multimedia flows.

Such *informed* adaptation is possible only with EXACT's explicit rate signals. Although all the flows are treated as best effort at the transport layer, using EXACT as the flow control scheme facilitates running multimedia applications over MANET.

13.4.5.4 Evaluations

Here we show the efficiency of EXACT compared to traditional TCP flow control. Using the ns-2 simulator, we create a MANET with 30 nodes moving in a 1500-m by 300-m space with a maximum speed of 20 m/s and different pause times (0, 5, 10, 15, and 20 s) to create different levels of network dynamics. Under these mobility patterns, we compare EXACT with TCP-Reno and TCP-SACK. For each scenario, we average the total number of reliably transmitted packets over 10 runs for each scheme. The results in Figure 13.24 show that under all mobility

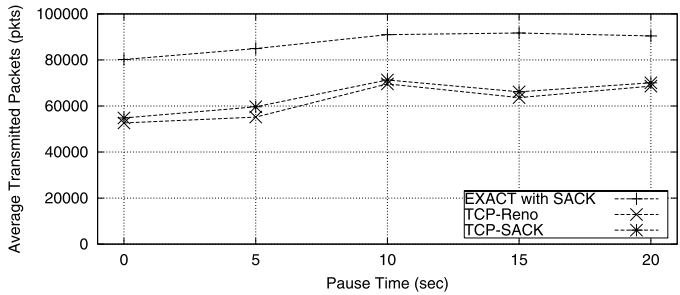


FIGURE 13.24: Comparison of EXACT with TCP under different mobility patterns.

scenarios, EXACT overall outperforms TCP-Reno and TCP-SACK by 42 and 36% more packets, respectively. This demonstrates the effectiveness of the EXACT flow control scheme in a dynamic MANET environment.

13.5 DESIGN PRINCIPLES LEARNED

In this section we summarize various design principles we have learned in our study.

13.5.1 Cross-Layer Strategies

Cross-layer resource management strategies in wireless networks have attracted increasing attention in recent years.[1] The need for cross-layer design is based on two characteristics of wireless networks. First, the wireless medium is a shared medium. Sending a packet from one node to another creates interference to other nodes in the same neighborhood. Therefore, in designing network packet scheduling algorithms, we have to consider the interaction between the network layer and the MAC layer due to wireless interference. At the network layer, only packets within the same host are scheduled; interhost packet transmissions can only be enabled at the MAC layer.

Second, resources are generally scarce and variable in wireless environments, and hence they must be managed carefully. For example, we translate multimedia application requirements into bandwidth and CPU resource requirements and use controllers at the lower level to monitor the load on the resource. The feedbacks from the controllers are then used to tune the network and CPU schedulers so as to satisfy multimedia application requirements. In this picture, the control flow of resource management is two way. Variations in the load on the resource are fed by the lower-level controllers to the application so that the application can adapt. The chosen operating quality level of the multimedia application is fed to the lower-level controllers so that the schedulers can be tuned accordingly.

13.5.2 Tightly Coupled Resources

In wireless networks, resources are tightly coupled with each other. Therefore, we need to adopt *coordinated* resource management strategies because resources cannot be managed independently of each other.

[1] Interested readers are referred to [1] for more research results in cross-layer design for wireless networks.

For example, if a lot of bandwidth is available, a bandwidth management scheme may allot a high operating quality level to a media streaming application. However, if CPU resources are scarce, a CPU management scheme will allot the same application a lower operating quality level. Obviously, this is a contradictory scenario, hence the resource management strategy must be coordinated between resources.

A direct result of the tight coupling of resources means that we need a multilevel resource management strategy. For example, some resources are global to the hosts (nodes) in a wireless network, for example, network bandwidth. Other resources are global within a single host, for example, energy. Thus, a multilevel resource management strategy is to let the network-wide resource management constrain the host's resource availability and let the host-wide resource management constrain each application's resource availability.

13.5.3 Adaptation of Both Software and Hardware

Resource adaptation should not be limited to software only. To achieve greater flexibility, the middleware layer needs to consider the adaptability of both software and hardware. For example, adaptive hardware such as adaptive CPU and wireless network interface cards can trade off performance for energy consumption; multimedia applications can trade off quality for resource demand.

Software and hardware adaptation and optimization can take place at different time granularities. At a coarse time granularity, for example, when an application starts, we can optimize to achieve high application utility and desired battery lifetime. At a finer time granularity, for example, when the application renders a frame, we can optimize to save more energy.

13.5.4 Suitable QoS Model

Selecting a suitable QoS model is the most important step in designing a QoS support architecture because it has a fundamental impact on the overall architecture. This is especially true in wireless networks due to the scarcity of bandwidth resources.

While selecting a QoS model, we need to keep in mind the unique characteristics of wireless networks. The QoS models in the Internet, such as IntServ and DiffServ, should be carefully re-examined. For example, the main challenge of deploying IntServ over the Internet is the scalability problem in keeping per-flow state at the Internet routers. In contrast, the challenge of adopting IntServ in wireless networks is not the scalability problem; instead, it is the time-varying resource availability problem, which may result in repeated QoS setups. Relative DiffServ may be a more suitable QoS model here since it does not require the QoS setup phase and hence avoids the difficulty and overhead in doing repeated admission control and resource signaling.

13.6 SUMMARY AND FURTHER READING

In this chapter we discussed QoS support in mobile operating systems and mobile wireless networks for multimedia applications. We have shown that with careful OS design with respect to scheduling and dynamic voltage scaling, we can achieve deadline guarantees for wireless multimedia applications as well as energy efficiency of mobile nodes to extend the application lifetime. Furthermore, we have shown two cross-layer networking architectures that support statistical bandwidth and delay guarantees in cooperative wireless single-hop environments. The bandwidth management architecture realizes the IntServ QoS model, while the proportional delay differentiation architecture realizes the DiffServ model. We have shown that by leveraging the cross-layer design principle, both of them can achieve different levels of QoS protection and are suitable in many situations in wireless networks.

Wireless networks are at the critical junction of being widely accepted into everyday life by the proliferation of small wireless devices such as smart phones, as well as the maturation of VoIP software. Now many municipal governments are planning to roll out city-wide mesh 802.11 networks to the general public. Such networks are owned by a single entity, for example, the city government, so that cooperation among the nodes can be assumed. It is likely that the IntServ QoS model can be implemented by per-user bandwidth provisioning based on their subscriptions, considering the fact that the number of users and flows should be manageable for a city-scale network. Higher speed 802.11 standards such as 802.11n using MIMO are also going to alleviate the scarcity of wireless bandwidth. Wireless multimedia may become the next killer application, and QoS is certainly an important enabler in this picture.

Beyond the references cited in this chapter, the reader is recommended to read Chapter 4 on Bandwidth Adaptation Mechanisms, Chapter 10 on Network-Adaptive Media Transport, and Chapter 12 on Cross-Layer Wireless Multimedia.

REFERENCES

[1] *IEEE Transactions on Vehicular Technology, Special Issue on Cross-layer Design in Mobile Ad hoc Networks and Wireless Sensor Networks*, 55(3), May 2006.

[2] G.-S. Ahn, A. T. Campbell, A. Veres, and L.-H. Sun. "SWAN: Service Differentiation in Stateless Wireless Ad Hoc Networks." In *IEEE INFOCOM 2002*, New York, NY, June 2002.

[3] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," in *ACM SigComm*, Stanford, CA, 1996.

[4] B. Bensaou, Y. Wang, and C. Ko. "Fair Medium Access in 802.11 Based Wireless Ad Hoc Networks," in *IEEE MobiHoc*, Boston, MA, 2000.

[5] D. Bertsekas and R. Gallager. *Data Networks (2nd Ed.)*. Prentice-Hall, 1992.

[6] V. Bharghavan, K. W. Lee, S. Lu, S. Hu, J. R. Li, and D. Dwyer. "The Timely Adaptive Resource Management Architecture," *IEEE Personal Communication Magazine*, 5(4), August 1998.

[7] S. Brandt. "Performance Analysis of Soft Real-Time Systems," in *20th IEEE International Performance, Computing and Networking Conference (IPCCC 2001)*, April 2001.

[8] A. Chandra, M. Adler, P. Goyal, and P. Shenoy. "Surplus Fair Scheduling: A Proportional-Share CPU Scheduling Algorithm for Symmetric Multiprocessors," in *Proceedings of 4th Symposium on Operating System Design and Implementation*, San Diego, CA, October 2000.

[9] A. Chandrakasan, S. Sheng, and R. W. Brodersen. "Low-power CMOS Digital Design," *IEEE Journal of Solid-State Circuits*, 27:473–484, April 1992.

[10] K. Chen, K. Nahrstedt, and N. Vaidya. "The Utility of Explicit Rate-Based Flow Control in Mobile Ad Hoc Networks," in *Proc. of IEEE Wireless Communications and Networking Conference (WCNC 2004)*, Atlanta, GA, March 2004.

[11] H. H. Chu and K. Nahrstedt. "CPU Service Classes for Multimedia Applications," in *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 296–301, Florence, Italy, June 1999.

[12] C. Dovrolis and P. Ramanathan. "A Case for Relative Differentiated Services and the Proportional Differentiation Model," *IEEE Network*, 13(5):26–34, 1999.

[13] C. Dovrolis and P. Ramanathan. "Dynamic Class Selection: From Relative Differentiation to Absolute QoS," in *IEEE International Conference on Network Protocols*, 2001.

[14] C. Dovrolis, P. Ramanathan, and D. Stiliadis. "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling," *IEEE/ACM Transactions on Networking*, 10:12–26, February 2002.

[15] D. Eckhardt and P. Steenkiste. "Effort-Limited Fair (elf) Scheduling for Wireless Networks," in *IEEE InfoCom*, Tel Aviv, Israel, March 2000.

[16] K. Flautner and T. Mudge. "Vertigo: Automatic Performance-Setting for Linux," in *Proceedings of 5th Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002.

[17] T. Ishihara and H. Yasuura. "Voltage Scheduling Problem for Dynamically Variable Voltage Processors," in *Proceedings of International Symposium on Low-Power Electronics and Design*, Monterey, CA, 1998.

[18] M. Jones, D. Rosu, and M. Rosu. "CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities," in *Proceedings of 16th Symposium on Operating Systems Principles*, St-Malo, France, October 1997.

[19] V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, and E. Knightly. "Distributed Multi-hop Scheduling and Medium Access with Delay and Throughput Constraints," in *ACM MobiCom*, Rome, Italy, 2001.

[20] S.-B. Lee, G.-S. Ahn, X. Zhang, and A. T. Campbell. "INSIGNIA: An IP-Based Quality of Service Framework for Mobile Ad Hoc Networks," *Journal of Parallel and Distributed Computing*, 60(4):374–406, 2000.

[21] C. L. Liu and J. W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *JACM*, 20(1):46–61, January 1973.

[22] S. Lu, T. Nandagopal, and V. Bharghavan. "Design and Analysis of an Algorithm for Fair Service in Error-Prone Wireless Channels," *ACM Wireless Networks*, 6(4), 2000.

[23] H. Luo, P. Medvedev, J. Cheng, and S. Lu. "A Self-Coordinating Approach to Distributed Fair Queuing in Ad Hoc Wireless Networks," in *IEEE InfoCom*, Anchorage, AK, 2001.

[24] H. Luo, S. Lu, and V. Bharghavan. "A New Model for Packet Scheduling in Multihop Wireless Networks," in *ACM MobiCom*, Boston, MA, August 2000.

[25] L. Magalhaes and R. Kravets. "Mmtp: Multimedia Multiplexing Transport Protocol," in *Workshop on Data Communication in Latin America and Caribbean (SIGCOMM-LA)*, 2001.

[26] M. Mirhakkak, N. Schult, and D. Thomson. "Dynamic Bandwidth Management and Adaptive Applications for a Variable Bandwidth Wireless Environment," *IEEE Journal of Selected Areas in Communications*, 19(10), October 2001.

[27] Shivajit Mohapatra, Radu Cornea, Nikil Dutt, Alex Nicolau, and Nalini Venkatasubramanian. "Integrated Power Management for Video Streaming to Mobile Devices," in *Proceedings of ACM Multimedia*, Berkeley, CA, November 2003.

[28] M. Moser, D. Jokanovi, and N. Shiratori. "An Algorithm for Multidimensional Multiple-Choice Knapsack Problem," *IEEE Transactions on Fundamentals*, 80(2), March 1997.

[29] R. Neugebauer and D. McAuley. "Energy Is Just Another Resource: Energy Accounting and Energy Pricing in the Nemesis OS," in *Proceedings of 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Schloss Elmau, Germany, May 2001.

[30] J. Nieh and M. S. Lam. "The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications," in *Proceedings of 16th Symposium on Operating Systems Principles*, St-Malo, France, October 1997.

[31] Jason Nieh and Monica S. Lam. "A Smart Scheduler for Multimedia Applications," *ACM Transaction on Computer Systems*, 21(2):117–163, 2003.

[32] C. S. Ong, Y. Xue, and K. Nahrstedt. "A Middleware for Service Adaptation in Differentiated 802.11 Wireless Networks," in *Proc. of The Workshop on Coordinated Quality of Service in Distributed Systems (COQODS), held in conjunction with IEEE International Conference on Networks (ICON)*, Singapore, November 2004.

[33] C. Pereira, R. Gupta, P. Spanos, and M. Srivastava. "Power-Aware API for Embedded and Portable Systems." In R. Graybill and R. Melhem, editors, *Power Aware Computing*, pages 153–166. Plenum/Kluwer Publisher, 2002.

[34] P. Pillai, H. Huang, and K. G. Shin. "Energy-Aware Quality of Service Adaptation." Technical report CSE-TR-479-03, University of Michigan, 2003.

[35] D. Pisinger. "A Minimal Algorithm for the Multiple-Choice Knapsack Problem," *European Journal of Operational Research*, 83:94–410, 1995.

[36] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. "Resource Kernels: A Resource-Centric Approach to Real-Time Systems," in *Proceedings of SPIE Multimedia Computing and Networking Conference*, January 1998.

[37] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. "A Resource Allocation Model for QoS Management," in *Proceedings of 18th IEEE Real-Time Systems Symposium*, San Francisco, CA, December 1997.

[38] S. H. Shah, K. Chen, and K. Nahrstedt. "Dynamic Bandwidth Management for Single-Hop Ad Hoc Wireless Networks," *ACM/Kluwer Mobile Networks and Applications (MONET) Journal*, 10(1), 2005.

[39] S. H. Shah and K. Nahrstedt. "Channel-Aware Throughput Fairness in Multi-cell Wireless Lans," in *Proc. IEEE VTC'04-Fall*, Los Angeles, California, September 2004.

[40] P. Sinha, T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Bharghavan. "WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks," in *ACM MobiCom*, Seattle, WA, August 1999.

[41] Web Site. The network simulator ns-2. *http://www.isi.edu/nsnam/ns/*, 2005.

[42] R. Steinmetz and K. Nahrstedt. *Multimedia Systems*. Springer Verlag, Heidelberg, 2004.

[43] H. Tokuda and T. Kitayama. "Dynamic qos Control Based on Real-Time Threads," in *3rd International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, November 1993.

[44] N. Vaidya, P. Bahl, and S. Gupta. "Distributed Fair Scheduling in a Wireless Lan," in *ACM MobiCom*, Boston, MA, August 2000.

[45] Y. Xue, K. Chen, and K. Nahrstedt. "Achieving Proportional Delay Differentiation in Wireless LAN via Cross-Layer Scheduling," *Journal of Wireless Communications and Mobile Computing, special issue on Emerging WLAN Technologies and Applications*, 4(8):849–866, 2004.

[46] Y. Xue, B. Li, and K. Nahrstedt. "Optimal Resource Allocation in Wireless Ad Hoc Networks: A Price-Based Approach," *IEEE Transactions on Mobile Computing*, 5(4):347–364, April 2006.

[47] Y. Yang and R. Kravets. "Throughput Guarantees for Multi-priority Traffic in Ad Hoc Networks," in *IEEE MASS*, Fort Lauderdale, FL, October 2004.

[48] W. Yuan and K. Nahrstedt. "Integration of Dynamic Voltage Scaling and Soft Real-Time Scheduling for Open Mobile Systems," in *Proceedings of 12th International Workshop on Network and OS Support for Digital Audio and Video*, Miami Beach, FL, May 2002.

[49] W. Yuan and K. Nahrstedt. "Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems," in *Proceedings of 19th Symposium on Operating Systems Principles*, Bolton Landing, NY, October 2003.

[50] W. Yuan and K. Nahrstedt. "Energy-Efficient CPU Scheduling for Multimedia Applications," *ACM Transactions on Computer Systems*, 24(3):1–40, 2006.

[51] W. Yuan, K. Nahrstedt, S. Adve, D. Jones, and R. Kravets. "Grace-1: Cross-Layer Adaptation for Multimedia Quality and Battery Energy," *IEEE Transactions on Mobile Computing*, 5(7):799–815, 2006.

[52] H. Zeng, X. Fan, C. Ellis, A. Lebeck, and A. Vahdat. "ECOSystem: Managing Energy as a First Class Operating System Resource," in *Proceedings of 10th Intl. Conf. on ASPLOS*, pages 123–132, San Jose, CA, October 2002.

PART **E**

SYSTEMS

This page intentionally left blank

14

Streaming Media on Demand and Live Broadcast

Philip A. Chou

14.1 INTRODUCTION

Media on demand is a user scenario epitomized by playing back audio or video locally from a CD or DVD, whereas *live broadcast* is a user scenario epitomized by tuning in to a radio or television program. In the former scenario, the user has control over the start time for specific content, and in addition may have various other interactive controls (fast forward, pause, seek, etc.). In the latter scenario, the user simply joins an ongoing session and has little control except the ability to leave. While in the session, the user hears and sees the same content at the same time as other users in the session.

Today it is common to see both of these scenarios fulfilled by content delivered over the Internet. A subscriber to any of a number of music services, for example, can click on any of millions of songs and hear them on demand. Numerous Internet radio stations have sprung up, offering broadcast content either free or by subscription. Video content is also popular, offering news shorts, movie trailers, and so forth on demand, as well as live news feeds available at various web sites.

How are these scenarios enabled, technically? This chapter pulls together components from the previous chapters, while adding fundamental elements such as buffering, to outline the construction of systems for streaming media on demand and live broadcast over the Internet and over other IP networks such as wireless networks within the home.

Section 14.2 provides an overview of architectures, protocols, and format issues. Section 14.3 covers buffering and timing fundamentals. Section 14.4 details how media data may be communicated in a system for streaming media on

demand. Section 14.5 details how media data may be communicated in a system for live broadcast.

14.2 ARCHITECTURES, PROTOCOLS, AND FILE FORMATS

In this section, we review the basic architectures of systems for streaming media on demand and live streaming, covering the roles of the encoder, media file, server, network, buffer, and client. We introduce basic elements of the communication protocols required for streaming media on demand. These elements come in layers and include (from top to bottom) content discovery; file specification and interactive control (start, stop, pause, fast forward, seek); stream selection and coding rate control; congestion control (transmission rate control); and the transport protocol. We mention the continuum between sender-driven and receiver-driven protocols, and we discuss some of the related standards (RTP, RTSP, etc.). Finally we cover the basic elements of file formats: header information, streams, data units, and indexing, and we comment on some of the related standards (MPEG4, QuickTime, ASF, etc.). We also discuss content format in general terms: multi bit rate (MBR) coding vs. scalable coding, the details for which we refer to Chapters 5 and 6.

14.2.1 Architectures

Streaming media on demand and live broadcast require somewhat different architectures, as depicted in Figure 14.1. Figure 14.1a pertains to streaming media on demand, while Figure 14.1b pertains to live broadcast. In streaming media on demand, a source of media is encoded off line and the encoded source is placed into a media file. The format of the media file may be specialized to support various modes of streaming, as discussed in Section 14.2.3. The media file is placed

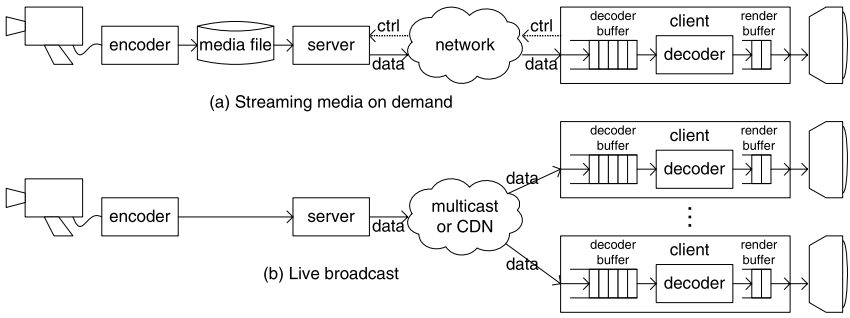


FIGURE 14.1: (a) Streaming media on demand. (b) Live broadcast.

in a location on the server from which it can be streamed. Various protocols between the server and the client, as discussed in Section 14.2.2, are used to stream the media across the network to the client. During streaming, the client temporarily buffers the encoded media data in a *decoder buffer* before decoding and then temporarily buffers the decoded media data in a *render buffer* before rendering the media, or presenting the media to the user. The render buffer is usually fairly short—a frame or two—and is used to buffer the relatively large decoded frames after a variable amount of decoder computation time. The decoder buffer, however, is usually relatively long and is used for a variety of purposes: network jitter compensation, error recovery, bandwidth management, and variable rate coding. Indeed the decoder buffer is a key element in streaming media on demand and is usually simply called the *client buffer*. In some client devices, such as desktop computers, the client buffer can be quite large, indeed, large enough to store the entire media content, such as a movie. In other client devices, such as mobile phones and consumer electronics, the client buffer may be fairly limited, capable of storing at most only a few seconds of encoded media content. In all cases, the user is generally able to control the experience through VCR-like commands such as play, fast forward, stop, and seek. Communication between the server and the client can be tailored to the resources of the client and to the network connection between the server and the client.

In streaming media on demand, because the entire file is available to the server ahead of time and because the server can individualize streaming to each client, there is great flexibility in which parts of the media file are transmitted at any given time. For example, if the client buffer is sufficiently large and the network bandwidth to the client is sufficiently large, then the server can look arbitrarily far ahead, streaming the media to the client faster than real time, essentially downloading the media file to the client while the client is simultaneously playing back the content in real time. This is sometimes called *progressive downloading* instead of streaming, although it is really just an extreme case of streaming with a large client buffer and a network bandwidth that is larger than the bit rate at which the content is encoded—the latter henceforth is known as the *source coding rate*. If the file format satisfies some basic properties, such as the ability to be decoded sequentially, then progressive downloading can be accomplished using any of a number of simple file transfer protocols, such as FTP over TCP/IP or HTTP over TCP/IP. Thus, progressive downloading can often be done using an ordinary FTP or web server. Even if the client has a limited media buffer, progressive downloading over TCP/IP can be done using simple TCP flow control. Specifically, the client can accept data from its TCP connection if and only if there is space in its media buffer. This technique was popularized by SHOUTcast, an early streaming music service [30].

Progressive downloading is a special case of streaming media on demand, which works only if the network bandwidth (specifically, the TCP fair share of

the path between the server and the client) is larger than the source coding rate, on average. Of course, the network bandwidth may fluctuate widely, as competing communication processes begin and end or even (if wireless networks are involved) when there are interfering elements, such as people walking near an antenna, turning on a microwave oven, or picking up a cordless phone. Furthermore, a user generally wants an average quality commensurate with the highest possible source coding rate, not a source coding rate less than the worst-case network bandwidth. Hence, it is generally desirable or necessary to adapt the source coding rate to the available transmission rate. Herein lie many of the intricacies of streaming media on demand, affecting the communication protocol between the client and the server (covered in Section 14.2.2), file format (covered in Section 14.2.3), and rate control (covered in Section 14.4).

In contrast, in live broadcast, as depicted in Figure 14.1b, the encoder may be directly connected to the server through an encoder buffer. To maintain a fixed and acceptably short end-to-end delay, the encoder buffer must contain only a limited amount of data. Thus the server can access data only so far ahead of the client's playback point, rather than access arbitrary data in a file. This restricts adaptivity of what the server can transmit to the client. Furthermore, in live broadcast, the server ordinarily communicates to multiple clients simultaneously through a multicast or content distribution network of some kind. (See Chapter 19 for more information on infrastructure-based content distribution systems.) Thus, in live broadcast, it is generally not possible for the server to give clients VCR-like interactive access to the media.[1] Furthermore, in live broadcast, it is generally difficult for the server to adapt its transmission rate to the bandwidths of particular clients. However, some adaptation is possible, using, for example, receiver-driven layered multicast (RLM) [28,29]. Finally, in live broadcast, it is generally difficult for the server to use retransmission-based error control due to the so-called negative acknowledgment (NAK) implosion problem, in which the server would potentially have to handle retransmissions to a huge number of clients. Hence, error control becomes an especially acute issue for live broadcast. Section 14.5 is dedicated to all of these issues. Thus, in the following, up until Section 14.5 we will consider only streaming media on demand.

There has been a fair amount of work on video "on demand" systems that, rather than devoting a unique stream to each user, multicast a relatively small number of time-staggered streams to an unlimited number of users who may start the video on demand or nearly on demand [4]. Such systems work by, for example, devoting some of the multicast streams to enabling users to "catch up" to one of the main multicast streams. VCR-like functionalities such as fast forward are not

[1] Of course, it is always possible for clients to cache the broadcasts and then enable local playback of the cached content with VCR-like functionality, as is done with many personal video recorders (e.g., TiVo, Replay TV) and set top boxes (e.g., Comcast) today.

generally available, although of course some of these can be partially recovered through client-side caching. These systems have been mainly of interest to the cable/satellite TV industry, who may be able to afford a dozen full-bandwidth multicast channels flowing simultaneously into a set top box to enable a single movie on demand. However, this approach is not generally possible when the bandwidth into a client is at a premium and the client wishes to make full use of that bandwidth for the highest possible quality. Hence we will not address this approach further in this chapter, but rather leave the subject to further reading.

14.2.2 Communication Protocols

Streaming media on demand requires a large number of communication protocols at different levels. At the topmost level are protocols for content discovery and connection to a specific streaming media server. Typically, content discovery is done "out of band," for example, by browsing a web page or receiving a link to the content in an email message. In either case the link would typically have a form such as

```
http://www.microsoft.com/directory/contentname.asx
http://www.realnetworks.com/directory/contentname.ram
http://www.apple.com/directory/contentname.mov
```

which are, respectively, representative of links to content from Microsoft Windows Media, RealNetworks RealMedia, and Apple QuickTime. These uniform resource locators (URLs) point, actually, to small auxiliary, metadata, or reference files typically located on a web server rather than to the media file itself. An `asx` file, for example, is actually an XML file describing the URL of the media server and the specific name of the media file on the server, including the protocol and potentially other parameters and content presentation instructions [17]. RealMedia uses a similar format, called `ram`. Figure 14.2 provides examples of such auxiliary files.[2] QuickTime accomplishes a similar task with a small reference movie file in the QuickTime `mov` format.

Once an auxiliary file is retrieved from a web server or other location, its file name extension and MIME type indicate that the default client software (e.g., media player) should be launched, and the auxiliary file is then read by the client application or embedded object or control (known hereafter simply as the client).

[2]The auxiliary file can provide minor scripting of the presentation, such as insertion of advertisements and player background color. However, if more complex scripting is needed, an alternative or additional level of indirection, the SMIL file, can be used. SMIL is a variant of HTML capable of describing fairly complex multimedia presentations involving, for example, fading from one piece of content into another at a particular time in the presentation, integrating with images and text, etc. Originally promulgated by RealNetworks, it was substantially reworked by a number of interested parties and is now a W3C recommendation [3].

```
<ASX Version="3.0">
<ENTRY>
<REF HREF="mms://streamingmedia/studios/0505/24721/MTV_XBOX_preview_160k.wmv" />
</ENTRY>
<ENTRY>
<REF HREF="mms://winmedianw/studios/0505/24721/MTV_XBOX_preview_160k.wmv" />
</ENTRY>
</ASX>
```

(a)

```
# First URL that opens a related info pane.
rtsp://helixserver.example.com/video3.rm?rpcontextheight=350
&rpcontextwidth=300&rpcontexturl="http://www.example.com/relatedinfo2.html"
&rpcontexttime=5.5&rpvideofillcolor=rgb(30,60,200)
#
# Second URL that keeps the same related info pane,
# but changes the media playback pane's background color.
rtsp://helixserver.example.com/video4.rm?rpcontexturl=_keep &rpvideofillcolor=red
```

(b)

FIGURE 14.2: Auxiliary files describing where to find streaming media content. (a) An ASX file, from Microsoft. (b) A RAM file, from RealNetworks.

At the appropriate time, the client contacts the server using the URL for the content, for example,

```
rtsp://wms.microsoft.com/directory/contentname.wmv
rtsp://helixserver.example.com/audio1.rm?start=55&end=1:25
rtsp://qtserver.apple.com/directory/contentname.mov
```

where the prefix indicates the streaming protocol used, and various optional suffixes can pass information to the server, such as seek point and play speed.

The "streaming protocol" is a high-level control protocol enabling the client to interactively control playback using VCR-like functions, including start, stop, pause, fast forward, and seek. These commands are typically communicated reliably over a TCP connection. Although various roughly equivalent proprietary protocols are used here, one protocol that is now widely adopted is the Real Time Streaming Protocol (RTSP), which is codified by the Internet Engineering Task Force (IETF) Request for Comments (RFC) 2326 [37]. RTSP is an HTTP-like protocol, including commands to play and stop. Sample commands are shown in Table 14.1. Although the commands appear very basic, extensions can be defined using the SET_PARAMETER command, for example, to allow dynamic selection of particular streams from the media file. As discussed in the next section, a media file generally offers several streams, not only an audio stream and a video stream, but potentially several audio and video streams, for example, for different languages, subtitles, source coding rates, etc. Some of these, such as languages

Table 14.1: RTSP protocol.

DESCRIBE	Retrieves description of presentation, usually in Session Description Protocol (SDP) format [19], together with all initialization information.
SETUP	Causes server to allocate resources for a stream; specifies transport protocol; starts an RTSP session.
SET_PARAMETER	Specifies stream bit rate, etc.
PLAY	Starts data transmission of a stream from a specified time point at a specified speed.
PAUSE	Temporarily halts stream without freeing server resources.
TEARDOWN	Frees resources associated with the stream; ends an RTSP session.

and subtitles, must ultimately be user-selectable, while others, such as source coding rate, may be automatically selectable by the server and/or client. The SET_PARAMETER command can be used to pass information from the client to the server to enable dynamic stream selection.

The "streaming protocol" also enables the client to specify which lower level data transport protocol to use. The data transport protocol is usually either RTP over UDP or RTP over TCP (the only two transport protocols that are standardized with RTSP) or HTTP over TCP (which can be specified with nonstandardized streaming protocols). Systems such as Windows Media and Helix implement multiple transport protocols and use whichever protocol is most appropriate for a given situation. For example, HTTP over TCP may be used when there are firewall issues to be avoided. RTP over UDP or RTP over TCP is usually preferred for bandwidth efficiency. However, when RTP over UDP is used, there must be a proprietary means of transmission rate control (i.e., congestion control) and error control (i.e., packet loss recovery). To date, there is no standard means of transmission rate control and error control for RTP (Real Time Transport Protocol, IETF RFC 3550 [36]). RTP is essentially only a packet format that adds a timestamp, a sequence number, a contributing source identifier, and a payload type and format on top of an ordinary UDP packet. As with UDP, the application is left to perform transmission rate control and error control. RTCP (Real Time Control Protocol, IETF RFC 3551 [35]) is often paired with RTP, but provides only a format with which receivers may provide statistical feedback to the sender. There is no standard protocol by which receivers may provide timely feedback to the sender. This makes RTP+RTCP over UDP inherently *not interoperable*, proprietary, and hence (in the author's opinion) of limited use as a standard. However, the IETF is currently at work trying to change this.

The Windows Media system uses a particular form of transmission rate control and error control for RTP over UDP. Transmission rate control is currently based on constant bit rate transmission from the server, at the source coding rate of the content. By monitoring its buffer, the client can determine whether congestion is occurring and, if so, can signal to the server to change to a stream with a lower source coding rate. Roughly, if the client buffer duration drops below an adaptive theshold or if the packet loss rate rises above an adaptive threshold, then congestion is detected and the stream transmission or source coding rate is switched down. The extended absence of congestion allows the stream transmission/source coding rate to be switched up.

An alternative method of transmission rate control would be to use, for example, either a TCP-friendly rate control (TFRC) algorithm [13,18] or a TCP-like congestion control algorithm (i.e., window-based additive increase and multiplicative decrease without retransmission or in-order delivery). Both TFRC and a TCP-like congestion control are being standardized as two profiles in the umbrella Datagram Congestion Control Protocol (DCCP) [14,15,25]. However, such a transmission rate control protocol must be paired with a source coding rate control protocol, since the source coding rate of the content must also, at least over the long run, rise and fall with the transmission rate and yet it may not be possible or desirable to make the source coding rate always equal to the transmission rate. A potential source coding rate control algorithm that can be paired with an arbitrary transmission rate protocol is the rate-distortion optimized (RaDiO) scheduling algorithm described in Chapter 10. Section 14.4 presents an alternative method based on optimal control theory.

Error control in Windows Media is currently based on selective retransmission. If the client detects gaps in the packet sequence numbers, it sends a NAK to the server, which retransmits the missing packet(s). The number of packets requested for retransmission is limited to a percentage of the overall bandwidth. Audio packets are given highest priority, while video packets closest to their playout deadlines are given lowest priority, on the presumption that if the client is scrambling to recover packets in a bandwidth-limited situation, then these packets are the most likely to miss their playback deadlines even if they are retransmitted. A more precise way of prioritizing retransmissions, of course, is using the rate-distortion optimized scheduling algorithm, as described in Chapter 10. Rate-distortion optimized scheduling will optimize user quality under deadline pressure. Regardless of the scheduling and retransmission algorithm, some packets may remain missing at their playback deadlines. Hence in such cases the client must ultimately decide whether to stall playback until the packets can be recovered or conceal the missing packets. The Windows Media player chooses to stall until all audio packets can be recovered, while skipping lost video packets. Lost video packets can be dealt with by some form of error concealment, as discussed in Chapter 2. A rudimentary form of error concealment is to freeze until the next I frame.

14.2.3 File Formats

The greatest challenge for streaming media on demand is adapting the content of a fixed media file to various network and client conditions. Unlike so-called *real-time communication* (RTC), such as telephony, conferencing, and online gaming (covered in Chapter 15), in streaming media on demand the encoder must encode its content off line, possibly years before it is actually streamed. Thus it cannot have direct knowledge of the communication channel selected (including the capacity and loss rate of the network path and the capacity of the client buffer) nor can it have direct knowledge of the instantaneous communication state (including the level of network congestion or interference and the state of the client buffer). Instead, the encoder must build a certain degree of flexibility into the media file and leave it to the server to adapt the media file to the network and client conditions.

As an extreme case, the media file may be simply a raw, uncompressed recording of the content, and the server may spawn an online encoder to adaptively compress the content for each connected client. However, this is usually infeasible, both because of the high storage requirements for uncompressed media and the high computational requirements for real-time encoding for multiple simultaneous clients. As a somewhat less extreme case, the media file may be a high bit rate encoding of the content, and the server may spawn an online transcoder, or *transrater*, to adaptively recompress the content for each connected client. At the opposite end of the spectrum, the media file may contain an immutable encoding of the content, which the server simply copies onto the network connection (such as in the progressive download scenario). Usually, however, a better engineering trade-off involves making the media file format flexible enough for the server to simply index into the file and extract the content that it wants for specific users. For this, the format of the file, and possibly the format of its encoded bit streams, must be thoughtfully designed.

There are a number of streaming file formats available, from international standards such as the MPEG-4 file format [38] to *de facto* industry standards such as Apple's QuickTime format [10] (on which the MPEG-4 file format is based), RealNetworks' RealMedia format [33], and Microsoft's Advanced Streaming Format (ASF) [11]. All files in these formats have some common characteristics. First, they are able to contain, or multiplex, not only multiple media, but also multiple versions of each medium. Each version is recorded in a *track* (in MPEG-4/QuickTime parlance) or a *stream* (in ASF parlance). Each track or stream is decomposed into a sequence of *chunks* (in MPEG-4/QuickTime parlance) or packets (in ASF parlance), which contain actual encoded media data. In this chapter we will call these *data units*. In each file, header structures contain static metadata relating to the overall file as well as to specific tracks or streams. These metadata may include, for example, title, author, and date of composition, encryption

and rights management information, table of contents, track/stream enumeration, and descriptions of their relationships, as well as individual track/stream properties such as start time, duration, bit rate, buffer size, sampling rate, picture size, and scalability capabilities. Time-varying metadata are associated with each track/stream. These metadata may include, for example, network packetization information, decoding and presentation time stamps, SMPTE time codes, key frame, switch frame, or other clean point information, and fine grain scalability information, such as a set of cut points and associated distortion or RD-slope information for each chunk or data unit. How these time-varying metadata are associated with individual chunks or data units, however, reveals one of the few philosophical differences between formats. MPEG-4/QuickTime uses separate tracks, called *hint tracks*, to add time-varying metadata to a track, whereas ASF uses extensible side structures (which may be called *hints*) associated with each data unit in a stream. A final type of metadata common to all formats is an index to allow seeking to particular time locations in each track/stream. One way to create such an index so that it can be efficiently searched (given, e.g., a SMPTE time code) is to arrange the index as a sequence of fixed-length records, each record corresponding to a unit of time (say, a 1-s interval), and each record containing a pointer into each track or stream. In all these formats, the structures are, in principle, highly extensible, by assigning new 32-bit (four-character) codes ("fourCC"s), by assigning new 128-bit globally unique identifiers (GUIDs), or by defining new structures in areas for opaque data. In practice, however, such extensions are limited in utility by the availability of servers and clients that understand the extensions.

Some of the metadata in a streaming media file is intended for consumption by the server only, while other metadata must be transmitted across the network for consumption by the client. For example, the hint and index information is usually used only at the server and is not transmitted across the network. If the end user wishes to seek to a particular time in a presentation, then the client will send a seek command and time to the server (see the previous section, Section 14.2.2), which will then use the seek time to look up in the index an appropriate starting offset within each stream[3] (e.g., at the last key frame before the seek time), and then will begin streaming from that point.

Other metadata, such as the descriptions and relationships of the streams, are usually transmitted over the network to the client so that the end user (and/or the client application on the user's behalf) can choose an appropriate set of streams for transmission, whether the selection has to do with the content (e.g., language of the audio track, optional subtitles) or the encoding of the content (e.g., bit rate, picture size, number of audio channels). As a rule of thumb, static metadata, whose size is independent of the length of the data, are relatively inexpensive to

[3]Henceforth for simplicity we will use only the *stream* terminology.

transmit over the network, whereas time-varying metadata, whose size grows with the length of the data, are relatively expensive to transmit over the network.

The encoded media data in the data units, of course, are generally intended to be transmitted over the network to the client. However, in a given session, usually only a fraction of all the encoded media data in a file is actually transmitted. Indeed, perhaps the main purpose of a streaming media file format is to provide a structure in which metadata can be used to easily select an appropriate subset of the data for transmission. Selection of an appropriate subset may be either *coarse grained* or *fine grained*. In coarse-grained selection, data units are selected on a per-stream basis. Thus, at any given time, the server is streaming only a particular subset of streams to the client. The subset of selected streams may change over time, but in coarse-grained selection this subset changes only on a timescale that is long compared to a data unit. In fine-grained selection, not only may a subset of streams be selected, but also some fraction of data within a stream may be selected for transmission. This fine-grained selection may be either at the data unit level (in which some data units in a stream are transmitted, while others are not) or below the data unit level, in which a portion of each data unit may be selected for transmission.

Multibit rate (MBR) coding and scalable coding are two means of encoding media data into a streaming media file that allow computationally simple selection of data for transmission based on the source coding rate. In MBR coding, multiple independent encodings of the same media content, each at a different source coding rate, are stored in different streams in the same file. Adaptation is based on stream selection. For example, as discussed in Section 14.2.2, if the client detects congestion based on buffer fullness, it can ask the server to switch to a stream with a lower source coding rate. In scalable coding, as described in detail in Chapters 4–6, encoded bit streams at different source coding rates are embedded, like layers of an onion. In coarse-grained scalability, if the client detects congestion, it can ask the server simply to drop the stream corresponding to the currently uppermost layer for a medium. In fine-grained scalability (FGS), in contrast, the client may instead send a parameter (such as a Lagrange multiplier) related to the desired source coding rate for a medium, with which the server may adjust the source coding rate for the medium using fine-grained selection of the data in the medium's streams. This would typically be done using a simple threshold. For example, if each data unit is tagged with metadata that sets a collection of breakpoints in the data unit along with a Lagrange multiplier corresponding to each breakpoint (in a decreasing sequence), then the server could use the Lagrange multiplier given to it by the client to threshold each data unit to determine where to truncate the data unit. Those segments of the data unit with Lagrange multipliers higher than the Lagrange multiplier specified by the client are transmitted, while those segments of the data unit with Lagrange multipliers below are not transmitted.

The data units (or portions of data units) selected for transmission are packetized into network data units, or *packets*, for transmission over the network. In the parlance of ITU H.264/ISO MPEG-4 AVC, the file format is a Network Adaptation Layer (NAL), which is an encapsulation layer below the encoded bit stream layer at the same level as, say, RTP and other network transport protocols. Thus packetization can be regarded as re-encapsulations of the encoded media data from one transport protocol (the file format) into another (e.g., RTP).

It can now be appreciated that media file formats designed for local playback and storage are not suitable for streaming, in general. Even QuickTime, which was originally designed for local playback of media but which is highly flexible, adds metadata in "hint tracks" for the purposes of streaming and restricts its many levels of indirection and indexing both to allow the file to be used for progressive downloading and to allow the server maximal access efficiency. Furthermore, even "streaming" formats, such as the MPEG-2 transport stream, which was designed for streaming data over isochronous cable, terrestrial radio, and satellite channels, is not very suitable as a file format for adaptive streaming over packet networks, since it is not very easy for a server to adaptively extract selected portions of the stream, nor does it have an indexing mechanism to allow a user to randomly access (i.e., seek to) arbitrary points in the stream. Thus streaming media file formats must be carefully designed to fit their purposes.

14.3 FUNDAMENTAL ABSTRACTIONS

In this section, we lay out the fundamental abstractions of streaming media on demand. In particular, we cover leaky bucket models of the bit streams, constant bit rate (CBR) vs. variable bit rate (VBR) streams, compound (multiple media) streams, preroll delay, playback speed, timing, clocks, and decoder and presentation timestamps. At the end of this section, the reader will know, for example, for streaming multiple media over an ideal (isochronous) network, when it is safe for the client to begin playback after streaming begins, at any playback speed.

14.3.1 Buffering and Leaky Bucket Models

We first consider the constant bit rate scenario in which both the encoder and the decoder communicate over a dedicated isochronous noiseless communication channel.[4] In this scenario, to match the instantaneous coding rate of the source to the constant transmission rate of the channel, an *encoder buffer* is required between the encoder and the channel and a *decoder buffer* is required between the

[4] An isochronous channel is one, such as a telephone modem, in which equal amounts of data are communicated in equal amounts of time (from the Greek *iso*, "same," plus *chronos*, "time"). Information flows through an isochronous channel as fluid through a pipe.

channel and the decoder, as illustrated in Figure 14.3. A *schedule* is the sequence of times at which successive bits in the coded bit stream pass a given point in the communication pipeline. Figure 14.4 illustrates the schedules of bits passing the points A, B, C, and D in Figure 14.3. Schedule A is the schedule at which captured frames are instantaneously encoded and put into the encoder buffer. This schedule is a staircase in which the nth step rises by $b(n)$ bits at time $\tau(n)$, where $\tau(n)$ is the time at which frame n is encoded, and $b(n)$ is the number of bits in the resulting encoding. Schedules B and C are the schedules at which bits, respectively, enter and leave the channel. The slope of these schedules is R bits per second, where R is the transmission rate of the channel. Schedule D is the schedule at which frames are removed from the decoder buffer and instantaneously decoded for presentation. Note that Schedule D is a right shift of Schedule A, assuming constant end-to-end delay and to zero encoding and decoding delay, including algorithmic delay. (These assumptions will be relaxed in Section 14.4.) Note also that Schedule B is a lower bound to Schedule A, while Schedule C is an upper bound to Schedule D. Indeed the gap between Schedules A and B represents, at any point in time, the fullness in bits of the encoder buffer, while the gap between Schedules C and D likewise represents the fullness of the decoder buffer. Thus it is clear that the source coding schedule (either A or D) can be contained within a *buffer tube*, as illustrated in Figure 14.5, having slope R bits per second, some height B bits, and some initial offset F_e bits from the bottom of the tube or, equivalently, some initial offset $F_d = B - F_e$ bits from the top of the tube. The buffer tube characterizes with three parameters (R, B, F_e) or, equivalently, (R, B, F_d)—the variability of the instantaneous rate of the source coding sched-

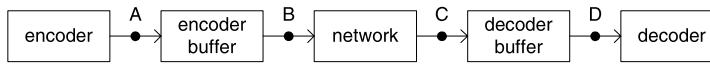


FIGURE 14.3: Communication pipeline.

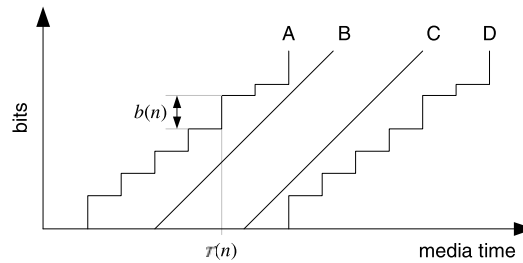


FIGURE 14.4: Schedules at which bits in the coded bit stream pass the points A, B, C, and D in the communication pipeline.

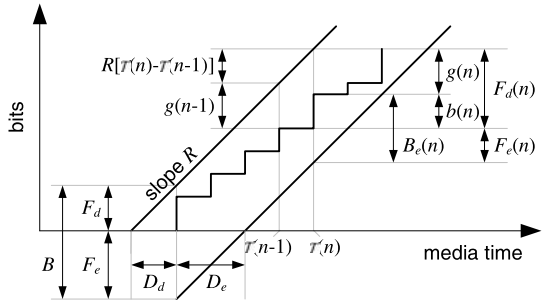


FIGURE 14.5: Buffer tube containing a coding schedule.

ule around an average rate R. In some sense, the buffer tube paints with a broad brush stroke the source coding schedule (a possibly infinite sequence of step sizes and times $(b(n), \tau(n))$, $n = 0, 1, 2, \ldots$) as a straight line with slope R, thickness B, and offset F_e.

Traditionally, the buffer tube is used by an online source encoder to ensure that its output will not cause the decoder buffer to underflow or overflow. It is clear that at any given frame, the fullness of the encoder buffer and the fullness of the decoder buffer are complementary, adding up to B bits, after an appropriate delay. Thus, if the encoder and decoder buffers each have capacity B bits, an overflow of the encoder buffer is equivalent to an underflow of the decoder buffer, and vice versa. The online source encoder traditionally uses a "rate control" algorithm to assign a number of bits $b(n)$ to each frame n to ensure that its B-bit buffer neither underflows nor overflows, when beginning with initial fullness F_e bits. (Beginning with initial fullness F_e bits means simply that the first bit inserted into the buffer by the encoder will be delayed $D_e = F_e/R$ seconds before it enters the channel.) Thus any B-bit decoder buffer will not overflow or underflow if it begins with initial fullness $F_d = B - F_e$. (Beginning with initial fullness F_d bits means simply that the first bit entering the buffer from the channel will be delayed $D_d = F_d/R$ seconds before it is extracted by the decoder.) The decoder buffer delay $D_d = F_d/R$ is thus complementary to the encoder buffer delay, with overall buffer delay equal to B/R seconds. The end-to-end delay is thus B/R plus any transmission delay. The resulting bit stream is called a CBR bit stream with average rate R even though, unlike the rate at which bits enter the channel, the rate that the online source encoder produces bits is obviously not constant over timescales shorter than one frame period.

For off-line source encoding, such as for streaming media on demand, the encoder often produces a CBR bit stream in exactly the same way—assuming a decoder buffer with size B. However, unlike the online case in which it is important to keep the overall buffer delay B/R low, in the off-line case it is important

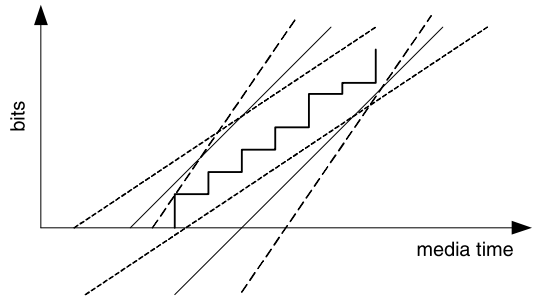


FIGURE 14.6: Multiple buffer tubes containing a given bit stream.

only to keep the decoder buffer delay $D_d = F_d/R$ low. The encoder buffer delay portion of the overall buffer delay is not important. The *decoder buffer delay*, also loosely known as the *preroll delay*, is usually the largest component in the *start-up delay*, or the time between a user pushing "play" and seeing or hearing the first frame (during which media players usually display a "Buffering ..." message).[5] The encoder can produce a CBR bit stream minimizing the decoder buffer delay by starting the encoder buffer in a full state (F_e close to B), although this would mean that the initial frame would get very few bits. This is a reflection of a fundamental trade-off, at any given transmission rate, between preroll delay and initial quality, which we will explore further in Section 14.4.

It is also possible for off-line source encoders to ignore all buffer constraints. A typical such scenario is the scenario where the encoder tries to produce a constant distortion bit stream (e.g., with equal quantization stepsizes for all frames). The resulting bit stream is usually called a variable bit rate stream. Ultimately, there is no firm distinction between VBR and CBR streams, although they may be produced in different ways. The distinction, if any, is one of degree. Both CBR and VBR streams have schedules that can be represented by finite sequences of step sizes and times, $\{(b(n), \tau(n))\}_{n=0}^{N}$. Thus, both can be contained in buffer tubes with some appropriate slope, width, and offset. However, VBR streams tend to require wider buffer tubes than CBR streams. This usually means that VBR streams require a larger start-up delay.

It is clear that any given finite-duration bit stream is containable by an infinite number of buffer tubes, as illustrated in Figure 14.6. The slope R, the width B, and the offset F_e are not unique. This implies, in particular, that the average bit rate of a stream, whether VBR or CBR, is not well defined by the bit rate of a

[5]The start-up delay includes, in addition to the decoder buffer delay, usually much smaller delays such as the round trip delay between client and server, server processing delays, and decoding and presentation delays, if any.

constant bit rate channel over which the stream may flow with sufficient buffering. Of course, there are ways to define the average bit rate of a stream uniquely. One possible definition is the slope of parallel lines bounding the stream's schedule $\{(b(n), \tau(n))\}_{n=0}^{N}$, clamped together by a vice as tightly as possible (so to speak), to make the buffer tube unique. Another possible definition is the total number of bits in the stream divided by the duration of the stream. These are approximately equal if the stream is long enough. Furthermore they will be approximately equal to the slope R of any buffer tube (R, B, F_e) containing the stream as long as the duration of the stream is long compared to the buffer size B. Thus any reasonable definition of average bit rate will suffice as long as we are given a corresponding buffer tube.

We now extend the original online scenario to the case in which the encoder does not use the channel continuously. In this case, the channel has a peak transmission rate R higher than the average bit rate of the stream. When the encoder has bits to send, it sends them at rate R. Otherwise it does not use the channel and the channel may be used to transmit other information during this time. This is a realistic setting for shared channels such as packet networks. As far as the encoder is concerned, the channel time shares between transmitting at rate R and transmitting at rate 0, such that the time average of these two instantaneous channel rates is approximately the average bit rate of the stream.

In this scenario, the encoder buffer is best modeled by a *leaky bucket*. The encoder dumps $b(n)$ bits into the leaky bucket at time $\tau(n)$, and the bits leak out of the bucket (into the channel) at rate R. When the leak rate R is higher than the average bit rate, the bucket will occasionally become empty. Thus the encoder buffer fullness $F_e(n)$ immediately before frame n is added to the bucket and the encoder buffer fullness $B_e(n)$ immediately after frame n is added to the bucket evolve from an initial encoder buffer fullness $F_e(0) = F_e$ according to the dynamical system

$$B_e(n) = F_e(n) + b(n), \tag{14.1}$$

$$F_e(n+1) = \max\bigl\{0, B_e(n) - R\bigl[\tau(n+1) - \tau(n)\bigr]\bigr\}, \tag{14.2}$$

for $n = 0, 1, 2, \ldots$. As described earlier, a leaky bucket can be specified by three parameters: its leak rate R, its capacity B, and its initial fullness F_e. A leaky bucket (R, B, F_e) is said to *contain* a bit stream with schedule $\{(b(n), \tau(n))\}_{n=0}^{N}$ if the bucket does not overflow, that is, $B_e(n) \le B$ for all $n = 0, 1, \ldots, N$ in the dynamical system (14.1) and (14.2).

It is of interest to find among all leaky buckets containing a stream the one that leads to the smallest decoder buffer size and also the one that leads to the smallest decoder buffer delay. For a given stream, define the minimum bucket capacity

given leak rate R and initial fullness F_e as

$$B^{\min}(R, F_e) = \min_n B_e(n), \tag{14.3}$$

and define the corresponding initial decoder buffer fullness as

$$F_d^{\min}(R, F_e) = B^{\min}(R, F_e) - F_e. \tag{14.4}$$

Denote the minimum of each of these over F_e as

$$B^{\min}(R) = \min_{F_e} B^{\min}(R, F_e), \tag{14.5}$$

$$F_d^{\min}(R) = \min_{F_e} F_d^{\min}(R, F_e). \tag{14.6}$$

It is shown in [34, Proposition 2] that remarkably, these are each minimized by the same value of F_e, which is hence equal to

$$F_e^{\min}(R) \triangleq B^{\min}(R) - F_d^{\min}(R). \tag{14.7}$$

Thus given a bit stream with schedule $\{(b(n), \tau(n))\}_{n=0}^{N}$, for each bit rate R there is a *tightest leaky bucket* containing the stream that has the minimum buffer capacity B as well as the minimum decoder buffer delay $D_d = F_d/R$, provided it begins with initial fullness $F_e = F_e^{\min}(R)$.

These formulae work for any $R > 0$, even for R less than the average bit rate of the stream. However, when R is less than the average bit rate of the stream, the leaky bucket accumulates bits faster than they leak out, causing the required bucket capacity to grow linearly as R goes to zero, up to the size of the entire stream. In a like manner, the preroll delay that is required to ensure that the decoder buffer does not underflow during playback grows linearly up to the playback time of the entire stream as R goes to zero. However, when R is above the average bit rate of the stream, bits leak out of the bucket faster than they are put in, causing the bucket to become empty on occasion. This provides another possible definition of average bit rate of a stream: the average bit rate of a stream, henceforth called the *source coding rate* R_c of a stream, is the maximum leak rate R such that a leaky bucket (R, B, F_e) containing the stream does not underflow when starting with initial fullness $F_e = F_e^{\min}(R)$.

It is intuitively clear from the leaky bucket metaphor that the larger the leak rate R, the smaller the required capacity. It is also true that the decoder buffer delay can be lower. Indeed, it can be shown using results from Ribas-Corbera *et al.* [34, using Lemmas 3, 5, 6, 11, 12] that both $B^{\min}(R)$ and $F_d^{\min}(R)$ are decreasing, piecewise linear, convex functions of R, as shown in Figure 14.7. Hence if the

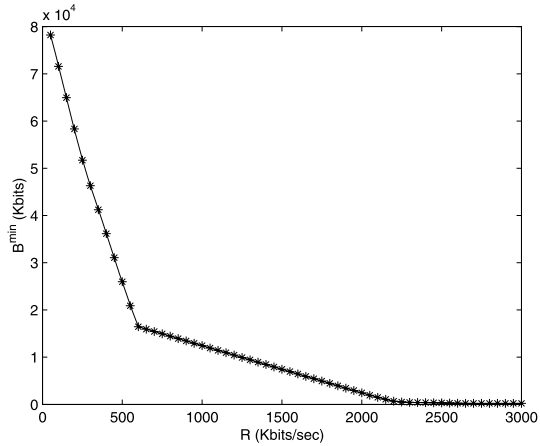


FIGURE 14.7: Plot of $B^{\min}(R)$ for a VBR video clip compressed by H.264/AVC with QP=26.

transmission rate R is greater than the source coding rate R_c, the decoder buffer size $B = B^{\min}(R)$ can be reduced compared to $B = B^{\min}(R_c)$ and the decoder buffer delay $D_d = F_d^{\min}(R)/R$ can be reduced compared to $D_d = F_d^{\min}(R_c)/R_c$. Figure 14.7 shows, for a typical video bit stream with average source coding rate $R_c \approx 600$ Kbps, the minimum buffer size $B^{\min}(R)$ as a function of the peak channel transmission rate R. Observe that the function is piecewise constant and can be characterized by a small number of points.

In the off-line scenario, the client is generally connected to the server across a channel whose transmission rate is different than the source coding rate R_c of the bit stream. The client can determine the required buffer size and preroll delay from the functions $B^{\min}(R)$ and $F_d^{\min}(R)$. These functions can be computed off-line at a selected set of channel transmission rates R, say $R_1 < R_2 < \cdots < R_L$ and stored in the bit stream header as a set of leaky bucket parameters (R_i, B_i, F_i), $i = 1, 2, \ldots, L$, where $B_i = B^{\min}(R_i)$ and $F_i = F_d^{\min}(R_i)$, which can be communicated to the client upon initialization. Typically, the first such leaky bucket can be a buffer tube representing the source coding rate, that is, $R_1 = R_c$, and the remaining leaky buckets can be located at other significant breakpoints in the piecewise linear functions $B^{\min}(R)$ and $F_d^{\min}(R)$. Then, given any channel transmission rate R between R_i and R_{i+1}, the client can estimate $B^{\min}(R)$ and $F_d^{\min}(R)$ by the linear interpolations

$$\hat{B}^{\min}(R) \triangleq \frac{R_{i+1} - R}{R_{i+1} - R_i} B_i + \frac{R - R_i}{R_{i+1} - R_i} B_{i+1} \geq B^{\min}(R), \qquad (14.8)$$

$$\hat{F}_d^{\min}(R) \triangleq \frac{R_{i+1} - R}{R_{i+1} - R_i} F_i + \frac{R - R_i}{R_{i+1} - R_i} F_{i+1} \geq F_d^{\min}(R). \tag{14.9}$$

The inequalities follow from the convexity of the functions $B_e^{\min}(R)$ and $F_d^{\min}(R)$ in R, and they guarantee that a buffer size B and a preroll delay F_d/R are sufficient to ensure glitch-free playback when the channel has peak transmission rate R and the client can use flow control to ensure that the buffer does not overflow. Extrapolation at the low and high ends is also possible. For $R < R_1$,

$$\hat{B}^{\min}(R) \triangleq B_1 + (R_1 - R)T \geq B^{\min}(R), \tag{14.10}$$

$$\hat{F}_d^{\min}(R) \triangleq F_1 + (R_1 - R)T \geq F_d^{\min}(R), \tag{14.11}$$

where T is the duration of the bit stream, while for $R > R_L$,

$$\hat{B}^{\min}(R) \triangleq B_L \geq B^{\min}(R), \tag{14.12}$$

$$\hat{F}_d^{\min}(R) \triangleq F_L \geq F_d^{\min}(R). \tag{14.13}$$

Thus, a small set of leaky buckets $\{(R_i, B_i, F_i)\}_{i=1}^{L}$ stored in the bit stream header can be used to derive a fairly tight leaky bucket (R, B, F) for any channel transmission rate $R \geq 0$.

14.3.2 Compound Streams

A streaming media file often contains multiple independently encoded media streams, such as an audio stream, a video stream, and perhaps other streams, intended to be streamed concurrently.

When multiple media streams are selected for concurrent transmission, it is convenient to consider them as a single *compound stream* having an aggregate source coding rate and set of leaky buckets. Happily, a leaky bucket (B, F, R) for a compound stream can be easily derived as the sum of leaky buckets for its component streams. For example, if (R^a, B^a, F^a) and (R^v, B^v, F^v) are leaky buckets containing the component (say audio and video streams) streams, then the parameters

$$R = R^a + R^v, \tag{14.14}$$

$$B = B^a + B^v, \tag{14.15}$$

$$F = F^a + F^v \tag{14.16}$$

characterize a leaky bucket containing the compound stream. This is because, as is intuitively clear from the leaky bucket metaphor, if the separate leaky buckets contain the component streams without overflowing, then the combined leaky bucket

will contain the combination of the streams without overflowing. (However, the combined bucket is, in general, not the tightest leaky bucket that is able to contain the compound stream.) It is simple to show this mathematically, although we will not do so here.

If the component streams each have multiple leaky buckets, then any combination of their leaky buckets will suffice to contain the compound stream. For example, if $\{(R_i^a, B_i^a, F_i^a)\}_{i=1}^{L^a}$ and $\{(R_j^v, B_j^v, F_j^v)\}_{j=1}^{L^v}$ are, respectively, sets of leaky buckets containing an audio stream and a video stream, then $\{(R_{i,j}, B_{i,j}, F_{i,j}) : (i, j) \in [1, \ldots, L^a] \times [1, \ldots, L^v]\}$ is a set of leaky buckets containing the compound audio and video stream, where $R_{i,j} = R_i^a + R_j^v$, $B_{i,j} = B_i^a + B_j^v$, and $F_{i,j} = F_i^a + F_j^v$. However, it turns out that not all $L^a \times L^v$ leaky buckets in this set are useful for characterizing the compound stream. In fact, at most $L^a + L^v$ index pairs (i, j) have the property that $(R_{i,j}, B_{i,j})$ lies on the lower convex hull of the set $\{(R_{i,j}, B_{i,j}) : (i, j) \in [1, \ldots, L^a] \times [1, \ldots, L^v]\}$ used to estimate $B_e^{\min}(R)$ and $F_d^{\min}(R)$. Indices for pairs on the lower convex hull can be found easily by minimizing a Lagrangian for some positive Lagrange multiplier $\lambda > 0$, namely

$$(i_\lambda, j_\lambda) = \arg\min_{(i,j)} \{B_{i,j} + \lambda R_{i,j}\} \tag{14.17}$$

$$= \arg\min_{(i,j)} \left\{B_i^a + B_j^v + \lambda\left[R_i^a + R_j^v\right]\right\} \tag{14.18}$$

$$= \left(\arg\min_i \left\{B_i^a + \lambda R_i^a\right\}, \arg\min_j \left\{B_j^v + \lambda R_j^v\right\}\right). \tag{14.19}$$

Thus, as λ is swept from 0 to ∞, a sequence of L^a audio leaky buckets indexed by i_λ can be chosen by minimizing the Lagrangian $B_i^a + \lambda R_i^a$, and *independently* a sequence of L^v video leaky buckets indexed by j_λ can be chosen by minimizing the Lagrangian $B_j^v + \lambda R_j^v$. These can be paired by matching their Lagrange multipliers λ to find the (at most) $L^a + L^v$ leaky buckets for the compound stream.

This approach can be easily extended to more media than just audio and video. For example, suppose there are M concurrent media streams in a streamed video game and $m = 1, 2, \ldots, M$ indexes the media, and suppose that for each medium m, there is a set of leaky buckets indexed by $i^m = 1, \ldots, L^m$. Then following the aforementioned arguments it is easy to see that for each medium m, one can select for each $\lambda > 0$ a leaky bucket index $i_\lambda^m = \arg\min_i \{B_i^m + \lambda R_i^m\}$, where (R_i^m, B_i^m, F_i^m) is the ith leaky bucket for medium m. These can then be aligned by λ to choose the component leaky buckets for the (at most) $\sum_m L^m$ leaky buckets for the compound stream. Even further simplifications accrue when $L^m = 2$ for all m, for example, when there are leaky buckets (R_1^m, B_1^m, F_1^m) and (R_2^m, B_2^m, F_2^m) for only average and peak bit rates for each component stream. In that case, as λ goes from 0 to ∞, for each m there is a simple threshold, namely $\lambda^m = [B_1^m - B_2^m]/[R_2^m - R_1^m]$, such that when $\lambda \geq \lambda^m$ we have $i_\lambda^m = 1$ (choose the LB for

average bit rate) and when $\lambda < \lambda^m$ we have $i_\lambda^m = 2$ (choose the LB for peak bit rate). Thus a set of $M+1$ leaky buckets $\{(R_k, B_k, F_k)\}_{k=0}^M$ for the compound stream can be obtained by sorting the media on λ^m and successively flipping the chosen component leaky buckets from average to peak bit rate, namely $R_k = \sum_{m=1}^k R_1^m + \sum_{m=k+1}^M R_2^m$ (and similarly for B_k and F_k).

14.3.3 MBR and Scalable Streams

In MBR and scalable streaming, in addition to the possibility of streaming multiple *concurrent* media streams such as audio and video, each concurrent media stream is generally selected from a list of *mutually exclusive* media streams, each encoded at a different source coding rate. Combining all possible mutually exclusive audio streams with all possible mutually exclusive video streams can lead to a large number of compound streams, each having a different aggregate source coding rate and set of leaky buckets. In principle, each of the (say) N^a mutually exclusive audio streams can be matched with each of the N^v mutually exclusive video streams, producing all possible $N^a \times N^v$ combinations. However, most of these combinations are not desirable. In fact, typically there are only on the order of $N^a + N^v$ desirable combinations. For example, if audio quality is more important than video quality, then during network congestion it may be desirable to reduce video quality through N^v levels before reducing audio quality through an additional N^a level. However, it may instead be desirable to reduce the audio and video bit rates together. A principled way to decide which of the $N^a \times N^v$ combinations are desirable is to take a distortion–rate approach such as the following. Assign a distortion D_i^a and a source coding rate R_i^a to each audio stream $i = 0, 1, \ldots, N^a$ (which includes the empty stream $i = 0$) and a corresponding distortion D_j^v and source coding rate R_j^v to each video stream $j = 0, 1, \ldots, N^v$. Define for each combined stream (i, j) an overall distortion and an overall source coding rate,

$$D_{i,j} = \alpha D_i^a + D_j^v \tag{14.20}$$

$$R_{i,j} = R_i^a + R_j^v, \tag{14.21}$$

allowing the audio distortion to be arbitrarily weighted by a parameter α relative to the video distortion. Select a "desirable" subset of the audio/video substream combinations (i, j) such that for each (i, j) in the subset, $D_{i,j} \le D_{i',j'}$ for all (i', j') such that $R_{i',j'} \le R_{i,j}$. That is, desirable combinations have the property that they have the lowest total distortion among all combinations with the same or lower total bit rate. One such desirable subset consists of the combinations (i, j) whose rate–distortion pairs $(R_{i,j}, D_{i,j})$ lie on the lower convex hull of the set of rate–distortion pairs for all possible combinations. Similar to the methodology described in the previous subsection, pairs on this lower convex hull can be easily

found by minimizing a Lagrangian for some positive Lagrange multiplier $\lambda > 0$. That is,

$$(i_\lambda, j_\lambda) = \arg\min_{(i,j)}\{D_{i,j} + \lambda R_{i,j}\}, \tag{14.22}$$

$$= \arg\min_{(i,j)}\left\{\alpha D_i^a + D_j^v + \lambda\left[R_i^a + R_j^v\right]\right\}, \tag{14.23}$$

$$= \left(\arg\min_i\left\{D_i^a + (\lambda/\alpha)R_i^a\right\}, \arg\min_j\left\{D_j^v + \lambda R_j^v\right\}\right). \tag{14.24}$$

Thus, as λ is swept from 0 to ∞, a sequence of $N^a + 1$ audio streams i_λ (including the empty substream $i = 0$) can be chosen by minimizing the Lagrangian $D_i^a + (\lambda/\alpha)R_i^a$, and (independently) a sequence of $N^v + 1$ video streams j_λ can be chosen by minimizing the Lagrangian $D_j^v + \lambda R_j^v$. These can be paired by matching their Lagrange multipliers λ. Note that it is a simple matter to repair them if the relative audio weight α changes, possibly under user control.

Also similar to the methodology in the previous section, this approach can be easily extended to more media than just audio and video. For example, suppose there are M concurrent media streams in a streamed video game and $m = 1, 2, \ldots, M$ indexes the media, and suppose that for each medium m, there is a set of mutually exclusive streams $i^m = 0, 1, \ldots, N^m$ (including the empty stream $i^m = 0$), one of which can be combined with streams from other media to form a compound stream. It is easy to see that for each medium m, one can select for each $\lambda > 0$ a substream $i_\lambda^m = \arg\min_i\{D_i^m + \lambda R_i^m\}$, where (R_i^m, D_i^m) is the rate–distortion pair for the ith stream of medium m. These can then be aligned by λ to choose the components of the "desirable" compound streams, a process that is linear in M instead of exponential in M. Even further simplifications accrue when $N^m = 1$ for all m. In that case, as λ goes from 0 to ∞, for each m there is a simple threshold, namely $\lambda^m = [D_0^m - D_1^m]/R_1^m$, such that when $\lambda \le \lambda^m$ we have $i_\lambda^m = 1$ (i.e., medium m is included in the compound stream) and when $\lambda > \lambda_m$ we have $i_\lambda^m = 0$ (i.e., medium m is not included in the compound stream). Thus the set of desirable compound streams can be obtained by sorting the media elements on λ^m and including them, in order, into the compound streams.

14.3.4 Temporal Coordinate Systems and Timestamps

Timestamps are generally associated with each encoded frame to instruct the client when to extract the frame from the decoder buffer and (instantaneously) decode it. These timestamps are known as *decoder timestamps* (DTS) in MPEG terminology and are the primary timestamps that we consider here. They can also be considered *decoding deadlines* by which the frame must arrive at the client in order to be decoded on time. It is fair for the client to decode received frames

ahead of their decoding deadlines, if there is sufficient room in the presentation buffer between the decoder and the renderer. The presentation buffer holds decoded frames. Decoded frames are also associated with timestamps, known as *presentation timestamps* (PTS) in MPEG terminology. These timestamps instruct the renderer when to render the frame and are critical to achieving synchronization between separate streams such as audio and video. Presentation timestamps lie at a layer above decoding timestamps and hence do not need to be visible to the client system until after decoding. In fact the decoder may generate most presentation timestamps from the decoding timestamps on the fly, for example, using a fixed delay. Presentation timestamps need to be explicitly different from decoding timestamps only in situations where frames must be presented out of decoding order. For example, MPEG frames $I_0, B_1, B_2, P_3, B_4, B_5, P_6, \ldots$ (in presentation order) must be decoded in the order $I_0, P_3, B_1, B_2, P_6, B_4, B_5, \ldots$. In the sequel we assume that frames are timestamped at the encoder with both decoder and presentation timestamps. They are inserted into the bit stream in decoding order. Henceforth in this chapter we will be concerned only with decoding timestamps.

It will pay to distinguish between the *clocks*, or temporal coordinate systems, in which timestamps are expressed. We use *media time* to refer to the clock running on the device used to capture and timestamp the original content, while *client time* refers to the clock running on the client used to play back the content. We assume that media time is real time (i.e., one second of media time elapses in one second of real time) at the time of media capture, while client time is real time at the time of media playback. We use the symbol τ to express media time and the symbol t to express client time, with subscripts and additional arguments to indicate corresponding events. For example, we use $\tau_{DTS}(0), \tau_{DTS}(1), \tau_{DTS}(2), \ldots$ to express the decoding deadlines of frames $0, 1, 2, \ldots$ in media time, while we use $t_{DTS}(0), t_{DTS}(1), t_{DTS}(2), \ldots$ to express the decoding deadlines of frames $0, 1, 2, \ldots$ at the client. The subscripts and/or arguments may be dropped or shortened if they are understood.

Content may be played back at a rate ν times real time. If $\nu = 2$, for example, the content is played back at twice real time (i.e., fast forward). The conversion from media time to client time can be expressed

$$t = t_0 + \frac{\tau - \tau_0}{\nu}, \tag{14.25}$$

where t_0 and τ_0 represent the time of a common initial event, such as the decoding time of frame 0 (or the decoding time of the first frame after a seek or rebuffering event) in media and client coordinate systems, respectively. Using (14.25), the leaky bucket update (14.2) becomes

$$F_e(n+1) = \max\{0, B_e(n) - R'[t(n+1) - t(n)]\}, \tag{14.26}$$

where $R' = R\nu$ is the arrival rate of bits into the client in bits per client time. Hence $R = R'/\nu$ is the rate that must be used to compute the required buffer size $B_e^{\min}(R)$ and initial decoder buffer fullness $F_d^{\min}(R)$. The preroll delay is thus $F_d^{\min}(R)/R' = F_d^{\min}(R)/R/\nu$. The larger the playback speed, the smaller the preroll delay.

14.4 STREAMING MEDIA OVER PACKET NETWORKS

In this section, we deal with streaming media over packet networks, such as the Internet. Unlike the idealized channel models of Section 14.3, packet networks are neither isochronous nor lossless, and they are shared by multiple communication processes, whose actions cause the network to have time-varying behavior. Hence, the major technical problem in streaming media on demand over packet networks is the need to maintain a good user experience in the face of time-varying network conditions. Users expect that regardless of the network conditions, the start-up delay will be low, playback will be continuous, and quality will be as high as possible given the average network bandwidth.

Buffering at the client is the key to meeting these user expectations. Technically, buffering serves several distinct but simultaneous purposes. First, as we have seen in Section 14.3, it allows the media to be coded with a variable instantaneous bit rate. Second, it allows the client to compensate for short-term variations in packet transmission delay (i.e., "jitter"). Third, it allows the client to continue playing back the media during lapses in network bandwidth. Finally, it gives the client time to perform packet loss recovery if needed. These last three purposes require additional buffer space at the client beyond the minimal buffer size $B^{\min}(R)$ computed in Section 14.3. However, a single buffer at the client can be shared among all four purposes. This section is primarily about how to keep the client buffer sufficiently full, on average, to serve all four purposes, and yet permit a low start-up delay, in the face of varying network conditions.

14.4.1 Source Coding, Channel Coding, Sending, Transmission, and Arrival Rates

In Section 14.3, we took pains to define the *source coding rate* R_c of a media stream as the slope of a tight buffer tube containing its schedule, measured in bits per second of media time.

Distinct from the source coding rate R_c is the *sending rate* R_s or the rate at which the server application injects bits into a reliable transport layer, measured in bits per second of client time.

Distinct from both the source coding rate R_c and the sending rate R_s is the *transmission rate* R_x, which is the rate at which the server injects bits into the network layer, again measured in bits per second of client time. The transmission

rate is limited, preferably, by the congestion control mechanism in either standard TCP or a TCP-friendly rate control (TFRC) mechanism [13] such as DCCP [25] or a nonstandard transmission rate control mechanism over UDP that adapts the transmission rate to the degree of network congestion. However, a simple transmission rate control mechanism could also set R_x to be constant, for example, irrespective of the level of network congestion. This could be appropriate if the channel is dedicated.

The sending rate R_s is limited, in turn, by the transmission rate R_x. In fact the difference between R_s and R_x can be attributed to the error control overhead, or redundancy, needed for reliable communication over the network, as described in Chapters 7–10. The fraction R_s/R_x is known as the *channel coding rate*, or the rate of the error control code in source bits per channel bit, which can be no greater than the Shannon capacity of the channel. The channel coding rate R_s/R_x is determined by the retransmission mechanism in TCP or by any other error control layer, which typically adapts the channel coding rate to the loss rate of the channel. TFRC and DCCP do not define an error control mechanism and hence an explicit mechanism must be provided for reliable communication, just as one must be provided for any nonstandard transmission rate control mechanism used over raw UDP or RTP/UDP. This is illustrated in Figure 14.8a.

Since all bits injected into a reliable transport layer eventually arrive at the client application, the sending rate R_s is over the long run equal to the *arrival rate* R_a, or the rate at which bits emerge at the client application. Since relatively little data can be buffered in the network and error control layer compared to the client buffer, we generally assume that R_a and R_s are essentially equivalent at any

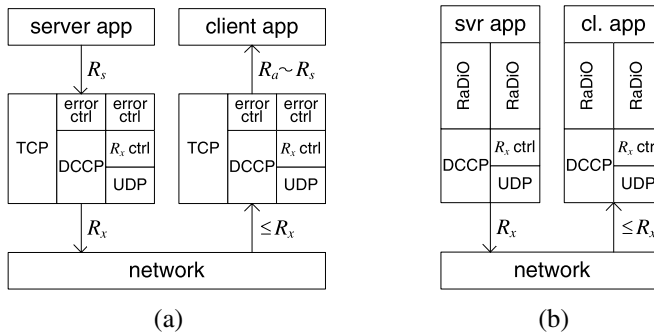


FIGURE 14.8: (a) Both error control and transmission rate control are factored out of the streaming application, which then deals only with source coding rate control. (b) Only transmission rate control-factored out of the streaming application, which performs joint source-channel coding such as in RaDiO (Chapter 10).

given time. In this chapter we will speak primarily in terms of the arrival rate R_a since it is from the client's point of view.

Many streaming media practitioners assume that once the user determines the playback speed ν, then the source coding rate R_c and the arrival rate R_a must be locked together by the relation $R_a = \nu R_c$ (except during the initial buffering period, during which $\nu = 0$ and $R_a > 0$). For example, if $\nu = 1$, a stream encoded at 100 Kbps must be communicated reliably at 100 Kbps (resulting in a raw transmission rate higher than 100 Kbps if retransmissions become necessary). However, this need not be the case, and in fact unlocking R_c and R_a can lead to important advantages.

The major advantage of decoupling the source coding and sending or arrival rates is that it makes possible continuous control of the number of seconds of content in the client buffer, known as the client buffer *duration*. The client buffer duration tends to increase or decrease depending on whether the arrival rate R_a (the average number of bits per second that arrive into the client buffer) is greater or less than the source coding rate R_c times the playback speed ν (the average number of bits per second that play out of the client buffer). That is, if $R_a > R_c\nu$ then the buffer duration increases; otherwise it decreases. For a given R_a and R_c, it is possible to adjust the playback speed ν to control the buffer duration. This approach is explored in detail in Chapter 16. On the one hand, in the long run, ν cannot remain very much different from the user's preferred playback speed. On the other hand, the arrival rate R_a is essentially limited by the network capacity. Hence, if the network capacity drops dramatically for a sustained period, reducing the source coding rate R_c is the only appropriate way to maintain the buffer duration and prevent an underflow leading to a rebuffering event. Adjusting the source coding rate in the face of time-varying network conditions is the problem of *source coding rate control*.

By continuously controlling the client buffer duration using source coding rate control, it is possible to begin playback after only $F_d^{\min}(R_a)/R_a$ seconds—when there is just enough data in the buffer to guarantee continuous playback under ideal, predictable conditions—then to continuously grow the client buffer duration to provide, over time, the necessary robustness to packet loss, jitter, and variations in network capacity—and finally to maintain, in the long run, a roughly constant buffer duration to ensure that playback quality is as high as the network capacity will allow. This helps meet the user expectations of low startup delay, continuous playback, and quality as high as possible given the average network bandwidth.

Being able to control R_c independently of R_s (or R_a) also makes efficient streaming possible with arbitrary transports, such as TCP, DCCP, or other proprietary transport protocols. Such protocols typically have their own error control and transmission rate control mechanisms, whose rates generally fluctuate according to network conditions and hence cannot be locked to a fixed source coding

rate unless they are locked well below the average network capacity, resulting in obvious inefficiency.

It is well worth mentioning here that if compatibility with TCP is not required, then it is possible for the source coding rate control and the error control modules illustrated in Figure 14.8a to be combined into a single module, as illustrated in Figure 14.8b. The transmission rate control module is not changed and still determines the frequency of transmission opportunities based on network congestion. In this way, the source coding and error control mechanisms can *jointly* determine whether a source packet or an error control packet (such as a retransmission of a previously transmitted packet, or a parity packet) should be put onto the wire at the next transmission opportunity. This is a joint source-channel coding problem sometimes known as the *scheduling* problem and is treated in detail in Chapter 10, using the RaDiO framework.

In this chapter, however, we will adopt the more classical approach of building a source coding layer on top of a reliable transport.

14.4.2 Source Coding Rate Control

In this section we detail an approach to source coding rate control based on the classical theory of optimal linear quadratic control [2]. The goal is to control the client buffer duration to a target, despite variations in the arrival rate R_a. This will be accomplished by choosing the source coding rate R_c as a function of R_a and the client buffer duration and its history (i.e., whether it is growing or shrinking). In order to choose an appropriate source coding rate R_c, we assume the existence at the server of multiple (or scalable) compound media streams at a variety of average bit rates $R^{(1)}, R^{(2)}, R^{(3)}, \ldots$, each having a schedule contained in an appropriately tight buffer tube $(R^{(i)}, B^{(i)}, F_e^{(i)})$, as illustrated in Figure 14.5.

14.4.2.1 Control Theoretic Model

Assume for the moment that bits arrive at the client at a constant rate R_a. Then frame n (having size $b(n)$, as illustrated in Figure 14.5) arrives at the client $b(n)/R_a$ seconds after frame $n-1$. Indeed, dividing the vertical scale of the schedules in Figure 14.5 by R_a, we obtain the schedules in terms of client time, rather than bits, as shown in Figure 14.9. The coding schedule divided by R_a becomes the *arrival schedule*, which provides for each n the time $t_a(n)$ of arrival of frame n at the client. The buffer tube upper bound (in bits) divided by R_a becomes the buffer tube upper bound (in time), which provides for each n the time $t_b(n)$ by which frame n is guaranteed to arrive. For continuous playback, it is sufficient that the buffer tube upper bound $t_b(n)$ be ahead of the *playback deadline* $t_d(n)$, which is the time at which frame n is scheduled to be instantaneously decoded

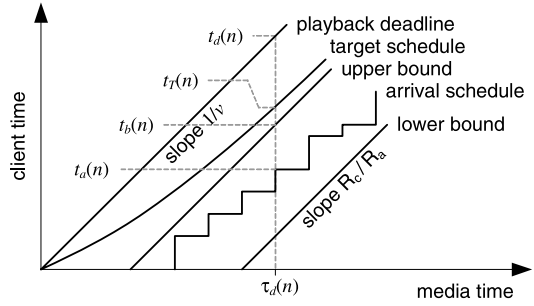


FIGURE 14.9: Arrival schedule and its upper bound in client time. The upper bound is controlled to the target schedule, which is increasingly in advance of the playback deadline to provide greater robustness over time.

and played. Note that the gap between a frame's arrival time $t_a(n)$ and its playback deadline $t_d(n)$ is the client buffer duration at the time of the frame's arrival. This must be nonnegative to allow continuous playback.

In reality the arrival rate is not constant. At any moment, the arrival rate may suddenly drop due to an increase in competing traffic, for example. Then each frame would take longer to be transmitted, the frame arrival times would become increasingly delayed, and the whole arrival schedule and its buffer tube would veer upward, threatening to cross over the playback schedule. To guard against such an event, if the buffer duration is too low, the buffer duration can be increased by switching to a lower source coding rate R_c, thereby reducing the slope of the buffer tube.

To see how, let $t_a(n-1)$ and $t_a(n)$ be the arrival times of frames n and $n-1$, respectively, and define

$$R_a(n) = \frac{b(n)}{t_a(n) - t_a(n-1)} \tag{14.27}$$

to be the *instantaneous arrival rate* at frame n. In practice, the average arrival rate at frame n will be estimated by a moving average $\tilde{R}_a(n)$ of previous values of $R_a(n)$, as detailed in the Appendix. Hence using (14.27) the arrival time of frame n can be expressed in terms of the arrival time of frame $n-1$ as

$$t_a(n) = t_a(n-1) + \frac{b(n)}{R_a(n)} \tag{14.28}$$

$$= t_a(n-1) + \frac{b(n)}{\tilde{R}_a(n)} + v(n), \tag{14.29}$$

where the $v(n)$ term is an error term that captures the effect of using the slowly moving average $\tilde{R}_a(n)$ instead of the instantaneous arrival rate $R_a(n)$.

How does the source coding rate R_c fit into all this? From Figure 14.5 it is clear that the decoder buffer fullness $F_d(n) = B - F_e(n)$ can also be expressed

$$F_d(n) = b(n) + g(n) = g(n-1) + \frac{R_c(n)}{f(n)}, \tag{14.30}$$

where

$$f(n) = \frac{1}{\tau(n) - \tau(n-1)} \tag{14.31}$$

is the instantaneous frame rate and the source coding rate $R_c(n)$ is now indexed by n, to take into account that different frames may lie in different buffer tubes with different coding rates as coding rate control is applied and streams are switched. Now from (14.30), we have

$$b(n) = \frac{R_c(n)}{f(n)} + g(n-1) - g(n), \tag{14.32}$$

whence [substituting (14.32) into (14.29)] we have

$$t_a(n) = t_a(n-1) + \frac{R_c(n)}{f(n)\tilde{R}_a(n)} + \frac{g(n-1)}{\tilde{R}_a(n)} - \frac{g(n)}{\tilde{R}_a(n)} + v(n). \tag{14.33}$$

Now defining the buffer tube upper bound (in time) of frame n as

$$t_b(n) = t_a(n) + \frac{g(n)}{\tilde{R}_a(n)}, \tag{14.34}$$

so that

$$t_b(n) - t_b(n-1) = t_a(n) - t_a(n-1) + \frac{g(n)}{\tilde{R}_a(n)} - \frac{g(n-1)}{\tilde{R}_a(n-1)}, \tag{14.35}$$

we obtain the following update equation:

$$t_b(n) = t_b(n-1) + \frac{R_c(n)}{f(n)\tilde{R}_a(n)} + w(n-1), \tag{14.36}$$

where

$$w(n-1) = \frac{g(n-1)}{\tilde{R}_a(n)} - \frac{g(n-1)}{\tilde{R}_a(n-1)} + v(n) \tag{14.37}$$

is again an error term that captures variations around a locally constant arrival rate.

Using (14.34), the client can compute $t_b(n-1)$ from the measured arrival time $t_a(n-1)$, the estimated arrival rate $\tilde{R}_a(n-1)$, and $g(n-1)$ [which can be transmitted to the client along with the data in frame $n-1$ or computed at the client from $g(n-2)$ and $R_c(n-1)$ using (14.30)]. Then using (14.36), the client can control the coding rate $R_c(n)$ so that $t_b(n)$ reaches a desired value, assuming the frame rate and arrival rate remain roughly constant. From this perspective, (14.36) can be regarded as the state transition equation of a feedback control system and it is thus possible to use a control-theoretic approach to regulate the coding rate.

14.4.2.2 Control Objective

With the state transition equation defined in (14.36), uninterrupted playback can be achieved by regulating the coding rate so that the client buffer does not underflow. To introduce a margin of safety that increases over time, we introduce a *target schedule*, illustrated along with the buffer tube in Figure 14.9, whose distance from the playback deadline grows slowly over time. By regulating the coding rate, we attempt to control the buffer tube upper bound so that it tracks the target schedule. If the buffer tube upper bound is close to the target schedule, then the arrival times of all frames will certainly be earlier than their playback deadlines and thus uninterrupted playback will be ensured. Note that controlling the actual arrival times (rather than their upper bounds) to the target would result in an approximately constant number of bits per frame, which would in turn result in very poor quality overall. By taking the leaky bucket model into account, we are able to establish a control that allows the instantaneous coding rate to fluctuate naturally according to the encoding complexity of the content, within previously established bounds for a given average coding rate.

Although controlling the upper bound to the target schedule is our primary goal, we also wish to minimize quality variations due to large or frequent changes to the coding rate. This can be achieved by introducing into the cost function a penalty for relative coding rate differences.

Letting $t_T(n)$ denote the target schedule for frame n, we use the following cost function to reflect both of our concerns:

$$I = \sum_{n=0}^{N} \left(\big(t_b(n) - t_T(n)\big)^2 + \sigma \left(\frac{R_c(n+1) - R_c(n)}{\tilde{R}_a(n)} \right)^2 \right), \tag{14.38}$$

where the first term penalizes the deviation of the buffer tube upper bound from the target schedule and the second term penalizes the relative coding rate difference between successive frames. N is the control window size and σ is a Lagrange multiplier or weighting parameter to balance the two terms.

14.4.2.3 Target Schedule Design

Figure 14.10 shows an illustrative target schedule. The gap between the playback deadline and the target schedule is the desired minimum client buffer duration. If the gap is small at the beginning of streaming, then it allows a small start-up delay, whereas if the gap grows slowly over time, it gradually increases the client's ability to counter jitter, delays, and throughput changes.

The slope of the target schedule relates the average source coding rate to the average arrival rate. Let $t_T(n)$ be the target for frame n. As illustrated in Figure 14.10, the slope of the target schedule at frame n is

$$s(n) = \frac{t_T(n+1) - t_T(n)}{\tau(n+1) - \tau(n)}. \tag{14.39}$$

If the upper bound $t_b(n)$ aligns perfectly with the target schedule [i.e., $t_b(n) = t_T(n)$] and the arrival rate R_a is constant [i.e., the $w(n-1)$ term vanishes], we get from (14.36)

$$s(n) = \frac{t_b(n+1) - t_b(n)}{\tau(n+1) - \tau(n)} = \frac{R_c(n+1)}{R_a}. \tag{14.40}$$

Thus initially, if the slope is low, that is, $s(n)$ is less than $1/\nu$, then R_a is greater than $R_c\nu$, causing the client buffer to grow. Over time, as the slope approaches $1/\nu$, R_a approaches $R_c\nu$, and the buffer remains relatively constant, except for changes due to variations in the instantaneous coding rate.

A reasonable way to choose the target schedule t_T is to have the client buffer duration grow logarithmically over time. Specifically, if t_d is the playback dead-

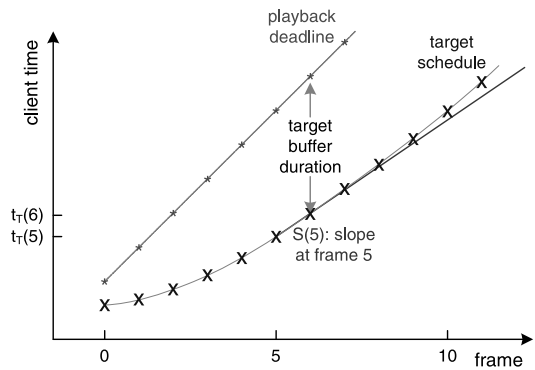


FIGURE 14.10: Target schedule design.

line, then for each t_d greater than some start time t_{d0},

$$t_T = t_d - \frac{b}{a}\ln\big(a(t_d - t_{d0}) + 1\big). \tag{14.41}$$

Since $t_d = t_{d0} + (\tau_d - \tau_{d0})/\nu$ by (14.25), we have

$$s = \frac{dt_T}{d\tau_d} = \frac{dt_T}{dt_d}\frac{dt_d}{d\tau_d} = \frac{1}{\nu} - \frac{b}{a(\tau_d - \tau_{d0}) + \nu}, \tag{14.42}$$

and hence the initial slope at frame 0 (when $t_d = t_{d0}$) is $s(0) = (1-b)/\nu$. Setting $b = 0.5$ implies that initially $R_c/R_a = 0.5/\nu$, causing the client buffer to grow initially at two times real time. Further setting $a = 0.15$ implies that the client buffer duration will be 7.68 s after 1 min, 15.04 s after 10 min, and 22.68 s after 100 min, regardless of ν.

14.4.2.4 Controller Design

Recall from (14.36) the fundamental state transition equation, which describes the evolution of the buffer tube upper bound $t_b(n)$ in terms of the source coding rate $R_c(n)$:

$$t_b(n+1) = t_b(n) + \frac{R_c(n+1)}{f\tilde{R}_a} + w(n). \tag{14.43}$$

Here we now assume that the frame rate f and the average arrival rate $\tilde{R}_a$ are relatively constant. Deviations from this assumption are captured by $w(n)$.

We wish to control the upper bound by adjusting the source coding rate. As each frame arrives at the client, a feedback loop can send a message to the server to adjust the source coding rate. Note, however, that by the time frame n arrives completely at the client, frame $n+1$ has already started streaming from the server. Thus the coding rate $R_c(n+1)$ for frame $n+1$ must already be determined by time $t_a(n)$. Indeed, at time $t_a(n)$, frame $n+2$ is the earliest frame for which the controller can determine the coding rate. Hence at time $t_a(n)$, the controller's job must be to choose $R_c(n+2)$. We must explicitly account for this one-frame delay in our feedback loop.

For simplicity, we linearize the target schedule around the time that frame n arrives. The linearization is equivalent to using a line tangent to the original target schedule at a particular point as an approximate target schedule. Thus we have

$$t_T(n+1) - 2t_T(n) + t_T(n-1) = 0. \tag{14.44}$$

Rather than directly control the evolution of the upper bound, which grows without bound, for the purposes of stability we use an error space formulation.

By defining the error

$$e(n) = t_b(n) - t_T(n), \tag{14.45}$$

we obtain

$$\begin{aligned} e(n+1) - e(n) &= \big(t_b(n+1) - t_T(n+1)\big) - \big(t_b(n) - t_T(n)\big) && (14.46)\\ &= \big(t_b(n+1) - t_b(n)\big) - \big(t_T(n+1) - t_T(n)\big) && (14.47)\\ &= \frac{R_c(n+1)}{f\tilde{R}_a} - \big(t_T(n+1) - t_T(n)\big) + w(n), && (14.48) \end{aligned}$$

from which we obtain in turn

$$\begin{aligned} &\big(e(n+1) - e(n)\big) - \big(e(n) - e(n-1)\big)\\ &\quad = \big[R_c(n+1) - R_c(n)\big]/f\tilde{R}_a\\ &\qquad - \big(t_T(n+1) - 2t_T(n) + t_T(n-1)\big)\\ &\qquad + \big(w(n) - w(n-1)\big) && (14.49)\\ &\quad = \frac{R_c(n+1) - R_c(n)}{f\tilde{R}_a} + \big(w(n) - w(n-1)\big). && (14.50) \end{aligned}$$

We next define the control input

$$u(n) = \frac{R_c(n+2) - R_c(n+1)}{\tilde{R}_a}, \tag{14.51}$$

and we define the disturbance

$$d(n) = w(n) - w(n-1). \tag{14.52}$$

Then (14.50) can be rewritten

$$e(n+1) = 2e(n) - e(n-1) + \frac{u(n-1)}{f} + d(n). \tag{14.53}$$

Therefore, defining the state vector

$$\mathbf{e}(n) = \begin{bmatrix} e(n)\\ e(n-1)\\ u(n-1) \end{bmatrix} = \begin{bmatrix} t_b(n)\\ t_b(n-1)\\ \frac{R_c(n+1)}{\tilde{R}_a} \end{bmatrix} - \begin{bmatrix} t_T(n)\\ t_T(n-1)\\ \frac{R_c(n)}{\tilde{R}_a} \end{bmatrix}, \tag{14.54}$$

the error space representation of the system can be expressed

$$\mathbf{e}(n+1) = \begin{bmatrix} 2 & -1 & \frac{1}{f} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{e}(n) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(n) + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} d(n), \tag{14.55}$$

or $\mathbf{e}(n+1) = \Phi\mathbf{e}(n) + \Gamma u(n) + \Gamma_d d(n)$ for appropriate matrices Φ, Γ, and Γ_d.

Assuming the disturbance $d(n)$ is a pure white noise, and assuming *perfect state measurement* (i.e., we can measure all components of $\mathbf{e}(n)$ without using an estimator), the disturbance $d(n)$ does not affect the controller design. Thus we can use a linear controller represented by

$$u(n) = -G\mathbf{e}(n), \tag{14.56}$$

where G is a vector *feedback gain*. By the time frame n is completely received, all elements of $\mathbf{e}(n)$ are available at the client and $u(n)$ can thus be computed. The ideal coding rate for frame $n+2$ can then be computed as

$$R_c(n+2) = R_c(n+1) - G\mathbf{e}(n)\tilde{R}_a. \tag{14.57}$$

Finding the optimal linear controller amounts to finding the feedback gain G^* that minimizes the quadratic cost function (14.38). Before continuing with the design, we first check the system *controllability matrix* $\mathcal{C}$,

$$\mathcal{C} = [\,\Gamma \quad \Phi\Gamma \quad \Phi^2\Gamma\,] = \begin{bmatrix} 0 & \frac{1}{f} & \frac{2}{f} \\ 0 & 0 & \frac{1}{f} \\ 1 & 0 & 0 \end{bmatrix}, \tag{14.58}$$

which has full rank for any frame rate f. Thus, the system is *completely controllable* [16] and the state $\mathbf{e}(n)$ can be regulated to any desirable value. Now recall that the cost function (14.38) is

$$I = \sum_{n=0}^{N} \left\{ \big(t_b(n) - t_T(n)\big)^2 + \sigma \left(\frac{R_c(n+1) - R_c(n)}{\tilde{R}_a} \right)^2 \right\} \tag{14.59}$$

$$= \sum_{n=0}^{N} \big\{ \mathbf{e}(n)^T Q\mathbf{e}(n) + u(n-1)^T Ru(n-1) \big\}, \tag{14.60}$$

where $Q = C^T C$ (with $C = [1 \;\; 0 \;\; 0]$) and $R = \sigma$. Then, the original control problem of tracking the target schedule while smoothing the coding rate fluctuations

(i.e., minimizing the cost function I) is converted to a standard regulator problem in the error space. Letting $N \to \infty$, the infinite horizon optimal control problem can be solved by applying the results in [2, Section 3.3] to obtain an optimal regulator in two steps: (1) solving, to get S, the discrete algebraic Riccati equation

$$S = \Phi^T \left\{ S - S\Gamma \left[\Gamma^T S \Gamma + R \right]^{-1} \Gamma^T S \right\} \Phi + Q, \tag{14.61}$$

and (2) computing the optimal feedback gain

$$G^* = \left[\Gamma^T S \Gamma + R \right]^{-1} \Gamma^T S \Phi. \tag{14.62}$$

The existence and uniqueness of S (and in turn of G^*) are guaranteed when Q is nonnegative definite and R is positive definite, which is straightforward to verify in our case.

14.4.2.5 Effect of Frame Rate

In the derivation given earlier, we assumed that the frame rate is constant. This assumption is reasonable when streaming a single medium, such as video without audio. However, usually video and audio are streamed together, and their merged coding schedule may have no fixed frame rate. Even if there is a fixed frame rate f, we may wish to operate the controller at a rate lower than f to reduce the feedback rate, for example.

To address these issues, in practice we use the notion of a *virtual frame rate*. We choose a virtual frame rate f, for example, $f = 1$ frame per second (fps); we partition media time into intervals of size $1/f$; and we model all of the (audio and video) frames arriving within each interval as a *virtual frame* whose decoding and playback deadline is the end of the interval.

This approach has several advantages. First, it allows us to design off line a universal feedback gain, which is independent of the actual frame rate of the stream or streams. Second, it allows us to reduce the rate of feedback from the client to the server. Finally, since the interval between virtual frames is typically safely larger than a round trip time (RTT), a one-frame delay in the error space model (as described in the previous section) is sufficient to model the feedback delay. Otherwise we would have to model the feedback delay with approximately RTT/f additional state variables to represent the network delay using a shift register of length RTT/f.

In the sequel we therefore use a virtual frame rate $f = 1$ fps, and we refer to this simply as the frame rate. With $f = 1$ and $\sigma = 50$ (chosen empirically for good damping), we can solve for $G^* = [0.6307, -0.5225, 0.5225]$.

14.4.2.6 Controller Interpretation

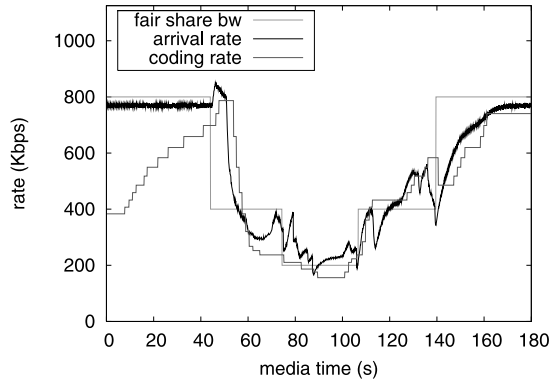
With the aforementioned coefficients for G^*, we are now able to give an intuitive explanation of the source coding rate control (14.57). Plugging the coefficients of G^* into (14.57), we obtain

$$\begin{aligned} R_c(n+2) = R_c(n+1) & \\ - 0.1082 e(n) \tilde{R}_a & \quad (14.63) \\ - 0.5225\big[e(n) - e(n-1)\big]\tilde{R}_a & \quad (14.64) \\ - 0.5225\big[R_c(n+1) - R_c(n)\big]. & \quad (14.65) \end{aligned}$$

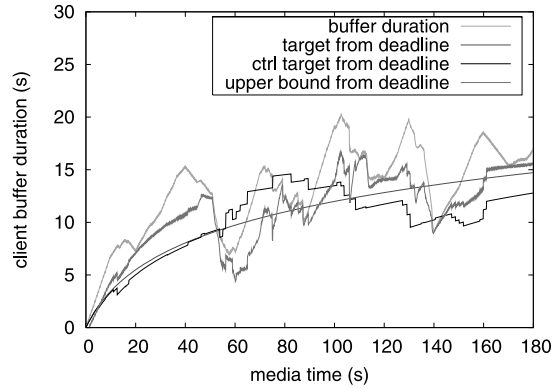
Focusing on the first term (14.63), it can be seen that the source coding rate R_c tends to decrease if the current error $e(n) = t_b(n) - t_T(n)$ is positive, and it tends to increase if $e(n)$ is negative, in proportion to $e(n)$ with proportionality constant 0.1082 times the estimated arrival rate $\tilde{R}_a$. This has the effect of moving the upper bound t_b toward the target t_T, whether it is above or below the target. At the same time, from the second term (14.64), it can be seen that the source coding rate tends to decrease if the current error $e(n)$ is numerically greater than the previous error $e(n-1)$, whether $e(n)$ is positive or negative. This has the effect of either strengthening the compensation or preventing the controller from overcompensating, since if $e(n)$ is positive then $e(n) > e(n-1)$ indicates that the magnitude of the error is still growing, whereas if $e(n)$ is negative then $e(n) > e(n-1)$ indicates that the magnitude of the error is shrinking too fast to be sustainable. The proportionality constant for this second effect is 0.5225 times $\tilde{R}_a$, which is even larger than that for the first effect. Finally, from the third term (14.65), it can be seen that the source coding rate tends to decrease if it had previously increased, with proportionality constant 0.5225. This ensures appropriate damping and smoothing of the source coding rate.

It is important to emphasize that the optimal feedback gain G^* is completely determined given σ and f, and that it is independent of the arrival rate and the source coding rate. Thus, G^* can be obtained off line, and only a linear calculation is required to compute the source coding rate $R_c(n+2)$ on the fly.

Figure 14.11 shows results of a simulation in which FTP cross traffic reduces the fair share for streaming from 800 to 200 Kbps and back again over 180 s. Figure 14.11a shows the fair share bandwidth, as well as the rate of arrival of reliable information over TCP and the resulting source coding rate. The source coding rate starts at about half the arrival rate to build up the client buffer duration and approaches the arrival rate over the first 45 s. Subsequently the source coding rate tends to track the arrival rate, but with less variation. Figure 14.11b shows the associated client buffer duration $t_d(n) - t_a(n)$, as well as the target $t_d(n) - t_T(n)$



(a)



(b)

FIGURE 14.11: Variable bandwidth over TCP.

and the buffer tube upper bound $t_d(n) - t_b(n)$ relative to the deadline. Despite sudden large changes in the TCP arrival rate (e.g., at the 50-s mark), the buffer duration remains safely positive and recovers quickly to the target.

14.5 LIVE BROADCAST

Section 14.2 covered the architectural differences between live broadcast and streaming media on demand. In live broadcast, the encoder is online, but communicates through the server to multiple clients simultaneously. Thus even though the encoder is online, it is still not feasible to know the channel conditions be-

tween the server and each client separately. In general, the clients experience different channels, such as different rates of packet loss, different patterns of packet loss, and different limitations on bit rate. Thus a major issue in live broadcast is dealing with the *heterogeneity of channels* across clients, in terms of both error and bit rate characteristics.

Another issue in live broadcast is dealing with the *heterogeneity of devices* across clients, in terms of resolution and computational power. Mobile phones have limited screen real estate, a limited number of audio channels, and limited computational power for decoding. In contrast, home media centers may have high-definition projection monitors, multichannel surround sound, multigigahertz, multicore computational engines, and specialized hardware or firmware decoders. Because of such heterogeneity in both devices and channels, the encoder cannot generally produce a single encoding that is satisfactory to all clients.

A time-honored means of broadcasting to heterogeneous clients is *simulcast*. Terrestrial television broadcast in North America, for example, is currently simulcasting (simultaneously broadcasting) standard definition and high-definition programs. This is analogous to multibit rate files for streaming media on demand. As a result, it is always possible to address heterogeneous client populations by simulcasting streams with different bit rates, error protection, source resolutions, and decoding requirements. Each client can "tune in" to the appropriate "channel" depending on its needs. In the case of IP multicast, each "channel" is represented by an IP multicast group address (or simply *multicast address*) [31]. Once these addresses and the descriptions of their contents are known (using an out-of-band mechanism such as the session announcement and description protocols [19,20]), it is possible for a client to subscribe to the appropriate multicast address to obtain an appropriate stream.

A means of broadcasting to heterogeneous clients that makes more efficient use of network resources is based on scalable coding rather than multibit rate coding. In *layered multicast*, each layer of a scalable encoding is multicast to a different address. Each client subscribes to the appropriate set of addresses to obtain the appropriate layers of the encoding. This idea was first operationalized by McCanne [28], although the idea itself precedes McCanne's work. In McCanne's protocol, called *receiver-driven layered multicast*, each client continually probes for bandwidth by subscribing to and unsubscribing from (i.e., adding and dropping) the client's current topmost enhancement layer. This is an effective way to adapt the bit rate and to provide congestion control in bandwidth-heterogeneous IP multicast networks.

McCanne's work did not address error control. As noted earlier, different clients generally experience different packet loss rates and packet loss patterns, in addition to different bit rate limitations. Unfortunately, the standard error control technique of end-to-end packet retransmission, used so effectively in streaming media on demand to control packet loss adaptively, cannot be used in the broadcast

scenario because of limitations on feedback to the server. Timely feedback of either positive or negative acknowledgments from the clients to the server would cause a *feedback implosion* at the server, and hence would not scale to large numbers of clients.

However, *statistical feedback*, as opposed to timely feedback, is possible. The common example of statistical feedback is RTCP receiver reports [35], which can be periodically sent from the client to each server to inform the server of the average packet loss rate. In multicast situations, the clients can send the receiver reports at random intervals with a frequency inversely proportional to the number of clients in the session. In this way, the rate at which receiver reports arrive at the server remains approximately constant regardless of the number of clients. The server can use such statistical feedback to understand its client population and to tailor one or more streams to the population.

Because timely feedback is generally not feasible in multicast settings, error control is usually based on feedforward rather than feedback mechanisms. Forward error correction (FEC) coding [27] or, more precisely, forward erasure coding is typically used to control packet loss in multicast settings. The Windows Media Server, for example, uses a systematic Reed–Solomon style erasure code, as described in Chapter 7, to generate and transmit $N - K$ parity packets after every block of K source packets when broadcasting over IP multicast. (IP multicast is widely used on the Microsoft campus to broadcast lectures, meetings, etc.) Thus every "source block" of K source packets maps to a "code block" of N source and parity packets. If at least K out of N packets in a code block are received (i.e., if no more than $N - K$ packets in a code block are lost), then the corresponding K source packets can still be recovered. The parameters (N, K) can be optimized to match the packet loss characteristics of the client population, as can be determined by RTCP receiver reports. Typically, $(N - K)/N$ is set close to the worst-case error rate.

More efficient than FEC, however, to protect scalable media is *priority encoded transmission* (PET) [1], as described in Chapter 9. FEC tends to degrade sharply if more than $N - K$ packets in a code block are lost—a "cliff" effect. In contrast, PET degrades gracefully as more packets are lost in a code block. This is because in PET, the number of layers that can be recovered by the client in the scalable source encoding is equal to the number of packets that are received, as illustrated in Figure 14.12. Figure 14.13 shows video quality as a function of the number of packets received per code block. Like FEC, PET can be optimized as a function of the packet loss characteristics of the client population. If $q_0, q_1, \ldots, q_N$ is the probability distribution of the number of packets received by a client in any block of N code packets, then the procedures detailed in Chapter 9 can be used to optimize the PET parameters $\mathbf{R} = (R_0, R_1, \ldots, R_N)$ to minimize (subject to a rate constraint) the average distortion $D = \sum_{n=0}^{N} q_n D(R_n)$ over the population. The resulting bit stream can be packetized as described in [26].

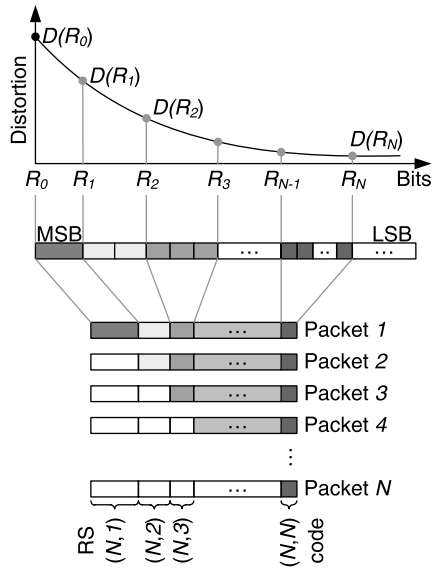


FIGURE 14.12: PET packetization. An embedded bit stream with distortion–rate function $D(R)$ is partitioned into N layers and poured into N packets. Layer K is protected with an (N, K) RS code such that if any K out of N packets are received, the first K layers are recoverable.

Optimizing the FEC or PET parameters for a heterogeneous client population may result in a stream that is far from optimal for any homogeneous subpopulation. In this situation, it may pay to provide multiple streams, each one targeted to a sub-population. Relatively homogeneous sub-populations can be identified by clustering, as follows. Suppose $q_{\theta,0}, q_{\theta,1}, \ldots, q_{\theta,N}$ is the probability distribution of the number of packets received by client $\theta \in \Lambda$ in any block of N code packets, where Λ represents the overall population of clients. Let $m(\theta) : \Lambda \to \{1, \ldots, M\}$ be the assignment of client θ to one of M sub-populations $\Lambda_m \subseteq \Lambda$ and let $\mathbf{R}(m) = (R_{m,0}, R_{m,1}, \ldots, R_{m,N})$ be the PET parameters for subpopulation Λ_m. Clearly, the optimal assignment for each client θ is the one that maps θ to the sub-population Λ_m whose PET parameters $\mathbf{R}(m)$ result in the lowest distortion for client θ, that is,

$$m(\theta) = \arg\min_{m} \sum_{n=0}^{n} q_{\theta,n} D(R_{m,n}). \tag{14.66}$$

FIGURE 14.13: PET quality as a function of number of packets received. From top to bottom, left to right: zero to eight packets received (out of eight transmitted).

However, the optimal PET parameters for each sub-population Λ_m minimize the average distortion for the sub-population, that is,

$$\mathbf{R}(m) = \arg\min_{\mathbf{R}} \sum_{n=0}^{N} q_{m,n} D(R_n), \tag{14.67}$$

where $q_{m,n}$ is the average of $q_{\theta,n}$ over $\theta \in \Lambda_m$. The minimization (14.67) can be performed with the algorithms in Chapter 9. The minimizations (14.66) and (14.67) can be repeated until the distortion averaged over the entire population Λ converges to a minimum, determining both the clustering and the PET parameters for each cluster [8].

Using FEC or PET for error control can be combined with either simulcast or layered multicast. The simulcast case is straightforward. As usual, the client population is partitioned into sub-populations according to bit rate. Then for each bit rate, FEC or PET is applied to make a loss-resilient stream at that bit rate. Multiple loss-resilient streams for a given bit rate are also possible, as described earlier, if the clients at that bit rate are heterogeneous in terms of loss.

Combining FEC or PET with layered multicast is trickier because insufficient protection of a lower (e.g., base) layer will render a higher (e.g., enhancement) layer irrelevant. Thus in general the lower layers need more error protection than the higher layers. Furthermore, the optimum amount of protection of any given layer increases as the overall bit rate increases. Therefore, as enhancement layers are added, stronger error protection must also be added for each of the preceding layers. Hence the error protection itself must be layered. In addition, the available bit rate must be optimally allocated to the various layers.

A natural approach to this problem is given in [7,40]. Suppose that all the source layers have an equal rate, say one packet per group of frames (GOF). As illustrated in Figure 14.14, partition the packets in each source layer into blocks having K packets per block, where the block size K is constant across all source layers and the block boundaries are synchronized across layers. For each block of K source packets in a source layer, generate $N - K$ parity packets using a systematic (N, K) Reed–Solomon style erasure code, where the "code length" N is determined by the maximum amount of redundancy that will be needed by any client to protect the source layer. Place each of the parity packets so generated in its own multicast stream so that each source layer is accompanied by $N - K$ parity layers, each at $1/K$ the rate of the source layer. In this way, a client now has many layers to which it can subscribe. It can subscribe to any collection of source layers and any collection of parity layers associated with those source layers.

The problem now is to determine the layers to which a client should subscribe to minimize the expected distortion given a total bit rate constraint and a packet loss rate. Let N_l be the number of code packets (i.e., source packets plus parity packets) for source layer l to which the client subscribes, $l = 1, \ldots, L$. Then N_l takes the value 0 if the client does not subscribe to source layer l or any of its associated parity layers; it takes the value K if the client subscribes to source layer

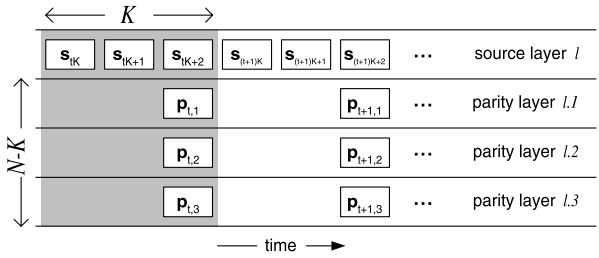


FIGURE 14.14: Generation of parity packets: block each source layer into K packets per block; produce $N - K$ parity packets per block with a systematic RS-style code; and send each parity packet to a different multicast address.

l but to none of its associated parity layers; and it takes the values $K+1, \ldots, N$ if the client subscribes to source layer l and $1, \ldots, N-K$ of its associated parity layers, respectively. Let the redundancy $\pi_l = N_l/K$ be the number of packets per GOF transmitted to the client in layer l. (This is the inverse of the code rate.) The vector $\pi = (\pi_1, \ldots, \pi_L)$, called the *transmission policy*, specifies which source layers to subscribe to and also which parity layers to subscribe to for each source layer. Any given transmission policy π induces a total bit rate and an expected distortion. The total bit rate, in terms of transmitted packets per GOF, is

$$R(\pi) = \sum_l \pi_l, \tag{14.68}$$

whereas the expected distortion per GOF is

$$D(\pi) = D_0 - \sum_l \Delta D_l \prod_{l' \preceq l} \big(1 - \varepsilon(\pi_{l'})\big), \tag{14.69}$$

as shown in [7]. Here, D_0 is the distortion if no packets in a GOF can be decoded, ΔD_l is the distortion reduction if the packet in layer l can be decoded, and the product $\prod_{l' \preceq l}(1 - \varepsilon(\pi_{l'}))$ is the probability for this to occur, as described in Chapter 10. The notation $l' \preceq l$ is shorthand for the set of layers l' on which layer l depends for decoding, and $\varepsilon(\pi_l)$ is the residual probability of packet loss after channel decoding under the transmission policy. In [7] this is shown to be $1 - M(N_l, K)/K$, where

$$M(N_l, K) = \sum_{i=0}^{N_l} \binom{N_l}{i} \varepsilon^{N_l - i} (1-\varepsilon)^i M(N_l, K \mid i) \tag{14.70}$$

is the expected number of source packets that can be recovered after channel decoding with a (N_l, K) code. Here $M(N_l, K \mid i)$ equals K if $i \geq K$ and equals iK/N_l if $i < K$, and ε is the packet loss rate for the client in question.

Chande and Farvardin [5] provide a dynamic programming algorithm to find the transmission policy π that minimizes $D(\pi)$ subject to $R(\pi) \leq R_{\max}$ when the dependencies between layers are sequential. When the dependencies are given more generally by a directed acyclic graph, then the transmission policy can be optimized by the Iterative Sensitivity Adjustment algorithm [6,7], as discussed in Chapter 10.

An alternative approach to combining error control with layered coding is based on PET, as described in [9,39]. However, this approach gets quite complicated with any more than two source layers.

In the aforementioned discussion, we assumed a dumb network such as an IP network, in which the network nodes can only copy and forward data. However, if

the network nodes are more intelligent, for example, if the network is an overlay network of servers or peers, then much more can be done. As a simple example, if the network nodes are computers, then communication between nodes can be made reliable by using sufficient buffering and retransmission (e.g., using TCP), thus eliminating the problem of error control altogether (although rate control remains a problem). For example, today's CoolStreaming and PPLive applications [41,42] deliver IP television peer to peer with essentially no glitching as long as the available bandwidth is high enough.

14.6 SUMMARY AND FURTHER READING

In this chapter, we treated both streaming media on demand and live broadcast. An attempt was made, more than in any other book we know, to formalize these systems and to show how they can be optimized.

Much of Section 14.3, on leaky bucket models, is based on [34]. Another primary source of information on leaky buckets is [21]. Much of Section 14.4, on rate control, is based on [22–24]. Much of Section 14.5, on multicast, is based on [7,8]. Please see these primary sources for more detailed information.

For more general information on streaming media, see the excellent books by Crowcroft *et al.* [12] and Perkins [32]. Of course an amazing amount of information is freely available through the web.

ACKNOWLEDGMENTS

The author is indebted to Anders Klemets, Cheng Huang, Jordi Ribas-Corbera, Shankar Regunathan, and Albert Wang for their contributions to this chapter.

APPENDIX: RATE ESTIMATION

This section details our exponential averaging algorithm for the arrival rate, the preferred algorithm in any context in which a bit rate must be estimated from a sequence of variable-sized packets with variable inter-packet intervals.

Let $\tilde{R}_a(k)$ and $R_a(k)$ be the average arrival rate and the instantaneous arrival rate, respectively, when packet k is received. Note that unlike the controlling operation, the rate averaging operation may be performed after the arrival of every *packet* rather than after the arrival of every *frame*. Hence we use the discrete packet index k rather than the frame index n. Instead of using the widely adopted

exponentially weighted moving average (EWMA)

$$\tilde{R}_a(k) = \beta(k)\tilde{R}_a(k-1) + \big(1-\beta(k)\big)R_a(k) \tag{14.71}$$

with constant $\beta(k) = \beta$, we perform the exponential averaging more carefully. In our algorithm, the factor $\beta(k)$ is not constant, but varies according to the packets' interarrival gaps. Our algorithm has several advantages over the EWMA algorithm with constant $\beta(k)$. First, the estimate of the average arrival rate $\tilde{R}_a(k)$ goes to zero naturally as the gap since the last packet goes to infinity, rather than being bounded below by $\beta\tilde{R}_a(k-1)$. Second, the estimate of the average arrival rate $\tilde{R}_a(k)$ does not go to infinity as the gap since the last packet goes to zero. This is especially important, as packets often arrive in bursts, causing extremely high instantaneous arrival rates. Finally, the estimate of the average arrival rate $\tilde{R}_a(k)$ does not overweight the initial condition, as if it represented the infinite past. This is especially important in the early stages of estimation.

As in (14.27), we define the instantaneous arrival rate after packet k as

$$R_a(k) = \frac{b(k)}{t_a(k) - t_a(k-1)}, \tag{14.72}$$

where here $b(k)$ denotes the size of packet k and $t_a(k)$ denotes the arrival time of packet k. We extend the discrete time function $R_a(k)$ to the piecewise constant continuous time function $R_a(t)$ by

$$R_a(t) = R_a(k) \quad \text{for all } t \in \big[t_a(k-1), t_a(k)\big], \tag{14.73}$$

as illustrated in Figure 14.15. Then we filter the function $R_a(t)$ by the exponential impulse response $\alpha e^{-\alpha t}$, $t \geq 0$, for some time constant $1/\alpha$:

$$\tilde{R}_a(k) = \frac{\int_{t(0)}^{t(k)} R_a(t')\alpha e^{-\alpha(t(k)-t')}dt'}{\int_{t(0)}^{t(k)} \alpha e^{-\alpha(t(k)-t')}dt'}. \tag{14.74}$$

[Here and in the remainder of the Appendix we suppress the subscript from the arrival time $t_a(k)$.] Noting that $\int_t^{\infty} \alpha e^{-\alpha t'}dt' = e^{-\alpha t}$, the denominator integral can be expressed $1 - e^{-\alpha(t(k)-t(0))}$. Now, we split the range of the numerator integral into ranges $[t(0), t(k-1)]$ and $[t(k-1), t(k)]$ to obtain a recursive expression for $\tilde{R}_a(k)$ in terms of $\tilde{R}_a(k-1)$ and $R_a(k)$,

$$\tilde{R}_a(k) = \frac{1 - e^{-\alpha[t(k-1)-t(0)]}}{1 - e^{-\alpha[t(k)-t(0)]}} e^{-\alpha[t(k)-t(k-1)]}\tilde{R}_a(k-1)$$

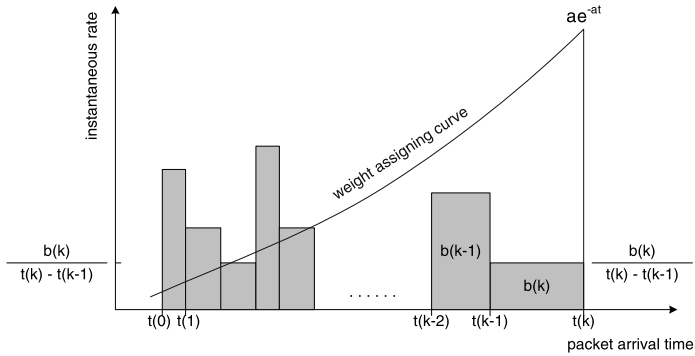


FIGURE 14.15: Exponential averaging.

$$+\frac{1-e^{-\alpha[t(k)-t(k-1)]}}{1-e^{-\alpha[t(k)-t(0)]}}R_a(k) \tag{14.75}$$

$$=\beta(k)\tilde{R}_a(k-1)+(1-\beta(k))R_a(k), \tag{14.76}$$

where

$$\beta(k)=\frac{e^{-\alpha[t(k)-t(k-1)]}-e^{-\alpha[t(k)-t(0)]}}{1-e^{-\alpha[t(k)-t(0)]}}. \tag{14.77}$$

Note that $\beta(k)$ is numerically stable as k goes to infinity. However, as the gap $\delta = t(k) - t(k-1)$ goes to zero, $1-\beta(k)$ goes to zero while $R_a(k)$ goes to infinity. Their product, however, is well behaved. Indeed,

$$\tilde{R}_a(k) = \frac{1-e^{-\alpha[t(k-1)-t(0)]}}{1-e^{-\alpha[\delta+t(k-1)-t(0)]}}e^{-\alpha\delta}\tilde{R}_a(k-1)$$

$$+\frac{1-e^{-\alpha\delta}}{1-e^{-\alpha[t(k)-t(0)]}}\frac{b(k)}{\delta} \tag{14.78}$$

$$\to \tilde{R}_a(k-1)+\frac{\alpha b(k)}{1-e^{-\alpha[t(k)-t(0)]}} \tag{14.79}$$

as $\delta \to 0$, using l'Hôpital's rule. Thus (14.79) is the update rule in the case when $t(k) = t(k-1)$.

REFERENCES

[1] A. Albanese, J. Blömer, J. Edmonds, M. Luby, and M. Sudan. "Priority Encoding Transmission," *IEEE Trans. Information Theory*, 42:1737–1744, November 1996.

[2] B. D. O. Anderson and J. B. Moore. *Optimal Control: Linear Quadratic Methods*. Prentice Hall, 1990.

[3] D. Bulterman, G. Grassel, J. Jansen, A. Koivisto, N. Layaïda, T. Michel, S. Mullender, and D. Zucker. Synchronized multimedia integration language (SMIL 2.1). Recommendation REC-SMIL2-20051213, W3C, December 2005. http://www.w3.org/AudioVideo/.

[4] S. W. Carter, D. D. E. Long, and J.-F. Pâris. "Video-on-Demand Broadcasting Protocols." In J. D. Gibson, editor, *Multimedia Communications: Directions and Innovations*, chapter 11, pages 179–190. Academic Press, 2001.

[5] V. Chande and N. Farvardin. "Progressive Transmission of Images over Memoryless Noisy Channels," *IEEE J. Selected Areas in Communications*, 18(6):850–860, June 2000.

[6] P. A. Chou and Z. Miao. "Rate-Distortion Optimized Streaming of Packetized Media," *IEEE Trans. Multimedia*, 8(2):390–404, April 2006.

[7] P. A. Chou, A. E. Mohr, A. Wang, and S. Mehrotra. "Error Control for Receiver-Driven Layered Multicast of Audio and Video," *IEEE Trans. Multimedia*, 3(1):108–122, March 2001.

[8] P. A. Chou and K. Ramchandran. "Clustering Source/Channel Rate Allocations for Receiver-Driven Multicast with Error Control under a Limited Number of Streams," in *Proc. Int'l Conf. Multimedia and Exhibition*, volume 3, pages 1221–1224, New York, NY, July 2000.

[9] P. A. Chou, H. J. Wang, and V. N. Padmanabhan. "Layered Multiple Description Coding," in *Proc. Int'l Packet Video Workshop*, Nantes, France, April 2003.

[10] Apple Computer Corporation. QuickTime reference library. http://developer.apple.com/referencelibrary/QuickTime.

[11] Microsoft Corporation. Advanced systems format (ASF) specification. http://www.microsoft.com/windows/windowsmedia/forpros/format/asfspec.aspx.

[12] J. Crowcroft, M. Handley, and I. Wakeman. *Internetworking Multimedia*. UCL Press, December 1998. http://www.cs.ucl.ac.uk/staff/jon/mmbook/book/book.html.

[13] S. Floyd, M. Handley, J. Padhye, and J. Widmer. "Equation-Based Congestion Control for Unicast Applications," in *Proc. Data Communication, Ann. Conf. Series (SIGCOMM)*, Stockholm, Sweden, August 2000. ACM.

[14] S. Floyd and E. Kohler. Profile for datagram congestion control protocol (DCCP) congestion control ID 2: TCP-like congestion control. Proposed standard RFC 4341, IETF, http://www.ietf.org/rfc/rfc4341, March 2006.

[15] S. Floyd, E. Kohler, and J. Padhye. Profile for datagram congestion control protocol (DCCP) congestion control ID 3: TCP-friendly rate control (TFRC). Proposed standard RFC 4342, IETF, http://www.ietf.org/rfc/rfc4342, March 2006.

[16] G. Franklin, J. Powell, and M. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley, 3rd edition, 1997.

[17] T. Gill and B. Birney. *Microsoft Windows Media Resource Kit*. Microsoft Press, February 2003.

[18] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP friendly rate control (TFRC): Protocol specification. Proposed standard RFC 3448, IETF, http://www.ietf.org/rfc/rfc3448, January 2003.

[19] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. Proposed standard RFC 4566, IETF, http://www.ietf.org/rfc/rfc4566, July 2006. Obsoletes RFC2327.

[20] M. Handley, C. Perkins, and E. Whelan. Session announcement protocol. Experimental RFC 2974, IETF, http://www.ietf.org/rfc/rfc2974, October 2000.

[21] C.-Y. Hsu, A. Ortega, and A. Reibman. "Joint Selection of Source and Channel Rate for VBR Video Transmission under ATM Policing Constraints," *IEEE Journal on Selected Areas in Communications*, 15(5):1016–1028, August 1997.

[22] C. Huang, P. A. Chou, and A. Klemets. "Optimal Coding Rate Control for Scalable Streaming Media," in *Proc. Int'l Packet Video Workshop*, Irvine, CA, December 2004. IEEE.

[23] C. Huang, P. A. Chou, and A. Klemets. "Optimal Control of Multiple Bit Rates for Streaming Media," in *Proc. Picture Coding Symposium*, San Francisco, CA, December 2004.

[24] C. Huang, P. A. Chou, and A. Klemets. "Optimal Coding Rate Control of Scalable and Multi Bit Rate Streaming Media." Technical Report MSR-TR-2005-47, Microsoft Research, Redmond, WA, April 2005.

[25] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). Proposed standard RFC 4340, IETF, http://www.ietf.org/rfc/rfc4340, March 2006.

[26] G. Leibl, T. Stockhammer, M. Wagner, J. Pandel, G. Baese, M. Nguyen, and F. Burkert. An RTP payload format for erasure-resilient transmission of progressive multimedia streams. Internet Draft draft-ieft-avt-uxp-00.txt, IETF, February 2001. Expired.

[27] S. Lin and D. J. Costello, Jr. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1984.

[28] S. R. McCanne. *Scalable Compression and Transmission of Internet Multicast Video*. Ph.D. thesis, The University of California, Berkeley, CA, December 1996.

[29] S. R. McCanne, V. Jacobson, and M. Vetterli. "Receiver-Driven Layered Multicast," in *Proc. SIGCOM*, pages 117–130, Stanford, CA, August 1996. ACM.

[30] NullSoft, Inc. Shoutcast. http://www.shoutcast.com, 1999.

[31] S. Paul. *Multicasting on the Internet and Its Applications*. Kluwer, 1998.

[32] C. Perkins. *RTP: Audio and Video for the Internet*. Addison-Wesley, June 2003.

[33] RealNetworks, Inc. Content production and authoring documentation. http://service.real.com/help/library/encoders.html.

[34] J. Ribas-Corbera, P. A. Chou, and S. Regunathan. "A Generalized Hypothetical Reference Decoder for H.264/AVC," *IEEE Trans. Circuits and Systems for Video Technology*, 13(7), July 2003.

[35] H. Schulzrinne. RTP profile for audio and video conferences with minimal control. Standard RFC 3551, IETF, http://www.ietf.org/rfc/rfc3551, July 2003. Obsoletes RFC1890.

[36] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. Standard RFC 3550, IETF, http://www.ietf.org/rfc/rfc3550, July 2003. Obsoletes RFC1889.

[37] H. Schulzrinne, A. Rao, and R. Lanphier. Real time streaming protocol (RTSP). Proposed standard RFC 2326, IETF, http://www.ietf.org/rfc/rfc2326, April 1998.

[38] D. Singer, W. Belkap, and G. Franceschini. ISO media file format specification. Technical report, ISO/IEC JTC1/SC29/WG11 MPEG01/N4270-1, 2001.

[39] V. Stanković, R. Hamzaoui, and Z. Xiong. "Robust Layered Multiple Description Coding of Scalable Media Data for Multicast," *IEEE Signal Processing Letters*, 12:154–157, February 2005.

[40] W.-T. Tan and A. Zakhor. "Video Multicast Using Layered FEC and Scalable Compression," *IEEE Trans. Circuits and Systems for Video Technology*, 11(3):373–387, March 2001.

[41] Wikipedia. Coolstreaming. http://en.wikipedia.org/wiki/CoolStreaming.

[42] Wikipedia. PPLive. http://en.wikipedia.org/wiki/PPLive.

This page intentionally left blank

15

Real-Time Communication: Internet Protocol Voice and Video Telephony and Teleconferencing

Yi Liang, Yen-Chi Lee, and Andy Teng

15.1 INTRODUCTION

Internet Protocol (IP)-based real-time communication, including voice-over IP (VoIP), video telephony, and teleconferencing, has been gaining popularity in recent years. One example is VoIP, which has been competing with the traditional public switched telephone network (PSTN) for years and now enjoys increased market share. This is due to the many advantages of IP-based communication, including lower cost as well as the capability of providing integrated data, voice, and video, a larger variety of features, and more value-added services.

Despite rapid expansion and improvement of the underlying infrastructure, quality-of-service (QoS) is still one of the major challenges of real-time communication over IP networks. The unreliable and stateless nature of today's Internet protocol results in a best-effort service, that is, packets may be delivered with an arbitrary delay or may even be lost. Transmitted over the best-effort network and suffering from variable throughput, delay, and loss, data packets have to be delivered by a deadline to become useful. Excessive delay severely impairs communication interactivity; packet loss results in glitches in audio and poor picture quality and frozen frames in video. The heterogeneity of today's Internet also poses a major challenge for media delivery to users with various connection speeds, where scalability is highly desirable.

The challenges that the industry faces, in conjunction with the commercial promise of the technology, have attracted considerable effort in research and product development. In this chapter, we will first describe an architecture for real-time communication, followed by topics on how to improve the QoS. In Section 15.2, we will describe the basic system architecture as well as two categories of the most important protocols: signaling and transport. In Section 15.3, we will address QoS issues, especially minimizing latency, combating loss, adapting to available bandwidth, and audio–video synchronization.

15.2 ARCHITECTURE AND FUNDAMENTALS

15.2.1 Systems

Figure 15.1 shows the setup of a typical VoIP system. An IP phone or a sufficiently equipped PC connects to the Internet to be able to make VoIP calls. For traditional phones in the PSTN network, a gateway is needed for the interoperation between the PSTN and the Internet. After introducing signaling and transport protocols later in this section, we will illustrate the process of setting up a call using corresponding protocols in more detail (Figure 15.8).

Figure 15.2 shows a typical architecture for real-time audio and video communication over IP networks. Transport protocols, including UDP, TCP, and Real-Time Transport (RTP)/Real-Time Transport Control Protocols (RTCP) are built on top of the IP layer. Audio and video codecs are applied on the content encapsulated or to be encapsulated in the payload. The upper-layer applications call audio and video codecs to perform data compression. Signaling protocols, such as session initiation protocol (SIP), are used for call setup and control. The signaling and transport protocols are described in more detail in the following sections.

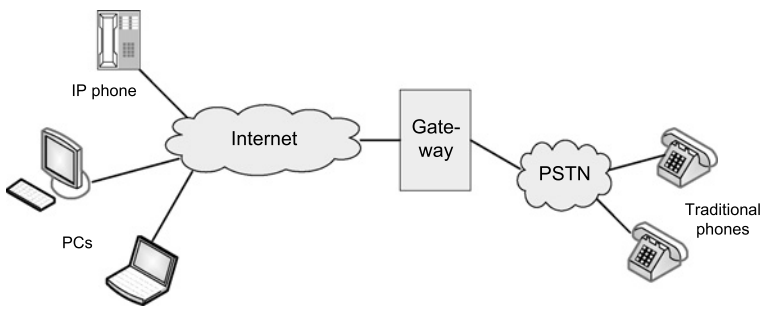


FIGURE 15.1: A typical configuration for a VoIP system with both IP-based devices and traditional PSTN phones.

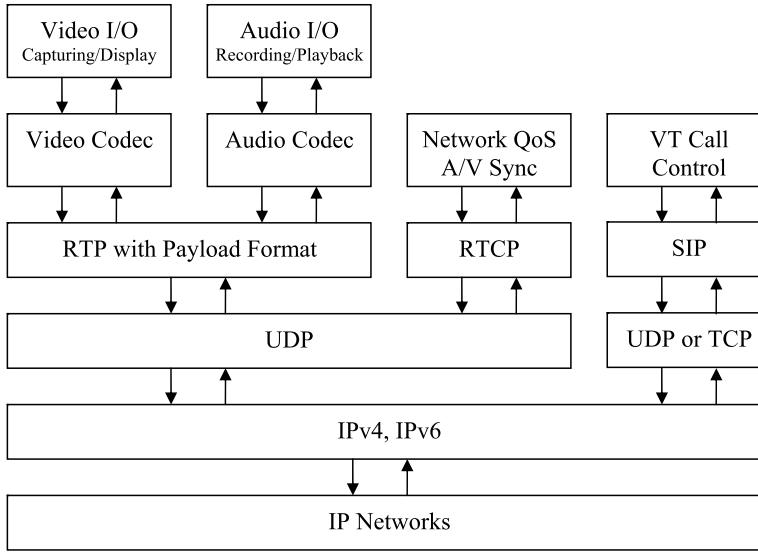


FIGURE 15.2: A typical architecture for an IP-based video and audio communication system.

15.2.2 Signaling Protocols

The SIP, originally developed by the IETF Multi-Party Multimedia Session Control Working Group, is the most widely used signaling protocol for real-time conversational applications over IP networks. As a signaling protocol, SIP provides the following functions:

- Call setup and tear down;
- Advanced features, such as call hold, call waiting, call forwarding, and call transfer;
- Capability exchange;
- Interoperability between different types of networks (e.g., PSTN) and different signaling protocols (e.g., H.323);
- Multicasting.

Moreover, SIP has been designed to be scalable enough to support simultaneous calls for a substantial number of users and to be extensible enough to include more features and functions in the future.

SIP may be transported by either TCP or UDP. TCP provides a reliable, connection-oriented transport, while UDP provides a best-effort, connectionless transport across the Internet. Port numbers 5060 and 5061 are the default ports

for SIP, although any number above 49172 may be used. The protocol stack for SIP-based IP phone service is shown in Figure 15.3.

There are two types of SIP messages: request and response. The request message is initiated by a user agent client (UAC) for registering, call setup, tear down, acknowledgment, etc., while the response message is generated by a user agent server (UAS) or a SIP server in response to the request.

The request message in SIP, as with the other IETF protocols (e.g., RTSP), is called a "method." There are six fundamental SIP methods considered as basic signaling for call setup and tear down, which are defined in IETF RFC 3261 [1]: INVITE, ACK, BYE, REGISTER, CANCEL, and OPTIONS. Specifically, the INVITE method is used to initiate a call. The ACK method is used by the call originator to acknowledge the final response to the INVITE request. The BYE method is used to terminate a call. The REGISTER method is used by a user agent (UA) to register itself to a SIP server with the addressing information (contact URI). The CANCEL method is used to cancel the request sent earlier. The OPTIONS method is used to query a SIP server/client capability. In addition to the six methods defined in RFC 3261, other methods were added later as SIP extensions and specified in different RFCs. Examples include INFO (RFC 2976 [2]), MESSAGE (RFC 3428 [3]), NOTIFY (RFC 3265 [4]), PRACK (RFC 3262 [5]), REFER (RFC 3515 [6]), SUBSCRIBE (RFC 3265 [4]), and UPDATE (RFC 3311 [7]).

The response message is called the "response code" in SIP. The SIP response codes are inherited from HTTP/1.1, except for the 6xx class, which is defined by SIP itself (RFC 3261). The six classes of SIP response codes are described briefly here.

> **1xx (provisional response)**: information to indicate current status before a definitive response. The 1xx response is designed such that an ACK is never triggered by it and thus the reliability for 1xx transmission is not critical. `180 Ringing` is an example of a 1xx response, which is used to inform the originator that the UA has already received the INVITE request.

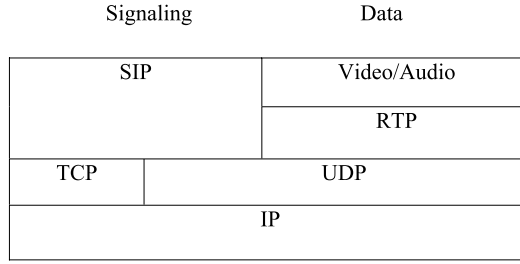


FIGURE 15.3: Protocol stack (signaling flow and data flow).

2xx (successful response): a response used to indicate that the request has been successfully received. Example: `200 OK`.
3xx (redirectional response): information used to indicate the user's new location or alternative services.
4xx (request failure): a response used by a UAS or a server to indicate that the request cannot be processed due to authorization failure, authentication failure, account issue, requesting itself, or other problems not related to the server. Example: `400 Bad Request` indicates that the server does not understand the request.
5xx (server failure): a response used by a UAS or a server to indicate that the request cannot be processed due to the server's problem. Examples include `500 Server Internal Error` and `501 Not Implemented`.
6xx (global failure): a response used to indicate that the response will fail in all locations and thus the request should not be delivered.

A simple message flow for call setup and tear down is illustrated in Figure 15.4.

The SIP request message is composed as follows: method name (e.g., INVITE), address, header fields, and message body. Each response message consists of a

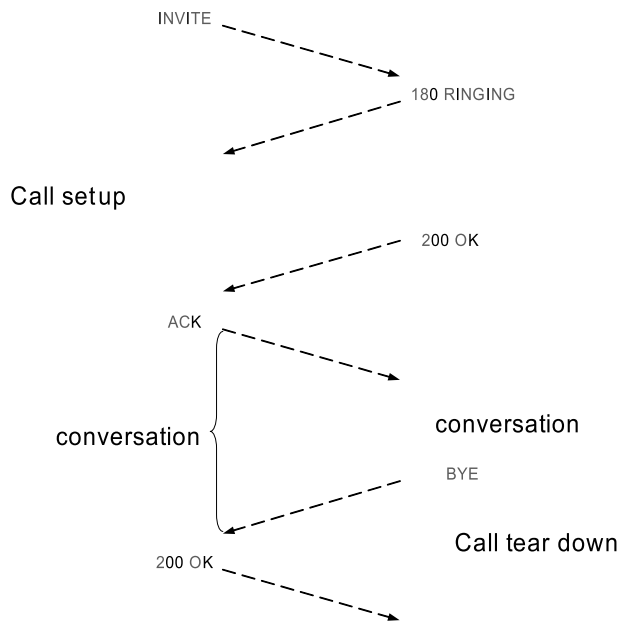


FIGURE 15.4: Simple message flow for call setup and tear down.

code (e.g., 200 OK), header fields, and message body. Note that the header fields and the message body may not appear in all messages.

15.2.2.1 Address

SIP supports a variety of addressing schemes, including SIP URI (Uniform Resource Identifiers), secure SIP URI, telephone URI, and e-mail URL (Uniform Resource Locator). SIP URI is usually represented as `sip:<userinfo>@<host>:<port>`.

15.2.2.2 Header Fields

A header field is composed as `<header>:<field>`. There are 44 header fields defined in RFC 3261: Accept, Accept-Encoding, Accept-Language, Alert-Info, Allow, Authentication-Info, Authorization, Call-ID, Call-Info, Contact, Content-Disposition, Content-Encoding, Content-Language, Content-Length, Content-Type, CSeq, Date, Error-Info, Expires, From, In-Reply-To, Max-Forwards, Min-Expires, MIME-Version, Organization, Priority, Proxy-Authenticate, Proxy-Authorization, Proxy-Require, Record-Route, Reply-To, Require, Retry-After, Route, Server, Subject, Supported, Timestamp, To, Unsupported, User-Agent, Via, Warning, and WWW-Authenticate. The most common ones are introduced here:

Call-ID: used to uniquely identify a call. Example: `CALL-ID: t315fde3-68te-33uyr@test.com`
Contact: used to carry a URI that identifies the resource requested or the request originator. Example: `Contact: sip:johnsmith@test.com`
CSeq: a decimal number used to uniquely identify a request. All responses corresponding to a request use the same CSeq as the request. The CSeq number is usually increased by one for a new request.
From: used to specify the originator. Example: `From: "John Smith" <sip:johnsmith@test.com>`
Max-Forwards: an integer in the range of 0–255 used to specify the maximum number of hops that a message can take. The recommended initial value is 70. It is decreased by one as the message passes through a proxy or gateway. The proxy/gateway discards the message when the value is dropped to zero.
To: used to specify the recipient of the request.
Via: used to record the path the request has been traveled. The response walks through the same path in the reverse order.

The mandatory headers for the six fundamental requests are shown in Table 15.1.

Table 15.1: Mandatory header fields for six fundamental SIP methods, where M denotes *mandatory*.

Header/Requests	INVITE	ACK	BYE	REGISTER	CANCEL	OPTIONS
Call-ID	M	M	M	M	M	M
Contact	M					
CSeq	M	M	M	M	M	M
From	M	M	M	M	M	M
Max-Forwards	M	M	M	M	M	M
To	M	M	M	M	M	M
Via	M	M	M	M	M	M

15.2.2.3 Message Body

Although any format can be used as a message body, the Session Description Protocol (SDP) [8] is the most popular one. SDP specifies media information such as media type, codec, author, title, encryption key, bandwidth, start time, and end time. SDP can be used for capability exchange at the call set-up stage. An example of SDP message is shown here

```
m = audio 49170 RTP/AVP 102
a = rtpmap:102 AMR/8000
a = fmtp:102 maxptime=60; octect-align=1; mode-set=4
m = video 49350 RTP/AVP 110
a = rtpmap:110 MP4V-ES/90000
a = fmtp:110 profile-level-id=0; config=000001B.....
```

The SDP message just given specifies the following information. Audio is transported by the RTP/AVP protocol through port 49170, with payload number 102. The RTP timestamp resolution is 1/8000 s. The audio is coded by AMR with the maximum bit rate of 7.4 kbps. Three-frame bundling is used, three audio frames are bundled together to form an RTP packet. Audio packetization should follow the rules defined in RFC 3267 [9]. Video is transported by the RTP/AVP protocol through port 49350, with payload number 110. The RTP timestamp resolution is 1/90,000 s. The video is coded by MPEG-4 video SVP L0 (simple visual profile level 0). Video packetization should follow the rules defined in RFC 3016 [10].

SIP is an IP telephony signaling protocol developed by the IETF, which competes with the H.323 protocol developed by the ITU-T for the same application. The fundamental difference between the two protocols is that SIP is a text-based protocol and inherits the rich set of the IETF protocols, such as SDP, whereas H.323 is binary encoded and utilizes many features from other ITU-T protocols, for example, H.245. Comparisons between the two protocols on features,

Table 15.2: Comparisons between SIP and H.323.

Comparisons	SIP	H.323
Encoding method	Text	Binary
Family	IETF	ITU-T
Transport	TCP or UDP	TCP
Packet loss recovery	Through SIP itself	Through TCP
Capability exchange	SDP (simple but limited)	H.245 (rich but complicated)
Security	Through other IETF; protocols for encryption, authentication, etc.	Not very good
Features	Call holding, call transfer, call forwarding, call waiting, conferencing, instant messaging	Call holding, call transfer, call forwarding, call waiting, conferencing

packet loss recovery, security mechanism, and capability exchange are listed in Table 15.2. A more detailed comparison can be found in [11].

15.2.3 Media Transport and Control Protocols

The commonly used media transport and control protocols in IP voice and video telephony applications are RTP and RTCP, as defined in RFC 3550. RTP and RTCP are designed to be independent of the underlying transport and network layers. Applications usually run RTP and RTCP on top of UDP and IP, as shown in Figure 15.3.

15.2.3.1 Real-Time Transport Protocols

RTP provides end-to-end delivery services for media data that have real-time characteristics. It defines useful information such as timestamp, sequence number, and marker, to allow receivers to keep the order of the packets, and to play out media at the proper pace. This is due to the fact that IP networks often introduce jitter in packets' arrival time and sometimes packet reordering. RTP itself, however, does not provide any mechanism to ensure timely delivery or to provide another quality of service.

Figure 15.5 shows the format of an RTP packet and its RTP header. Typically, in one-to-one telephony applications, the size of the RTP header is 12 bytes (no CSRC). V is a 2-bit field that identifies the version of the RTP. P is 1-bit information used to indicate if there are any padding octets at the end that are not part of the payload. X is a 1-bit field used to tell if there is any header extension information. CC means CSRC count, which uses 4 bits in the header and contains the number of CSRC identifiers that follow the header with fixed size. If an RTP session is one to one, such as in a video telephony application, the CSRC count should be set to zero. PT indicates payload type in 7 bits. It tells the format of the payload that an RTP packet carries.

M is a 1-bit marker and its interpretation is defined by a profile or payload format. For example, RFC 3016 is the payload format used for MPEG-4 audio and video. It specifies that if an encoded video frame is carried in multiple RTP packets, the marker bit of the last packet should be set to one to indicate the end of the frame. This is particularly useful for the RTP receiver to signal the video decoder to decode a video frame as soon as the last packet arrives.

SN specifies the sequence number of the RTP packet. It increases by one when one RTP data packet is sent. The initial sequence number of the first RTP packet for an RTP session should be randomly generated. For different media, the initial value may be different. For real-time telephony applications, the receiver can use the sequence numbers to detect any lost packets.

The timestamp TS reflects the sampling time of the first octet in the RTP packet payload. The sampling time should be calculated from a clock that increases monotonically and linearly in time to allow synchronization and jitter calculations. The timestamp may increase at a different pace for different media. For example, speech data are usually sampled at 8000 Hz and each speech frame can typically have 160 samples. Each RTP packet for speech will have a timestamp increment of 160. For video data sampled at 15 frames per second, the timestamp increment is 6000, based on a 90,000-Hz clock. If an encoded video frame is packetized into several RTP packets, each RTP packet will have the same timestamp as the data in each RTP packet are sampled at the same time instant.

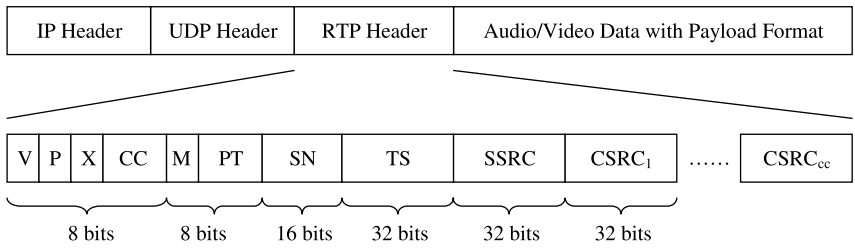


FIGURE 15.5: RTP header format.

A timestamp is particularly useful for media playout control at the receiver. The IP networks usually introduce packet interarrival jitter. In addition, for video encoding, it is possible that a video frame will be skipped in order to maintain a predefined fixed encoding bit rate. By looking at the timestamp, the receiver can properly play out the media at the pace when they were originally sampled. Timestamp information can also be used to synchronize the playout of different media, such as audio and video, with the help of RTCP. We will describe audio and video synchronization in more detail in Section 15.3.4.

SSRC specifies the synchronization source and has 32 bits. RTP packets generated from the same source, such as a camera or microphone, should have the same SSRC. SSRC can be used to help the receiver group RTP packets of the same media for playback. CSRC is also a 32-bit field. It indicates the source of a stream of RTP packets that have contributed to the combined stream produced by an RTP mixer. For one-to-one video telephony and VoIP applications, there is no CSRC present in the RTP header.

15.2.3.2 Real-Time Control Protocols

RTCP is used in conjunction with RTP to allow RTP session participants to monitor the quality of data delivery. It is based on the periodic transmission of control packets. There are five control packets defined in RFC 3550:

- SR: Send Report. This is sent by an RTP participant that sends and receives the RTP packets;
- RR: Receiver Report. This is sent by an RTP participant that only receives the RTP packets;
- SDES: Source DEScription, including CNAME;
- BYE: This is to indicate the end of the RTP participation;
- APP: Application-specific functions.

Both SR and RR control packets contain reception statistics such as interarrival jitter and packet loss rate. Each SR control packet further includes the sender's wallclock time and the corresponding RTP timestamp when it is generated, as well as transmission statistics, such as how many packets and bytes have been transmitted since the beginning of the RTP session. SR control packets can also be used to synchronize the playout of different media data.

SR and RR reports are also often used for flow and congestion control. For example, by analyzing the interarrival jitter field of the sender report, we can measure the jitter over a certain interval and indicate congestion. As defined in RFC 3550, when Packet i is received, the interarrival Jitter $J(i)$ is calculated as

$$J(i) = J(i-1) + \frac{|D(i-1,i)| - J(i-1)}{16}, \tag{15.1}$$

where

$$D(i-1,i) = \big(R(i) - R(i-1)\big) - \big(TS(i) - TS(i-1)\big), \qquad (15.2)$$

and $R(i)$ and $TS(i)$ are the arrival time and the timestamp of Packet i, respectively. Both are in RTP timestamp units. It is up to the implementation to decide what action to take when congestion occurs. A typical solution is to reduce the transmission rate until congestion becomes alleviated.

The round-trip time can also be estimated using last SR timestamp (LSR) and the delay since last SR (DLSR) information in both RR and SR control packets. Figure 15.6 demonstrates one example of a round-trip time calculation. Assume that the RTP sender sends one SR packet at time 10:20:30.250. The RTP receiver receives this SR and, after 5 s, sends an RR packet. In the RR control packet, LSR is the timestamp in SR(i) and the DLSR is 5 s. When the RTP sender receives this RR packet at time 10:20:36.750, it can calculate the round-trip time by subtracting the sending time of SR(i) and the DLSR from the arrival time of RR(i), which is 1.5 s as shown in Figure 15.6.

The fraction of loss in SR and RR control packets can also be used for the video encoder to perform error control. The packet loss rate is defined as the number of packets lost over the total number of received packets since the last SR or RR packet was sent.

The transmission interval of RTCP packets is often specified in proportion to the session bandwidth. It is recommended that the fraction of the session bandwidth added for RTCP be fixed at 5%. Some applications may specify the minimal transmission interval to be, for example, 5 s.

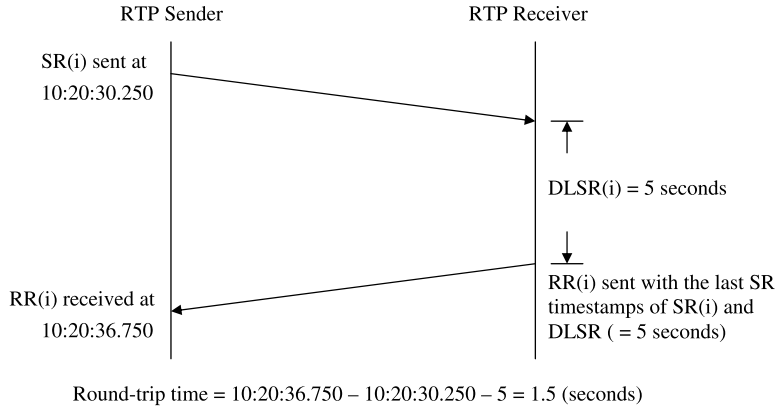


FIGURE 15.6: An example of a round-trip time calculation.

15.2.3.3 Video Payload Format

The purposes of using a video payload format are to specify an efficient way to encapsulate data to form a standard-compliant bit stream and to enhance the resilience against packet losses. The payload here means media data that are packed in an RTP packet. Forming the media payload can be done in a thin layer between the media encoder and the RTP transport layer. Currently, payload formats defined in RFC 3016 and RFC 2429 for encapsulating MPEG-4 and H.263 video data into individual packets are most commonly used.

The RTP payload formats are designed such that (i) a payload format should be devised so that the stream being transported is still useful even in the presence of a moderate amount of packet loss and (ii) ideally, each packet should possibly be decoded and played out irrespective of whether the preceding packets have been lost or arrived late.

Figure 15.7 shows examples of RTP packets generated for MPEG-4 video based on RFC 3016. Among these examples, Figure 15.7(b) shows one of the most commonly used packetization methods that have the best error-resilience capability. With this packetization method, one RTP packet contains one video packet. A video packet contains resynchronization marker information at the beginning of the video payload. When the RTP packet containing the VOP header is lost, the other RTP packets can still be decoded due to the use of the Header Extension Code information in the video packet header. No extra RTP header field is necessary.

For H.263 video, similar to MPEG-4 video described in Figure 15.7(b), RFC 2429 specifies that the PSC and slice header have to be at the beginning of each RTP packet. It also specifies that the picture header information can be repeated in

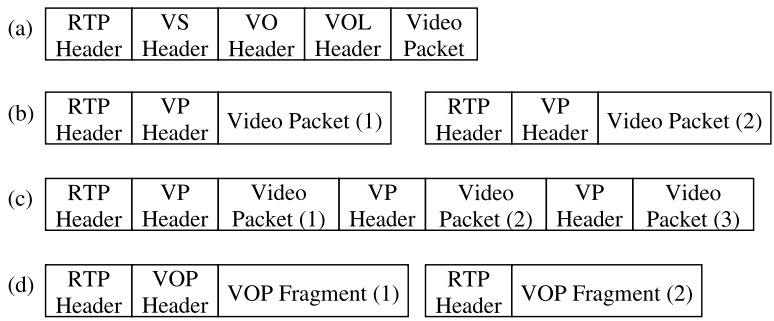


FIGURE 15.7: Examples of MPEG-4 video packetization based on RFC 3016 payload format. VS, visual object sequence; VO, visual object; VOL, visual object layer; VP, video packet; VOP, visual object plane.

each RTP packet. This can significantly reduce the number of frames that cannot be decoded due to picture header corruption. H.263 Annex W also provides a similar header protection mechanism, but this repeated header information can only be embedded once in the current picture header or the one in the previous or next frame. Thus, it has lower error resilience and may introduce delay due to waiting for the next frame.

Another purpose of using payload format is for interoperation between two video telephony users that use different applications. A certain video payload format for different codecs has to be supported and implemented to provide a unified video payload encapsulation.

15.2.3.4 An Example of a Call Setup Process

Before moving to the next section, we provide an example and illustrate the process of setting up a call using SIP. As illustrated in Figure 15.8, the caller PC, which is a SIP user agent, initiates a call by sending an INVITE request to the called party. The message has to go through the SIP server that serves the domain of the called party. The SIP server is responsible for locating the addressee via a location service and routing the message to the called party. Once the called party receives the INVITE request, it responses with `200 OK`, which is sent back to the caller. Then the caller sends an `ACK` directly to the called party, so that the call is set up, and an RTP pipe is established for audio and video transmission.

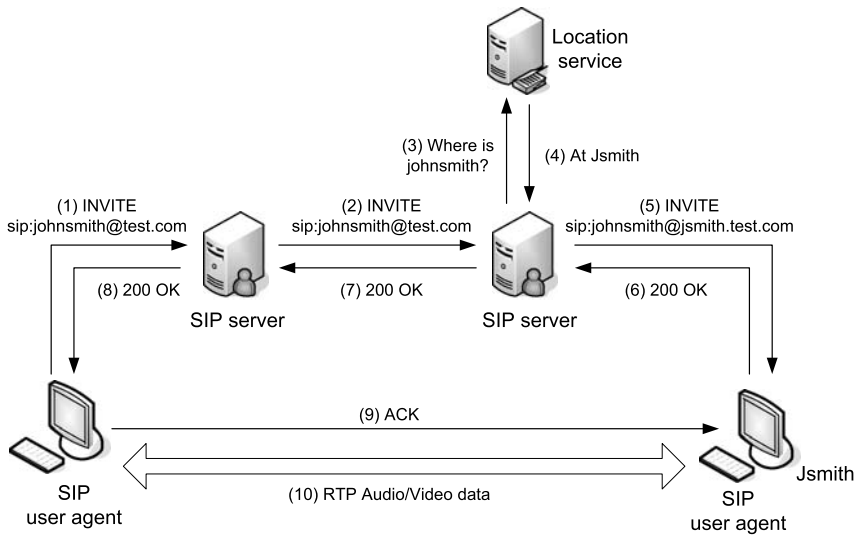


FIGURE 15.8: A call setup process using SIP.

SIP, being a signaling protocol, is only responsible for initiating and establishing the session, but the actual communication is directly between the caller and the called party.

15.3 QUALITY OF SERVICE

15.3.1 Minimizing Latency

To achieve toll quality for real-time communication, it is typically required that the round-trip delay be lower than 300 ms. Many factors contribute to the packet delay in a real-time communication system. The total end-to-end delay, D, can be divided into the following components:

$$D = d_{\text{enc}} + d_{\text{pack}} + d_{\text{net}} + d_{\text{buf}} + d_{\text{dec}}, \tag{15.3}$$

where d_{enc} is the encoding delay, d_{pack} is the packetization delay, d_{net} is the delay introduced by the network, d_{buf} is the buffering delay, and d_{dec} is the decoding delay (Figure 15.9). To minimize the end-to-end latency, each delay component has to be minimized and trade-offs have to be considered in optimizing the overall system design.

Encoding delay is introduced during the data compression process. For speech coders, encoding delay usually includes the frame size and look-ahead delay. Look-ahead delay is the time spent in processing part of the next frame so that a correlation between successive frames can be exploited. Typically, more advanced codecs achieve higher compression efficiency at the cost of higher encoding delays. Decoding delay is introduced during the data decompression process. Table 15.3 lists the coding delays for some common speech coders.

Packetization delay is the time spent in collecting sufficient data frames to form the payload of an IP packet. Since the packet headers have a fixed size, a larger payload size reduces the header overhead and improves the transmission efficiency. However, due to the stringent latency requirement, the payload can

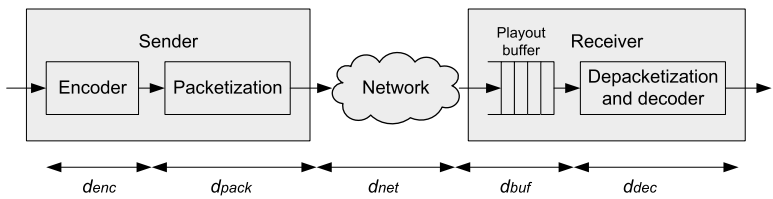


FIGURE 15.9: Total end-to-end delay in a typical real-time communication system.

Table 15.3: Coding delays for sample speech coders.

Speech coder	Encoded bit rate (kbit/s)	Frame size (ms)	Look-ahead delay (ms)	Decoding delay (ms)
G.711	64	0.125	0	0
G.729A	8	10	5	7.5
G.723.1	5.3/6.4	30	7.5	18.75

usually contain only a limited number of frames in order to reduce the packetization delay.

The network delay comprises the propagation delay and the queuing delay across all links in the transmission path. The propagation delay, which is a constant for a fixed path, depends on the packet size and the speed of links, as well as the length of the links. The queuing delay occurs when a packet is queued behind some other packets waiting to be transmitted over the same link. The queuing delay is a random variable depending on the packet size, traffic load and characteristics of the route, and the scheduling scheme. Advanced resource allocation and scheduling schemes such as Resource Reservation Protocol and Differentiated Services enable prioritization of audio and video packets and can efficiently reduce queuing delay for these real-time data streams.

Varying queuing delay, typically caused by congestions of links in the route and related to the queuing mechanisms, introduces delay jitter, which is usually unknown and random. Due to delay jitter, IP packets are sent periodically but are received in irregular patterns. For this reason, a playout buffer, also referred to as a dejitter buffer, is employed at the receiver to absorb the delay jitter before media are output. When using a playout buffer, packets are not played out immediately after being received but are held in a buffer until their scheduled playout time (playout deadline) arrives. Although this introduces additional delay for packets arriving early, this mechanism ensures continuous media playback. The buffering delay is the time a packet is held in the buffer before it is played out.

Note that a trade-off exists between the average buffering delay and the number of packets that have to be dropped because they arrive too late (late loss). Scheduling a later deadline increases the possibility of playing out more packets and results in a lower loss rate, but at the cost of a higher buffering delay. Vice versa, it is difficult to decrease the buffering delay without significantly increasing the loss rate. Therefore, packet loss in delay-sensitive, real-time applications is a result of not only a packet being dropped over the network, but also delay jitter, which impairs communication quality greatly.

Due to the aforementioned buffering delay—late loss rate trade-off—it is desirable to design smart playout scheduling mechanisms to reduce the buffering delay. Fixed scheduling poses a limitation for this trade-off. In real-time speech communication, more advanced mechanisms use a playout buffer to completely

absorb delay jitter within talkspurts and dynamically adjust the schedule between talkspurts [12–16]. Adaptive playout scheduling is proposed to allow adaptive schedules even within talkspurts [17], and this idea has also been extended to video streaming [18,19]. An adaptive playout schedule is able to reduce the latency and the effective packet loss rate at the same time. Interested readers may refer to Chapter 16 for more details.

15.3.2 Combating Losses

In real-time communications, losses are a result of not only packets dropping over the network, but also late arrival for packets. We introduce different loss-resilient techniques for both audio and video in two categories: client-side techniques and active techniques, depending on whether they require any encoder involvement.

15.3.2.1 Client-Side Techniques

One category of loss techniques is passive methods that are implemented at the client side, which do not require any cooperation of the sender or increase the cost of transmission. Client-side techniques impose low overhead for the communication system but can be highly efficient in enhancing the quality of the rendered media.

To combat channel losses, the client typically employs error-detection and loss-concealment techniques to mitigate the effect of lost data. For speech and audio coded techniques based on waveform, most client-side schemes take advantage of the data received adjacent to the lost packet and interpolate the missing information by exploiting the redundancy in the signal. In particular, waveform repetition simply repeats the information contained in the packets prior to the lost one [20, 21]. A more advanced loss-concealment technique using timescale modification is described in [22] and [23] and can be used in conjunction with adaptive playout scheduling in a low-latency scenario [17]. Waveform repetition typically does not introduce any algorithm delay as timescale modification typically does. However, it does not provide as good a sound quality [24]. Interested readers may further refer to Chapter 3 for error-resilient techniques for various codecs.

For video communication, postprocessing is typically applied at the client side for error concealment and loss recovery. Techniques to recover the damaged areas based on characteristics of image and video signals have been reviewed in [25]. Interested readers may further refer to Chapter 2 for more detailed descriptions on error-resilient video.

15.3.2.2 Active Techniques

A different category of error-resilience techniques requires the encoder to play a primary role. They are able to provide even higher robustness for media com-

munication over best-effort networks. We refer to these techniques as "active" to differentiate them from those only employed at the client side.

For speech communication, one widely accepted way to reduce the effective packet loss observed by the receiver is to add redundancy to the data stream at the sender. This is possible without imposing too much extra network load since the data rate of the voice traffic is very low when compared with other types of multimedia and data traffic. A common method to add redundancy is forward error correction (FEC), which transmits redundant information across packets, where loss recovery is performed at the cost of higher latency. The efficiency of FEC schemes is largely limited by the bursty nature of the channel losses. In order to combat burst losses, redundant information has to be added into temporally distant packets, which introduce higher delay.

Another sender-based loss recovery technique, interleaving, does not increase the data rate of transmission but still introduces delay at both encoder and decoder sides. The efficiency of loss recovery depends on over how many packets the source packet is interleaved and spread over. Again, the wider the spread, the higher the introduced delay. For low-latency speech communication, path diversity techniques, presented in [26] as well as in Chapter 17, have been demonstrated to be very powerful in combating losses.

Video communication typically requires much higher data transmission rates than audio. A variety of active schemes has been proposed not only to increase the robustness of communication, but also to take the data rate efficiency into consideration [27–29]. Many of the recent algorithms use rate–distortion (R–D) optimization techniques to improve the compression efficiency [30–32], as well as to improve the error-resilient performance over lossy networks [33,34]. The goal of the R–D optimization algorithms is to minimize the expected distortion due to both compression and channel losses subject to the bit-rate constraint.

One example of this area is Intra/Inter-mode switching [35–38], where Intra-coded macroblocks are updated according to the network condition to mitigate temporal error propagation. Another approach is to modify the temporal prediction dependency of motion-compensated video coding in order to mitigate or stop error propagation. Example implementations include reference picture selection [27,39–41] and NEWPRED in MPEG-4 [42,43], where channel feedback is used to efficiently stop error propagation due to any transmission error. Another example is video redundancy coding (VRC), where the video sequence is coded into independent threads (streams) in a round-robin fashion [27,44]. A Sync-frame is encoded by all threads at regular intervals to start a new thread series and stop error propagation. If one thread is damaged due to packet loss, the remaining threads can still be used to predict the Sync-frame. VRC provides improved error resilience, but at the cost of a much higher data rate. Dynamic control of the prediction dependency can also be used by employing long-term memory prediction to achieve improved R–D performance [33,45,46].

Typically a channel coding module in a robust video communication system may involve FEC and automatic retransmission on request (ARQ). Similar to their applications in speech communication, when FEC is employed across packets, missing packets can be recovered at the receiver as long as a sufficient number of packets is received [47–50]. FEC is widely used as an unequal error protection scheme to protect prioritized transmissions. In addition to FEC codes, data randomization and interleaving are also employed for enhanced protection [51–55].

ARQ techniques incorporate channel feedback and employ the retransmission of erroneous data [56–60]. Unlike FEC schemes, ARQ intrinsically adapts to the varying channel conditions and tends to be more efficient in transmission. However, for real-time communication and low-latency streaming, the latency introduced by ARQ is a major concern. In addition, like all feedback-based error control schemes, ARQ is not appropriate for multicasting.

15.3.3 Adapting to the Available Bandwidth

Due to the lack of a QoS guarantee over most commercially deployed networks, it is expected that the condition, as well as the available bandwidth of the network, varies during a real-time communication session. It is beneficial to employ bandwidth adaptation mechanisms to control the rate at which the media are transmitted. This helps avoid a potential penalty on overuse of bandwidth, which usually leads to quality degradation and even service interruption. Typical bandwidth adaptation techniques include rate control, transcoding, scalable coding, and bit stream switching. Readers may refer to Chapter 4 for details of various bandwidth adaptation techniques, and further refer to Chapters 5 and 6 for scalable coding for video and audio, respectively.

15.3.4 Audio–Video Synchronization

RTP timestamps from different media streams may advance at different rates and usually have independent and random offsets. Therefore, although these timestamps are sufficient to reconstruct the timing of a single stream, directly comparing RTP timestamps from different media is not effective for synchronization. Instead, for each medium the RTP timestamp is related to the sampling instant by pairing it with a timestamp from a reference clock (wallclock) that represents the time when the medium was sampled. The reference clock is shared by all media to be synchronized.

Synchronizing audio and video can be achieved by playing out audio and video according to their original sampled time. By doing so, the receiver can play back audio and video at a proper pace by mapping their original sampled time to the receiver's local time. RTCP SR control packets provide useful information to help

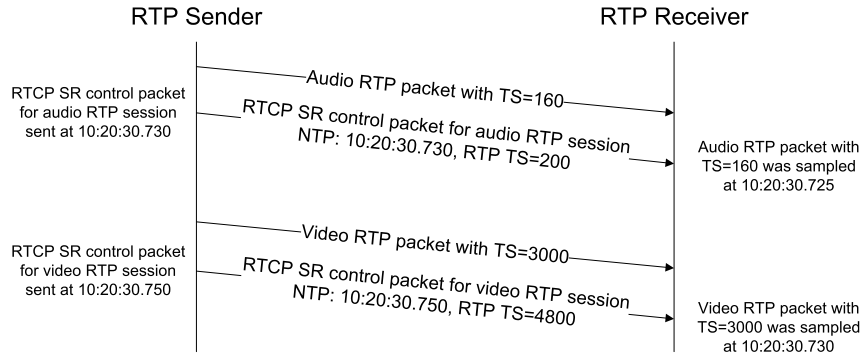


FIGURE 15.10: An example of audio and video synchronization.

the receiver calculate the sampled time of the audio and the video at the sender. Figure 15.10 illustrates an example of audio and video synchronization by using RTCP SR control packets. When an RTCP SR control packet is generated, it will carry the wallclock time (NTP) and the RTP timestamp using its corresponding media reference time. In Figure 15.10, the RTCP SR control packet for the audio RTP session is generated at time 10:20:30.730 and the corresponding timestamp is 200. When receiving this SR packet, the receiver is able to calculate when all the audio RTP packets are sampled at the sender. For the example in Figure 15.10, the audio packet with timestamp 160 is actually generated at time 10:20:30.725, assuming that a 8000-Hz clock is used for audio timestamping. Similarly, for video packets, the receiver can also calculate when each video frame is sampled. In this way, the receiver can easily find out which part of audio data and video data should be played back at the same time.

15.4 SUMMARY AND FURTHER READING

In this chapter, we have described the system and architecture for real-time communication, including two categories of the most important protocols, signaling and transport, respectively. We have also addressed the QoS issues, especially on minimizing latency, combating losses, adapting to available bandwidth, and audio–video synchronization. Beyond the references cited in this chapter, the reader is recommended to read Chapter 2, on error-resilient video, and Chapter 3, on error-resilient audio. Interested readers are further recommended to read Chapter 16, on adaptive media playout, as well as Chapter 17, on path diversity, for enhanced QoS performance.

REFERENCES

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, M. Hardley, and E. Schooler. "SIP: Session Initiation Protocol," *RFC 3261*, June 2002.

[2] S. Donovan. "The SIP INFO Method," *RFC 2976*, Oct. 2000.

[3] B. Campbell et al. "Session Initiation Protocol (SIP) Extension for Instant Messaging," *RFC 3428*, December 2002.

[4] A. B. Roach. "Session Initiation Protocol (SIP): Specific Event Notification," *RFC 3265*, June 2002.

[5] J. Rosenberg and H. Schulzrinne. "Reliability of Provisional Responses in the Session Initiation Protocol," *RFC 3262*, June 2002.

[6] R. Sparks. "The Session Initiation Protocol (SIP) Refer Method," *RFC 3515*, April 2003.

[7] J. Rosenberg. "The Session Initiation Protocol (SIP) UPDATE Method," *RFC 3311*, September 2002.

[8] M. Handley and V. Jacobson. "SDP: Session Description Protocol," *RFC 2327*, April 1998.

[9] S. Sjoberg, M. Westerlurd, A. Lakaniemi, and Q. Xie. "Real-Time Transport Protocol (RTP) Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codec," *RFC 3267*, June 2002.

[10] Y. Kikuchi, T. Nomura, S. Fukunaga, Y. Matsui, and H. Kimata. "RTP Payload Format for MPEG-4 Audio/Visual Streams," *RFC 3016*, November 2000.

[11] I. Dalgic and H. Fang. "Comparison of H.323 and SIP for IP Telephony Signaling," in *Proc. of Photonics East*, Boston, MA, September 1999.

[12] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne. "Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks," in *Proceedings IEEE INFOCOM '94*, vol. 2, pp. 680–688, June 1994.

[13] S. B. Moon, J. Kurose, and D. Towsley. "Packet Audio Playout Delay Adjustment: Performance Bounds and Algorithms," *Multimedia Systems*, vol. 6, no. 1, pp. 17–28, January 1998.

[14] J. Pinto and K. J. Christensen. "An Algorithm for Playout of Packet Voice Based on Adaptive Adjustment of Talkspurt Silence Periods," in *Proceedings 24th Conference on Local Computer Networks*, pp. 224–231, October 1999.

[15] P. DeLeon and C. J. Sreenan. "An Adaptive Predictor for Media Playout Buffering," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-99)*, vol. 6, pp. 3097–3100, March 1999.

[16] J. Rosenberg, L. Qiu, and H. Schulzrinne. "Integrating Packet FEC into Adaptive Voice Playout Buffer Algorithms on the Internet," in *Proceedings IEEE INFOCOM 2000*, vol. 3, pp. 1705–1714, Tel Aviv, Israel, March 2000.

[17] Y. J. Liang, N. Färber, and B. Girod. "Adaptive Playout Scheduling and Loss Concealment for Voice Communication over IP Networks," *IEEE Transactions on Multimedia*, vol. 5, no. 4, pp. 532–543, December 2003.

[18] E. Steinbach, N. Färber, and B. Girod. "Adaptive Playout for Low-Latency Video Streaming," in *IEEE International Conference on Image Processsing ICIP-01*, vol. 1, pp. 962–965, Thessaloniki, Greece, October 2001.

[19] M. Kalman, E. Steinbach, and B. Girod. "Adaptive Playout for Real-Time Media Streaming," in *IEEE International Symposium on Circuits and Systems*, vol. 1, pp. I–45–8, Scottsdale, AZ, May 2002.

[20] D. J. Goodman, G. B. Lockhart, O. J. Wasem, and W.-C. Wong. "Waveform Substitution Techniques for Recovering Missing Speech Segments in Packet Voice Communications," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 6, pp. 1440–1448, December 1986.

[21] O. J. Wasem, D. J. Goodman, C. A. Dvorak, and H. G. Page. "The Effect of Waveform Substitution on the Quality of PCM Packet Communications," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 3, pp. 342–348, March 1988.

[22] A. Stenger, K. Ben Younes, R. Reng, and B. Girod. "A New Error Concealment Technique for Audio Transmission with Packet Loss," in *Proc. European Signal Processing Conference*, vol. 3, pp. 1965–1968, September 1996.

[23] H. Sanneck, A. Stenger, K. Ben Younes, and B. Girod. "A New Technique for Audio Packet Loss Concealment," in *IEEE GLOBECOM*, pp. 48–52, November 1996.

[24] C. Perkins, O. Hodson, and V. Hardman. "A Survey of Packet Loss Recovery Techniques for Streaming Audio," *IEEE Network*, vol. 12, no. 5, pp. 40–48, September–October 1998.

[25] Yao Wang and Qin-Fan Zhu. "Error Control and Concealment for Video Communication: A Review," *Proceedings of the IEEE*, vol. 86, no. 5, pp. 974–997, May 1998.

[26] Y. J. Liang, E. G. Steinbach, and B. Girod. "Real-Time Voice Communication over the Internet Using Packet Path Diversity," in *Proceedings ACM Multimedia 2001*, pp. 431–440, Ottawa, Canada, October 2001.

[27] S. Wenger, G. D. Knorr, J. Ott, and F. Kossentini. "Error Resilience Support in H.263+," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 7, pp. 867–877, November 1998.

[28] R. Talluri. "Error-Resilient Video Coding in the ISO MPEG-4 Standard," *IEEE Communications Magazine*, pp. 112–119, June 1998.

[29] W. Tan and A. Zakhor. "Real-time Internet Video Using Error Resilient Scalable Compression and TCP-Friendly Transport Protocol," *IEEE Trans. Multimedia*, pp. 172–186, June 1999.

[30] G. J. Sullivan and T. Wiegand. "Rate-Distortion Optimization for Video Compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 74–90, November 1998.

[31] A. Ortega and K. Ramchandran. "From Rate-Distortion Theory to Commercial Image and Video Compression Technology," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 20–122, November 1998.

[32] T. Wiegand, X. Zhang, and B. Girod. "Long-Term Memory Motion-Compensated Prediction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 1, pp. 70–84, February 1999.

[33] T. Wiegand, N. Färber, and B. Girod. "Error-Resilient Video Transmission Using Long-Term Memory Motion-Compensated Prediction," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 6, pp. 1050–1062, June 2000.

[34] P. A. Chou, A. E. Mohr, A. Wang, and S. Mehrotra, "Error Control for Receiver-Driven Layered Multicast of Audio and Video," *IEEE Transactions on Multimedia*, vol. 3, no. 1, pp. 108–122, March 2001.

[35] J. Y. Liao and J. D. Villasenor. "Adaptive Intra Update for Video Coding over Noisy Channels," in *Proceedings IEEE International Conference on Image Processing*, Lausanne, Switzerland, vol. 3, pp. 763–766, September 1996.

[36] R. O. Hinds, T. N. Pappas, and J. S. Lim. "Joint Block-Based Video Source/Channel Coding for Packet-Switched Networks," in *Proceedings of the SPIE VCIP 98*, vol. 3309, pp. 124–133, San Jose, CA, October 1998.

[37] G. Cote and F. Kossentini. "Optimal Intra Coding of Blocks for Robust Video Communication over the Internet," *Signal Processing: Image Communication*, vol. 15, no. 1-2, pp. 25–34, September 1999.

[38] R. Zhang, S. L. Regunathan, and K. Rose. "Video Coding with Optimal Inter/Intra-mode Switching for Packet Loss Resilience," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 6, pp. 966–976, June 2000.

[39] S. Fukunaga, T. Nakai, and H. Inoue. "Error Resilient Video Coding by Dynamic Replacing of Reference Pictures," in *Proc. of the IEEE Global Telecommunications Conference*, vol. 3, pp. 1503–1508, London, UK, November 1996.

[40] ITU-T Recommendation H.263 Version 2 (H.263+), *Video Coding for Low Bitrate Communication*, January 1998.

[41] ITU-T Recommendation H.264, *Advanced Video Coding (AVC) for Generic Audiovisual Services*, May 2003.

[42] International Organisation for Standardisation, ISO/IEC JTC1/SC29/WG11 Final Committee Draft 14496-2, *Information Technology – Coding of Audio-Visual Objects: Visual (MPEG-4)*, March 1998.

[43] International Organisation for Standardisation, ISO/IEC JTC1/SC29/WG11 Final Committee Draft 14496-2, *Information Technology – Coding of Audio-Visual Objects: Visual (MPEG-4)*, March 1998.

[44] S. Wenger. "Video Redundancy Coding in H.263+," in *Proc. of the Workshop on Audio-Visual Services for Packet Networks*, September 1997.

[45] M. Budagavi and J.D Gibson. "Multiframe Video Coding for Improved Performance over Wireless Channels," *IEEE Transactions on Image Processing*, vol. 10, no. 2, pp. 252–265, February 2001.

[46] Y. J. Liang and B. Girod. "Network-Adaptive Low-Latency Video Communication over Best-Effort Networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 1, pp. 72–81, January 2006.

[47] Internet Engineering Task Force. "RTP Payload Format for MPEG-1/MPEG-2 Video," *RFC 2250*, January 1998.

[48] A. Albanese, J. Blömer, J. Edmonds, M. Luby, and M. Sudan. "Priority Encoding Transmission," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1737–1744, November 1996.

[49] P. C. Cosman, J. K. Rogers, P. G. Sherwood, and K. Zeger. "Image Transmission over Channels with Bit Errors and Packet Erasures," in *Proceedings of the Thirty-Second Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1621–1625, Pacific Grove, CA, November 1998.

[50] W. Tan and A. Zakhor. "Video Multicast Using Layered Fec and Scalable Compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 373–387, March 2001.

[51] J.-Y. Cochennec. *Method for the Correction of Cell Losses for Low Bit-Rate Signals Transport with the AAL Type 1*. ITU-T SG15 Doc. AVC-538, July 1993.

[52] Q.-F. Zhu, Y. Wang, and L. Shaw. "Coding and Cell-Loss Recovery in DCT-Based Packet Video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 3, no. 3, pp. 248–258, June 1993.

[53] T. Kinoshita, T. Nakahashi, and M. Maruyama, "Variable-Bit-Rate HDTV Codec with ATM-Cell-Loss Compensation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 3, no. 3, pp. 230–237, June 1993.

[54] K. Stuhlmüller, N. Färber, M. Link, and B. Girod. "Analysis of Video Transmission over Lossy Channels," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 6, pp. 1012–1032, June 2000.

[55] Y. J. Liang, J. G. Apostolopoulos, and B. Girod, "Model-Based Delay-Distortion Optimization for Video Streaming Using Packet Interleaving," in *Proceedings of the 36th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, November 2002.

[56] S. B. Wicker. *Error Control Systems for Digital Communication and Storage*, Prentice Hall, 1995.

[57] M. Khansari, A. Jalali, E. Dubois, and P. Mermelstein. "Low Bit-Rate Video Transmission over Fading Channels for Wireless Microcellular Systems," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 1, pp. 1–11, February 1996.

[58] B. Dempsey, J. Liebeherr, and A. Weaver. "On Retransmission-Based Error Control for Continuous Media Traffic in Packet-Switching Networks," *Computer Networks and ISDN Systems Journal*, vol. 28, no. 5, pp. 719–736, March 1996.

[59] C. Papadopoulos and G. M. Parulkar. "Retransmission-Based Error Control for Continuous Media Applications," in *Proc. Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Zushi, Japan, July 1996.

[60] H. Liu and M. El Zarki. "Performance of H.263 Video Transmission over Wireless Channels Using Hybrid ARQ," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 9, pp. 1775–1786, December 1999.

This page intentionally left blank

16

Adaptive Media Playout

Eckehard Steinbach, Yi Liang, Mark Kalman, and Bernd Girod

16.1 INTRODUCTION

This chapter discusses Adaptive Media Playout (AMP) as a method of reducing the user-perceived latencies that are inherent in systems that send packetized media over best-effort packet networks. These systems strive to allow the immediate display of media data as it is delivered from a remote sender. In practice, however, the systems must buffer an amount of media at the client to prevent packet losses and delays from constantly interrupting the playout of the stream. While the likelihood of a playout interruption decreases as more data is buffered, the delays that buffering introduces increase.

Adaptive media playout allows the client to buffer less data and, thus, introduces less delay to achieve a given playout reliability. In this scheme, the client varies the rate at which it plays out audio and video according to the state of its playout buffer. Generally, when the buffer occupancy is below a desired level, the client plays media slowly to reduce its data consumption rate. Faster-than-normal playout may be used during good channel periods to eliminate any excess latency accumulated with slowed playout. By manipulating playout speeds AMP can reduce initial buffering delays in the case of prestored streams and reduce the user-perceived latency of live streams, all without sacrificing playout reliability.

To control the playout speed of media, the client scales the duration that each video frame is shown and processes audio to scale it in time without affecting its pitch. Variations in the media playout rate are subjectively less irritating than playout interruptions and long delays. How much the media signal can be stretched or compressed depends on the application. In this chapter AMP is discussed using two popular applications. Internet Telephony (VoIP) is used as a representative

for bidirectional conversational applications with strict end-to-end delay requirements. Video streaming is selected as an application with comparatively relaxed end-to-end delay requirements.

This chapter is organized as follows. We first discuss the receiver buffer in combination with fixed playout as the traditional means of adapting the application to varying transmission characteristics. We then introduce AMP using Internet Telephony and Video Streaming as example applications. Next, algorithms for duration scaling of audio, speech, and video segments are discussed. Toward the end of the chapter we touch on advanced deployment of AMP in the context of multipath transmission.

16.2 SENDER AND RECEIVER CURVES

The end-to-end delay encountered when transmitting digital media signals over a packet switched or circuit switched network is the accumulation of various delay contributions (see also Chapter 15.3). End-to-end delay is considered to be the time difference between capturing the media signal at the sender and displaying the signal at the receiver. Figure 16.1 shows a block diagram of the individual steps involved. The sender sampling curve $p_s(t)$ describes the amount of media data (e.g., in bytes) captured up to a certain time instant t. Without loss of generality we can assume that $t = T_{\text{start}} = 0$ s is our starting point. The digital media signal may be fed into an encoder with the purpose of data compression. Typically multiple samples of the sender signal are compressed jointly, for example, a block of speech samples or a digital video frame.

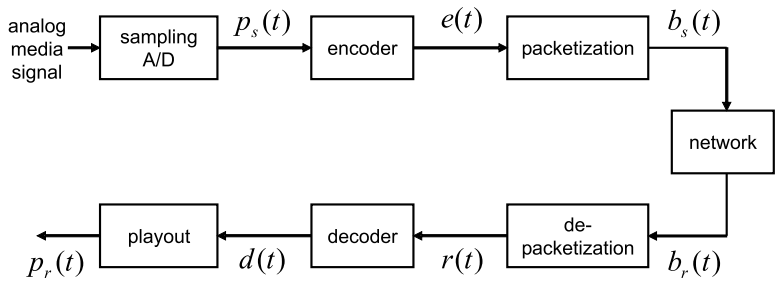


FIGURE 16.1: Processing and transmission of digital media signals. Each step introduces constant or variable delay, which accumulates to the total end-to-end delay. End-to-end delay is the time difference between capturing a media data sample at the sender side and displaying it at the receiver.

The encoder curve $e(t)$ describes the length of the bitstream output by the encoder up to time t. During packetization a certain number of data units is put into one data packet, which is then injected into the network. Packet size can be fixed or dynamically adapted. Once the encoder has output enough data units to fill the next packet we assume that the packet is immediately sent out on the network. The sender packet curve $b_s(t)$ describes the number of data units injected into the network up to time instant t. Depending on whether the network is circuit switched or packet switched, the delivery time, that is, the time it takes the packet to reach the receiver side, is either fixed or variable. The receiver packet curve $b_r(t)$ describes the amount of continuous packet payload data that has been received up to time instant t. Next, the payload is extracted from the packets and the depacketization curve $r(t)$ describes the input to the decoder. The output of the decoder is described by the decoder curve $d(t)$, which corresponds to the amount of decoded data ready for display. The playout process then decides which data to play at what time instant. The data played up to time t is described by the receiver playout curve $p_r(t)$.

The meaning of these various curves and their relationship can be best understood when looking at specific examples. The two examples discussed in the following are the transmission of a digital speech signal over a circuit-switched network and the transmission of a digital video signal over a packet-switched network. For additional discussion of buffering and timing fundamentals, see also Section 14.3.

Example 1: Sender and Receiver Curves for Transmission of a Digital Speech Signal

In this example the transmission of a digital speech signal over a circuit-switched network is considered. The digital speech signal is obtained by A/D conversion of an analog microphone signal. The sampling frequency is assumed to be $f_s = 8$ KHz and the signal amplitude resolution in our example is 8 bit/sample. Figure 16.2 illustrates possible sender and receiver curves. The sender sampling curve $p_s(t)$ is a straight line with slope 64 kbit/s. In this example it is assumed that the encoder does not perform data compression on the digital speech signal, which leads to $e(t) = p_s(t)$. The signal is partitioned into blocks or packets of 20 ms, which corresponds to a payload of 160 byte. The sender packet curve $b_s(t)$ hence becomes a step curve with step height 160 byte and step width 20 ms. The speech signal is transmitted over a circuit-switched network and the constant packet delivery time is assumed to be 100 ms. The receiver packet curve $b_r(t)$ in Figure 16.2 therefore becomes simply a shifted version of $b_s(t)$. Depacketization is assumed to be of negligible duration and decoding does not have to be performed for uncompressed data, which leads to $d(t) = r(t) = b_r(t)$. Playout is initiated by the receiver 20 ms after the arrival of the first packet. Playout is therefore started at $t = 140$ ms. This time instant is called the initial playout

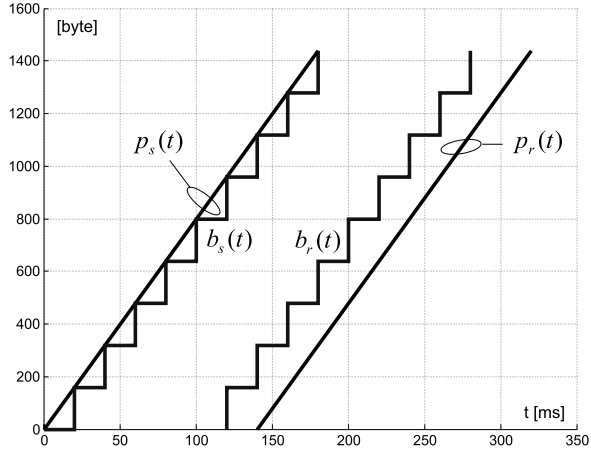


FIGURE 16.2: Example sender and receiver curves for transmission of a constant bit-rate signal (a digital speech signal with 64 kbit/s) over a circuit-switched network.

delay T_{initial}. The receiver playout curve $p_r(t)$ becomes a shifted version of the sender playout curve with a constant end-to-end delay of $T_{e2e} = 140$ ms.

Example 2: Sender and Receiver Curves for Transmission of a Digital Video Signal

For digital video we capture individual frames at a certain frame rate (e.g., 25 frames/second). After acquisition, the digital video frames are compressed and the encoder produces an encoder curve $e(t)$ that deviates from a uniform step curve as the output of the encoder is a variable bit rate stream (VBR stream). Let us consider a video sequence with a spatial resolution of 176×144 pixels, 25 frames/second, and an amplitude resolution of 12 bit/pixel. This leads to a raw data rate of $176 \times 144 \times 25 \times 12$ bit/s or 37.125 Kbyte/frame. The sender sampling curve $p_s(t)$ is a step curve with a step width of 40 ms, which corresponds to the inter-frame spacing. The step height corresponds to the frame size in bytes, in this example 37.125 Kbyte. After compression, every video frame has a different size, which leads to the varying step height of the encoder curve $e(t)$ shown in Figure 16.3. The video is transmitted over the Internet and for simplicity it is assumed that one encoded video frame is transmitted as the payload of one IP packet. This leads to variable size packets. The packetizer waits until the encoder outputs the encoded bit stream for a new video frame and then injects one packet into the network. Neglecting the packetization time leads to $b_s(t) = e(t)$. The packets are transmitted over a packet-switched network, which leads to a receiver

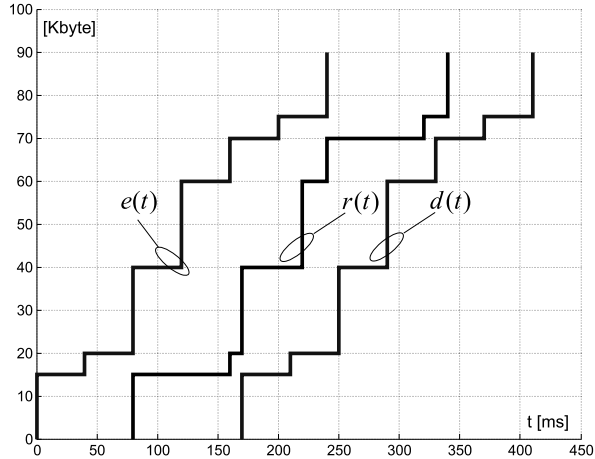


FIGURE 16.3: Example sender and receiver curves for the transmission of compressed video over a packet-switched network. The sender sampling curve is replaced by the encoder curve $e(t)$ and the receiver playout curve by the decoding curve $d(t)$ assuming that encoding and decoding times are negligible.

packet curve that is no longer a shifted version of the sender packet curve $b_s(t)$. Every packet encounters a different delivery time. Depacketization time is again neglected, which leads to a receiver curve $r(t)$ that is identical to the receiver packet curve $b_r(t)$. Once the bit stream portion of a video frame is available, the decoder can decode and display the frame. Because data compression reduces the amount of data per frame it is difficult to show the sender sampling and receiver playout curves in Figure 16.3. If we assume that the encoding time of a video frame is negligible we can replace the sender sampling curve $p_s(t)$ by the encoder curve $e(t)$. Please note that they are not the same but the steps in both curves happen at the same time. The step height, however, is different because of compression. Similarly, we can assume that the decoding of a frame starts when the frame is to be displayed and decoding time is negligible. Then, the receiver playout curve can be replaced by the decoder curve $d(t)$. Again, they are not identical but the steps in both curves happen at the same time instant. They only differ in step height. Most of the time we are only interested in identifying if enough data has been received by the client to display a certain media unit. Given the assumptions made earlier, we can draw the same conclusions about delay and playout interruptions from $e(t)$ and $d(t)$ that we would obtain when looking at $p_s(t)$ and $p_r(t)$.

In Figure 16.3, the receiver starts playout at time $T_{\text{initial}} = 170$ ms. Every 40 ms a new frame has to be displayed. It can be seen from Figure 16.3 that this selection of the initial playout time T_{initial} leads to a successful decoding process as the decoding curve $d(t)$ is always to the right of the receiver packet curve $b_r(t)$. This means that the decoder always has sufficient data to decode the video frames before their scheduled display time.

16.3 CLIENT BUFFERING

The standard way of dealing with the VBR nature of both the source bit stream and the network data delivery is by using a receiver buffer. The main purpose of this buffer is to store media data after the server starts sending the packets. For low delay applications, the client buffer mainly absorbs packet delay jitter. For applications with moderate delay requirements, the client buffer additionally provides time for the retransmission of lost packets. The amount of data prebuffered before the playout starts influences the initial waiting time and the late loss rate. Both quantities significantly influence user satisfaction.

16.3.1 Buffer Size versus Initial Delay

After a certain initial waiting time or once the buffer occupancy has reached a predefined target level, the client initiates the playout process, that is, the first media unit is played at T_{initial}. The size of the buffer determines the initial delay T_{initial} observed by the user. For live media streams the initial delay is the time difference between the sampling instant of the first media sample at the sender and display of this sample at the receiver. For pre-encoded media content, for example, in video streaming scenarios, T_{initial} is the time it takes between sending a request to the streaming server and displaying the first media unit at the client.

If we select the receiver buffer to be large, we are able to smooth significant delay variations. An extreme case is file download, where the buffer target fullness corresponds to the file size and playout only starts once the entire file has been completely transferred. If our aim is to keep the perceived end-to-end delay small, prebuffering has to be used carefully. This is particularly true for conversational applications where the end-to-end delay is critical for user satisfaction. For bidirectional conversational services involving speech and video, the tolerable end-to-end delay is typically given in the range of 150–250 ms. There is obviously a trade-off between robustness against network quality variations and initial delay.

16.3.2 Late Loss Rate versus End-to-End Delay

Once the client receives the first packets from the sender, in principle the playout process can be started. In order to allow a continuous playout at the receiver it

is, however, wise to wait some additional time to fill up the receiver buffer. The playout process at the receiver works without interruptions as long as the decoder curve $d(t)$ always stays to the right and below the receiver packet curve $b_r(t)$. If the two curves intersect, the decoder has to decode data that has not yet been received. In this case some of the packets arrive after their scheduled decoding time, which results in so-called late loss. Typically, regular packet loss and late loss can be jointly considered, as a true packet loss is simply a late loss where the delivery time is infinitely large. The influence of late loss on the reconstruction quality is application dependent. While speech applications where speech segments are encoded individually typically tolerate late loss rates of up to about 5%, video applications where the error-free decoding of one frame depends on the successful decoding of previous frames typically do not tolerate packet loss. The receiver therefore has the difficult task of deciding the initial playout delay T_{initial} such that the tolerable late loss rate is not exceeded. A large value of T_{initial} reduces late loss but at the same time increases the user perceived latency of the application.

Figure 16.4 shows sender and receiver curves for the transmission of voice packets over a packet-switched network and two example selections of T_{initial}. In the top plot of Figure 16.4 the playout process is started at $T_{\text{initial}} = 130$ ms, which leads to late loss of two out of nine packets. In the bottom plot of Figure 16.4 T_{initial} is reduced to 120 ms, which leads to late loss of four out of nine packets. In the lower plot of Figure 16.4 the played signal part is shown as a thick line on top of the desired decoding curve and it can be observed that three playout interruptions happen.

The resulting loss rate is determined by counting the packets that are not available at their decoding deadline and dividing this number n_{late} by the total number of packets of the session n_{session},

$$p_{\text{loss}} = \frac{n_{\text{late}}}{n_{\text{session}}}. \tag{16.1}$$

Depending on the selection of T_{initial} the receiver has to prebuffer different amounts of data. The larger T_{initial}, the larger the required buffer capacity at the receiver side. If the buffer is not large enough to hold all the data, buffer overflow occurs and media packets are lost despite their successful and timely arrival at the receiver. In practical applications we can typically assume that the receiver buffer capacity is large enough to hold all received packets before they are decoded. Our main concern is buffer underflow caused by late arrival of information.

Example 3: Late Loss Rate versus End-to-End Delay for VoIP

Figure 16.5 and Figure 16.6 show an example of late loss rate versus initial delay for a VoIP scenario where 20-ms voice packets are transmitted from a host located

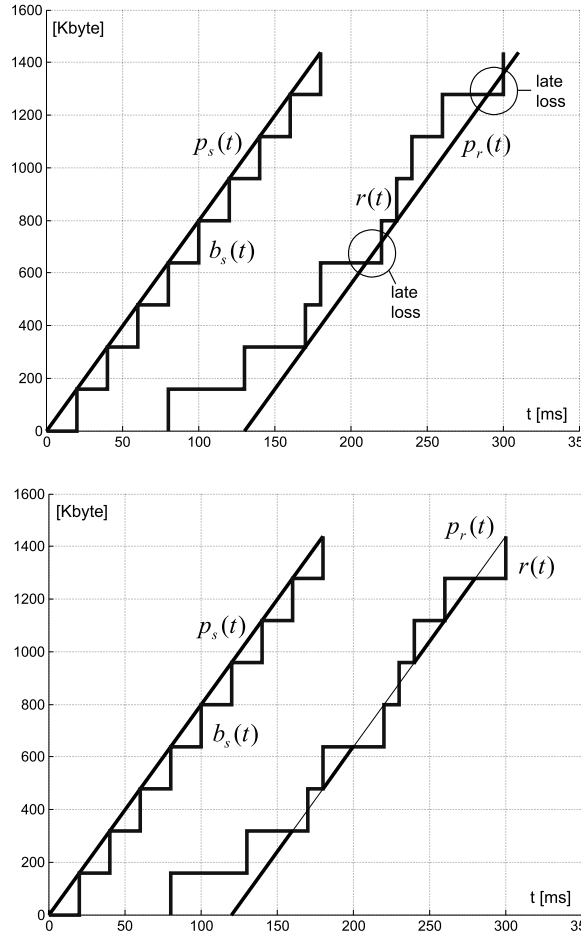


FIGURE 16.4: Sender and receiver curves for the transmission of voice packets over a packet-switched network. (Top) The initial playout time is $T_{\text{initial}} = 130$ ms, which leads to two late packets. (Bottom) The initial playout time is reduced to 120 ms, which leads to four out of nine packets being late for the playout process.

at the West Coast of the United States to a host at the East Coast. Figure 16.5 shows the measured delay values for 250 packets, and Figure 16.6 shows the resulting late loss rate as a function of the initial playout delay T_{initial}. The larger the end-to-end delay, the smaller the late loss rate. However, the larger the user-perceived application latency.

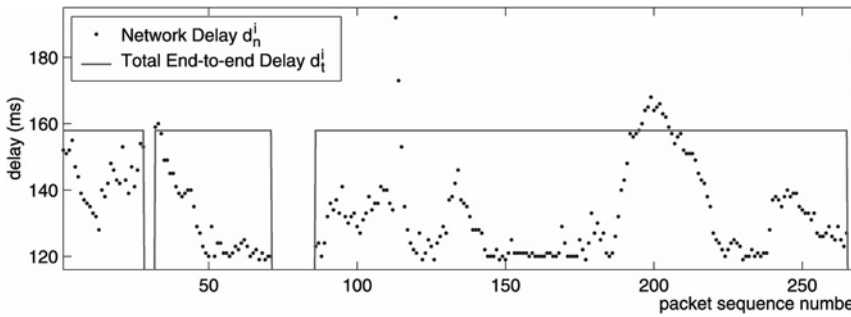


FIGURE 16.5: Measured packet delivery times for VoIP.

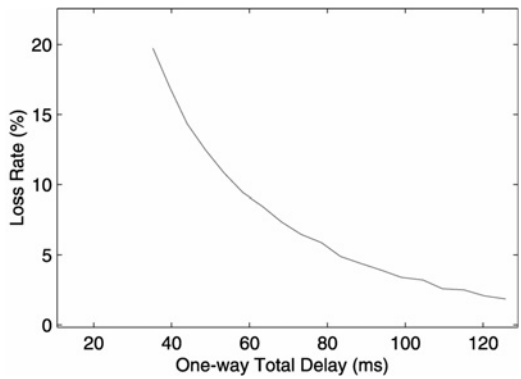


FIGURE 16.6: Late loss rate versus user-perceived end-to-end delay.

16.4 ADAPTIVE MEDIA PLAYOUT

Adaptive Media Playout allows the client to buffer less data and thus introduce less delay to achieve a given playout reliability. When using AMP, the receiver varies the rate at which it plays out audio and video. The playout speed can, for instance, be controlled by the state of the receiver buffer. In this case, when the buffer occupancy is below a desired level, the client plays media slowly to reduce its data consumption rate. For conversational services or streaming of live content, slowed playout causes the user-perceived latency to increase. Faster-than-normal playout is used in this case during good channel periods in order to eliminate or reduce excess latency accumulated with slowed playout. Faster-than-normal playout is unnecessary in the case of streaming of prestored programs however. Prestored programs that are slowed during bad channel periods will simply last longer at the client. By manipulating playout speed, AMP can reduce initial buffering delays in

the case of prestored streams and reduce the viewing latency (end-to-end delay) of live streams—all without sacrificing playout reliability. Figure 16.7 revisits the scenario introduced in the top plot of Figure 16.4 where two out of nine voice packets could not be played because of their late arrival. Adaptive Media Playout copes with this situation by stretching the playout duration of some voice packets. In Figure 16.7 the third voice segment is played twice as long as normal, which delays the playout deadline of all following packets by 20 ms. Hence, the receiver playout curve changes from $p_r(t)$ to $p_r^{AMP}(t)$. It can be seen from Figure 16.7 that now all packets are available at their playout deadlines. The playout curve $p_r^{AMP}(t)$ is always below and to the right of the receiver curve $r(t)$, which was not the case before. From a user perspective, playout interruptions are avoided. The end-to-end latency, however, increases by 20 ms after stretching the third packet.

To control the playout speed of media, the client scales the duration of one or more media units. For video signals this corresponds to changing the display duration of video frames. For audio or speech signals the duration of segments has to be changed without affecting its pitch [9,24]. Section 16.5 discusses algorithms for media duration scaling in detail.

It is interesting to note that playout speed modification has a precedent in traditional media broadcasting. Motion pictures shot at a frame rate of 24 fps are

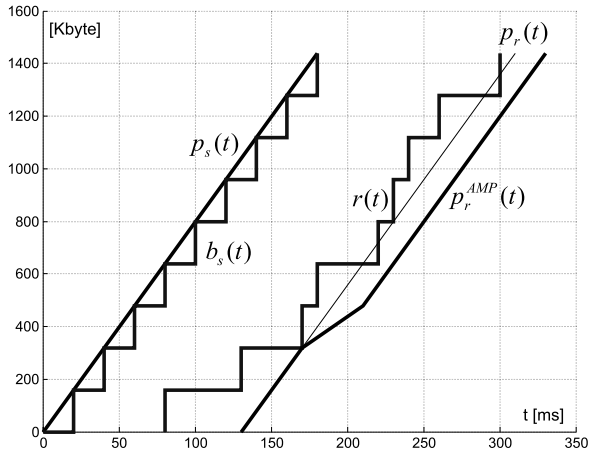


FIGURE 16.7: Adaptive Media Playout for the speech transmission scenario introduced in Figure 16.4. The third packet is played at half the speed at the receiver. This leads to a change of the playout curve and the deadline of all following packets is shifted to the right. Buffer underflow and hence playout interruption are avoided.

shown on European PAL/SECAM broadcast television at 25 fps. Video frames are displayed 1000/25 ms instead of 1000/24 ms, which corresponds to a media unit dilation of 4% and it is typically done without audio timescale modification.

16.4.1 Adaptive Media Playout for Low-Delay Conversational Services

In low-delay conversational services (e.g., video conferencing, Internet Telephony) excessive end-to-end delay impairs the interactivity of communication (see also Chapter 15). The latency experienced when completely absorbing delay jitter and eliminating late loss by receiver buffering can be very high. With Adaptive Media Playout, packet delay variations are compensated by playout speed variations.

The receiver has to decide when to start the playout of the media data once a session has been established. One way to decide the start of playout is to wait for the first speech packet to come in and then wait some additional time (safety margin) before playing this packet. Once the first packet is played, the playout deadlines for all following packets are fixed. If the safety margin was too conservative because the first packet was delayed exceptionally, the end-to-end delay is bigger than necessary. If the first packet arrives exceptionally early, the buffer will be selected too small and the late loss rate of the following packets might become too high.

Adaptive Media Playout addresses this issue by adaptively modifying the end-to-end delay using a playout scheduler that slows down playout if packet delivery times are increasing and speeds up playout if packet delivery times are decreasing. The basic operation of the playout scheduler is to set the playout time for each packet. As a result, network jitter is smoothed and mean end-to-end delay can be minimized. The actual end-to-end delay experienced by the user is continuously changing. As long as this variation stays within certain limits it is not impairing the quality of the communication. Only if the end-to-end delay increases significantly, bidirectional conversations become unnatural and participants start interrupting each other.

For low-delay conversational services the amount of media data available in the receiver buffer and therefore the number of packets available for playout scaling at any time are typically very limited due to the stringent end-to-end delay requirements. This means that the playout scheduler might have to significantly increase the playout duration of single packets in case of sudden changes in packet delay. In extreme situations, the scaling of the current media packet has to be decided without knowing the arrival time of the next packet. In order to keep the current packet concatenated with the next one at output, the arrival time of the next packet has to be estimated. If the delay estimation of the next packet is accurate and the current packet is scaled accordingly, the next packet should arrive and be ready by the end of playback of the current packet. A reliable estimation of the net-

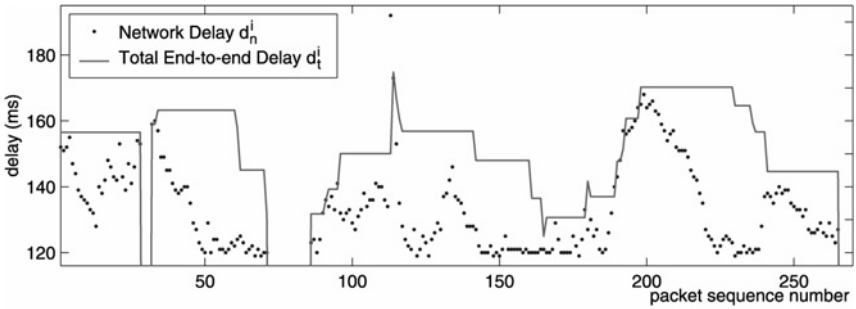


FIGURE 16.8: Adaptive Media Playout for VoIP. The playout schedule is adjusted within talkspurts. Gaps in solid lines correspond to silence periods between talkspurts.

work delay is therefore an important component for adaptive playout scheduling in low-delay conversational services.

Example 4: Adaptive Media Playout for VoIP

When using Adaptive Media Playout for Voice over IP, the playout schedule may not only be adjusted during silence periods but also within talkspurts, as illustrated in Figure 16.8. Each individual packet may have a different playout schedule, which is set according to the varying network condition. With Adaptive Media Playout, the interval between playout times or the length of each voice packet is no longer a constant, although the packetization period is. Continuous output of audio can be achieved by scaling the voice packets using the signal processing techniques described in Section 16.5. For the same delay trace as shown in Figure 16.5, the adaptive scheme is able to effectively reduce average delay and mitigate late loss by adjusting the playout schedule in a more dynamic way. The trade-off between buffering delay and late loss can hence be improved, as shown in Figure 16.9.

16.4.2 Adaptive Media Playout for Nonconversational Services with Moderate-Delay Requirements

Section 16.4.1 described Adaptive Media Playout for low-delay conversational services where due to stringent end-to-end delay requirements typically only very few packets are in the receiver buffer at any time. The main challenge for low-delay applications is to accurately estimate the arrival time of the following packets in order to be able to decide the playout duration of the current packet. The limit on the end-to-end delay is strict, which requires that additional delay intro-

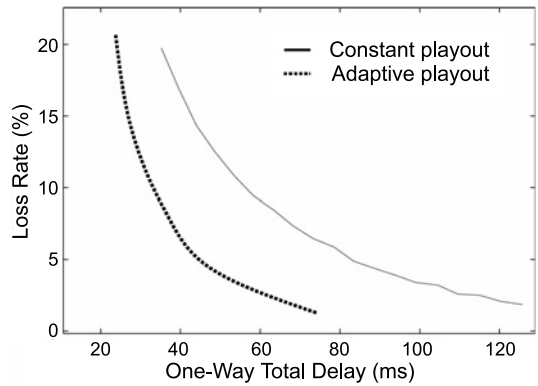


FIGURE 16.9: Trade-off between average end-to-end delay and late loss rate for constant and adaptive playout. The trade-off curve for constant playout is identical to the one shown in Figure 16.6.

duced by slow playout has to be compensated at a later time by faster-than-normal playout. In addition, the required changes in playout duration may be substantial if sudden changes in packet delivery time occur. Some packets may have to be scaled by 100% or more in order to follow network delay variations quickly enough.

The situation is very different for applications that can prebuffer significant amounts of data in their receiver buffer as a result of their moderate latency requirements. Internet Video Streaming is a popular application that falls into this category. This application will be used in the following to describe the use of Adaptive Media Playout for applications with moderate-delay requirements.

Video streaming over the Internet is an example for VBR traffic over a VBR channel. In video streaming, a client requests a pre-encoded media stream from a media server. The pre-encoded video stream is typically encoded at a variable bit rate so the encoder curve $e(t)$ is similar to the one shown in Figure 16.3. The media server packetizes the pre-encoded media stream and sends the packet stream $b_s(t)$ over the Internet to the client. Following the argument in Section 16.2 we use the encoder curve $e(t)$ and decoder curve $d(t)$ instead of the sender sampling curve $p_s(t)$ and receiver playout curve $p_r(t)$ as the video frames are compressed. The decoder curve $d(t)$ tells us at what time a certain media unit (video frame) has to be decoded and displayed at the decoder. This time is relative to the decoding and playout time of the first video frame and for constant playout simply becomes a shifted version of the encoder curve $d(t) = e(t - T_{\text{initial}})$.

Adaptive Media Playout can again be used to reduce the perceived latency while maintaining a desired playout reliability [19]. From a user perspective there

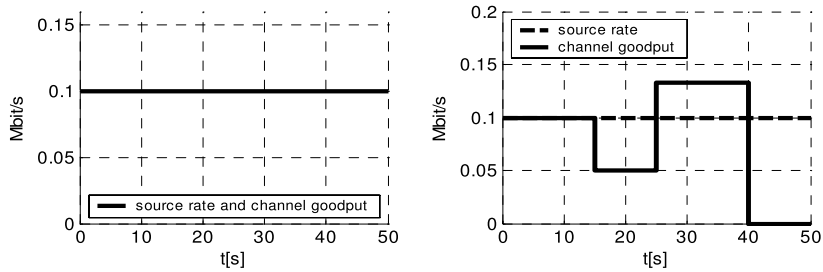


FIGURE 16.10: The source rate is fixed at 100 kbit/s. Two examples of transmission goodput are used in the following. (Left) Constant goodput that matches the source rate. (Right) Variable goodput in the range from 0 to 133 kbit/s.

are two buffering delays that are noticeable. Start-up delay is the time that it takes for the client buffer to fill to a desired level so that playout can begin after a user request. Viewing latency, noticeable in the case of live streams, is the time interval separating a live event and its viewing time at the client. To explain how AMP can be used to reduce these delays, we distinguish among three separate modes of operation [8], illustrated in Figures 16.11–16.13. These modes are called AMP-Initial, AMP-Robust, and AMP-Mean.

For illustrative purposes we will base our discussion in the following on the two specific transmission scenarios shown in Figure 16.10. On the left-hand side of Figure 16.10 a constant bit-rate source stream is transmitted over a constant bit-rate channel. The source and the channel rate match and are both 100 kbit/s. On the right hand side of Figure 16.10 the constant bit-rate source stream is sent over a channel with variable goodput. The goodput varies as a function of time. The maximum goodput $g(t)$ reaches 133 kbit/s.

16.4.2.1 Initial Playout Delay Reduction (AMP-Initial)

AMP-Initial is used to decrease the start-up delay. In this mode, the client initiates the playout process before the buffer is filled to the usual target level. Despite this early start of playout the buffer is able to fill to the target level over time by initially playing the media slower than normal. The buffer fills over time since the data consumption rate during slowed playout is smaller than the arrival rate of the media packets, assuming that during normal playout the source rate and the channel goodput match the data consumption rate at the decoder. Once the target level is reached, the playout speed returns to normal. This technique allows fast switching between different programs or channels without sacrificing protection against adverse channel conditions, after the initial buffer is built up.

Figure 16.11 illustrates AMP-Initial. The source rate and channel goodput correspond to the left-hand side of Figure 16.10. The consumption rate of the playout process at normal playout speed is 0.1 Mbit/s and hence matches the source rate. The second plot in Figure 16.11 shows the client buffer occupancy as a function of time for the case of nonadaptive playout. The target buffer level is assumed to be 1 Mbit, yielding a preroll time of 10 s in this example. The third plot illustrates the client buffer occupancy for the AMP-Initial scheme in which playout starts when the buffer occupancy is only half the target level. This happens after 5 s. The client slows playout initially to allow the buffer occupancy to increase over time. The

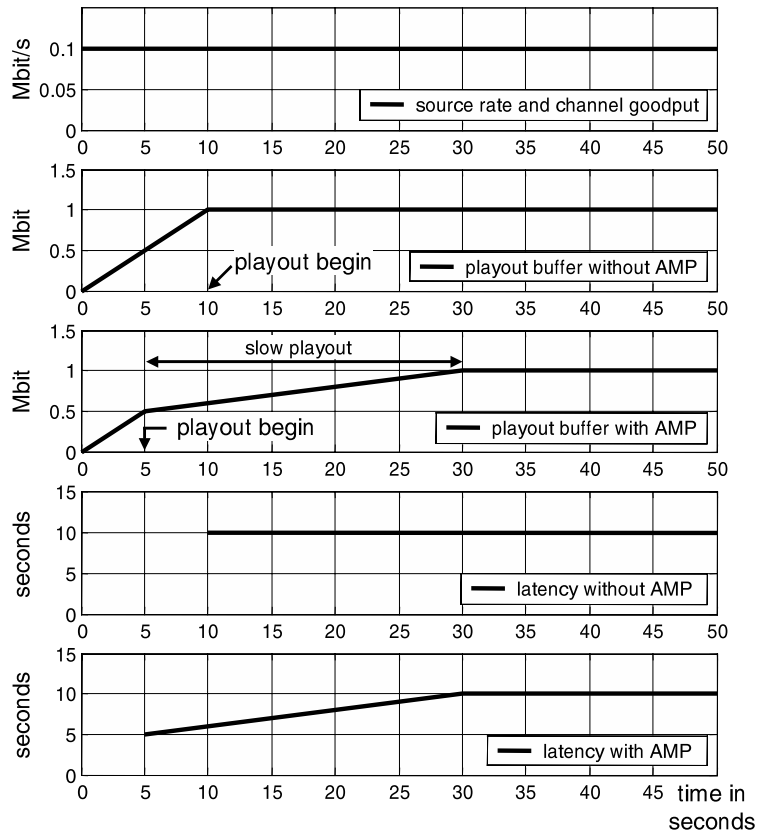


FIGURE 16.11: AMP-Initial: For low start-up delays, playout begins after a reduced number of frames are buffered at the client. Slowed playout allows the buffer occupancy to grow to a safer target level over time. In this example, frame periods are stretched by 20% during slow playout periods.

media units are stretched by 20% during slowed playout and hence after a total of 30 s the target buffer level is reached. The two lower plots in Figure 16.11 show the viewing latency with and without AMP-Initial. While the latency remains constant for the nonadaptive case, for the AMP-Initial scheme latency increases from 5 s initially to 10 s when the target buffer level is reached.

16.4.2.2 Improved Robustness Against Network Variations (AMP-Robust)

As illustrated in Figure 16.12, AMP-Robust increases the robustness of the playout process with respect to variations of goodput. In this mode the playout speed is

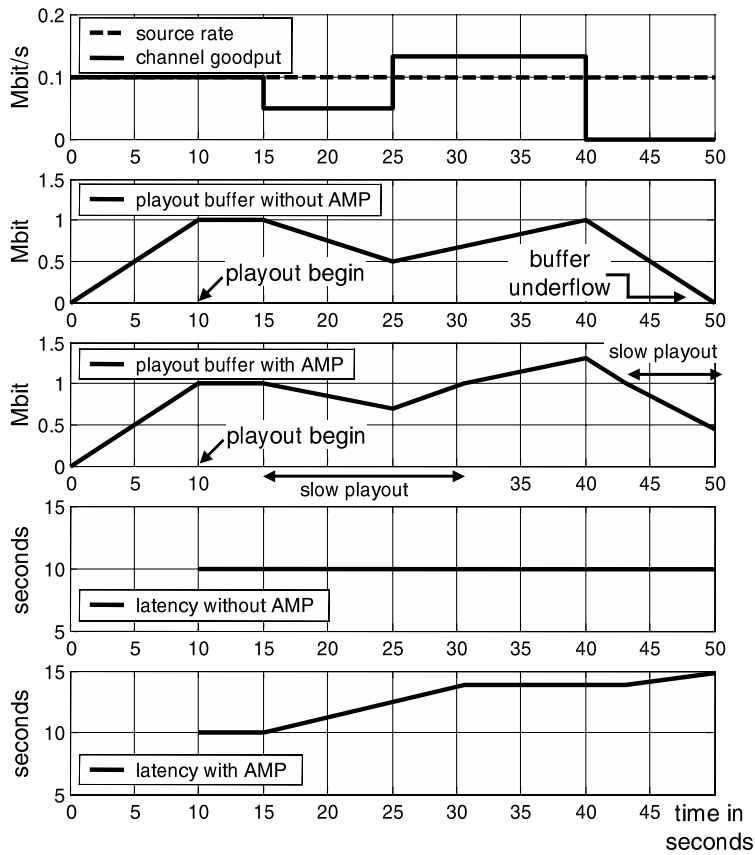


FIGURE 16.12: AMP-Robust: In this scheme, suitable for prestored programs where viewing latency is not important, slowed playout is used to keep the buffer occupancy at a desired level.

simply reduced whenever the buffer occupancy falls below the target level. Now, the transmission scenario shown on the right-hand side of Figure 16.10 is considered. As before, the source rate is a constant 0.1 Mbit/s. The channel goodput varies over time with a reduction to 0.05 Mbit/s at $t = 15$ s, an improvement to 0.133 Mbit/s at $t = 25$ s, and a complete channel outage at $t = 40$ s. The second plot of Figure 16.12 shows the buffer occupancy as a function of time for nonadaptive playout. The target buffer level is again 1 Mbit, which leads to a playout start at $t = 10$ s. Playout is interrupted, however, after 50 s, when reductions in the channel goodput lead to a buffer underflow. The third plot in Figure 16.12 shows the buffer occupancy for the AMP-Robust scheme in which the client stretches frame periods by 25% whenever the buffer occupancy falls below the target level. In this example, buffer underflow is averted with AMP. The lower two plots in Figure 16.12 show the viewing latency as a function of time. For nonadaptive playout, latency is constant. For the adaptive case, the latency increases whenever playout is slowed, which is fine for a prestored program. Note that playout starts at $t = 10$ s for both cases. AMP-Robust can be combined with AMP-Initial to also allow reduced start-up time.

16.4.2.3 Live Media Streaming (AMP-Live)

AMP-Live is suitable for the streaming of live programs. In this mode, the client slows playout during bad channel periods but also plays media faster than normal during good channel periods to reduce additional viewing latency that has accumulated during periods of slowed playout. By playing the media faster and slower than normal, the mean viewing latency can be reduced for a given probability of buffer underflow. An example of the application of AMP-Live is given in Figure 16.13 for the transmission scenario introduced in the right-hand side of Figure 16.10.

Whenever the buffer occupancy falls below the target level, playout is slowed. When the occupancy is greater than the target level, media is played faster than normal to eliminate excess latency. In Figure 16.13, during faster playout the client reduces frame periods by 25%, which corresponds to a 33% increase in the data consumption rate. Therefore, the buffer remains at the target level in the third plot of Figure 16.13 during fast playout. Latency, shown in the lower two plots of Figure 16.13, decreases during faster-than-normal playout.

16.5 SIGNAL PROCESSING FOR ADAPTIVE MEDIA PLAYOUT

16.5.1 Time Compression and Dilation of Speech and Audio Signals

When Adaptive Media Playout is used, the duration of the audio or speech signal has to be scaled without impairing quality. The scaling of a voice or audio

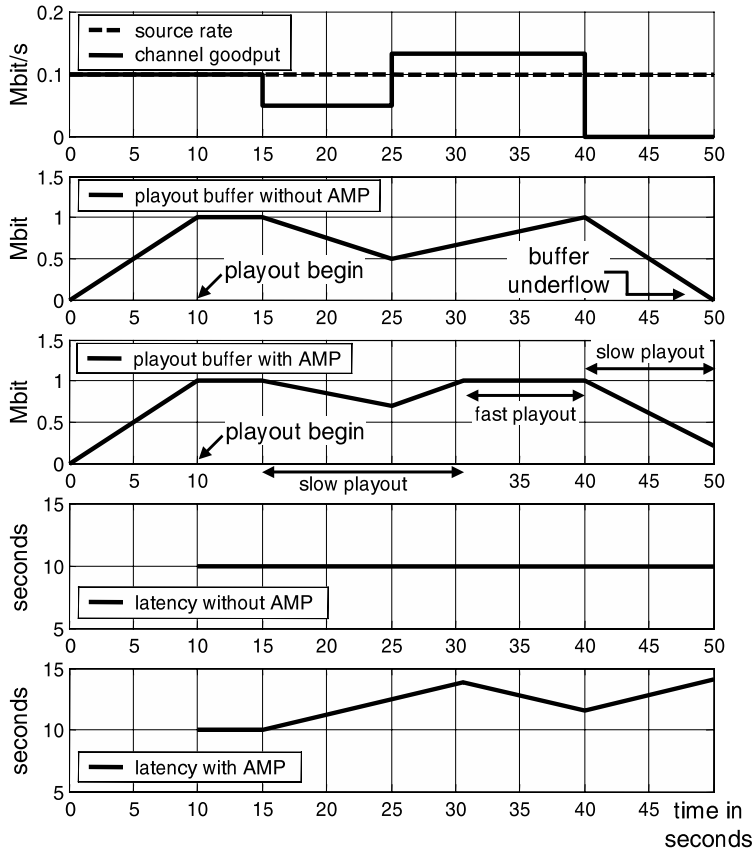


FIGURE 16.13: AMP-Live: For live streams, low viewing latency is desirable. The client slows playout when poor channel conditions threaten to starve the client buffer. During good channel periods, however, faster-than-normal playout is used to reduce or even eliminate latency accumulated during periods with slowed playout.

segment may be realized by *timescale modification* based on the *Waveform Similarity Overlap-Add* (WSOLA) algorithm, which is an interpolation-based method operating in the time domain. This technique was used in [21] to scale long audio blocks and was modified and improved in [20] and [17] for loss concealment by expanding a block of several packets. For a detailed discussion on error concealment for audio communication, refer to Chapter 3.

The basic idea of WSOLA is to decompose the input into overlapping segments of equal length, which are then realigned and superimposed to form the output with equal and fixed overlap. The realignment leads to modified output length. For those segments to be added in overlap, their relative positions in the input are found through the search of the maximum correlation between them so that they have the maximum similarity and the superposition will not cause any discontinuity in the output. Weighting windows are applied to the segments before they are superimposed to generate smooth transitions in the reconstructed output. For speech processing, WSOLA has the advantage of maintaining the pitch period, which results in improved quality compared to resampling.

Since the goal of Adaptive Media Playout is to reduce delay, low processing delay is desirable. The conventional WSOLA algorithm can be tailored and improved to work on only *one* packet. In other words, an incoming packet can be scaled immediately and independently, without introducing any additional processing delay. To scale a voice packet, a *template segment* of constant length is first selected in the input. Then a *similar segment* that exhibits maximum similarity to the template segment is being searched. The start of the similar segment is searched in a *search region*, as shown in Figure 16.14. When working on a single packet, the search for a similar segment is more constrained, as the realignment of the similar segments must be made in units of pitch periods and there are fewer pitch periods available in one short packet. For a 20-ms packet, depending on the speaker's gender and voice pitch, there could be fewer than two pitch periods included, which makes it difficult to extract the target segments with similarity. To overcome this problem, the conventional WSOLA algorithm has to be modified to decrease the segment length for correlation calculation, and the first template segment is positioned at the beginning of the input packet, as shown in Figure 16.14a. To expand short packets, the search region for the first similar segment is moved to the prior packet in order to have a larger range to look for similar waveforms. In Figure 16.14a, although the input packet starts in Pitch Period 2, the similar segment is found within Pitch Period 1. Although the prior packet might already be played out at the time of scaling, similar waveforms can still be extracted from it to construct new output without delaying the prior packet. Once the similar segment is found, it is weighted by a rising window and the template segment is weighted by a symmetric falling window. The similar segment followed by the rest of the samples in the packet is then shifted and superimposed with the template segment to generate the output. The resulting output is longer than the input due to the relative position of the similar segment found and the shift of the similar segment, as shown in Figure 16.14a. The amount of expansion depends on the position and the size of the defined search region.

In Figure 16.14, complete pitch periods in the waveform are separated by vertical dashed lines and marked with sequential numbers. For example, in Figure 16.14a, it is observed from the output waveform that one extra pitch period

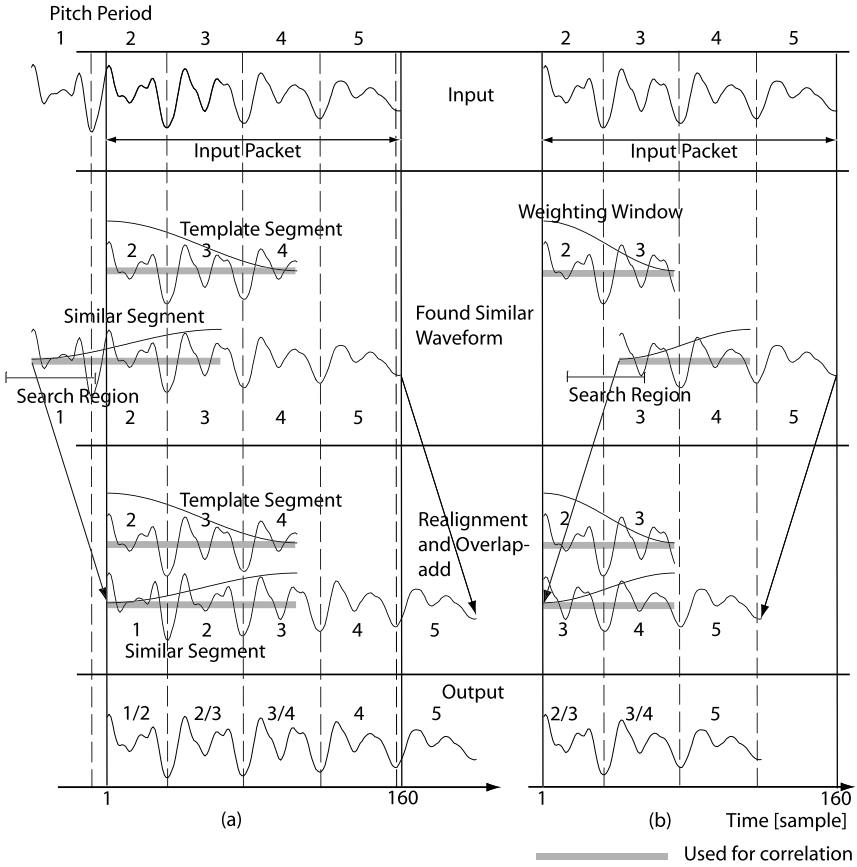


FIGURE 16.14: Extension (a) and compression (b) of single voice packets using timescale modification.

is created and added as a result of realignment and superposition of the extracted segments from the input. However, the extra pitch period is not just a simple replication of any pitch period from the input, but the interpolation of several pitch periods instead. For the output in Figure 16.14a, the first three pitch periods are the weighted superposition of Pitch Periods 1/2, 2/3, and 3/4, respectively. This explains why the sound quality using timescale modification is better than that of pitch repetition (described in [4,13]). The same is true for compressing a packet, where the information carried by a chopped pitch period is preserved and distributed among the remaining ones. The operations of searching for a similar segment and extending the packet by multiple pitch periods, as described earlier,

constitute one *iteration* in the scheme. If the output speech has not met the target length after such operations, additional iterations are performed. In a subsequent iteration, a new template segment of the same length is defined that immediately follows the template in the last iteration. All the defined template segments and the remaining samples following the last template in the input should cover the entire output with the target length. Packet compression is done in a similar way, as depicted in Figure 16.14b. The only difference is that the search region for the similar segment should not be defined in the prior packet in order to generate an output shorter in length.

Since the scaling of packets has to be performed in integer multiples of pitch periods, it is unlikely to achieve the exact packet length as targeted by the adaptive scheduler. However, as the resulting output packet length is fed into the scheduler, this inaccuracy will be absorbed and corrected in potential scaling of the following packets so that the overall schedule is maintained as targeted.

Comparing the input and output waveforms in Figure 16.14, it becomes obvious that the operation preserves the pitch frequency of the input speech. Only the packet length and hence the rate of speech are altered. Subjective listening tests show that infrequent scaling of the packets does not degrade speech quality, even if the scaling ratio is occasionally high [12]. Note that the scheme is speech codec independent. The operations can be applied on the PCM output.

One advantage of working with a short packet is that the input is divided into fewer template segments so that typically only one or two iterations will yield the output with the target length. Another important feature of the algorithm observed in Figure 16.14 is that the beginning and the end of each packet are not altered. As a result, when concatenating modified packets, no overlap or merging is needed to obtain smooth transitions. Hence, packets can be modified independently and sent to the output queue back to back. This type of operation is ideally suited for a highly adaptive playout scheduler.

16.5.2 Time Compression and Dilation of Video Signals

While changing the display duration of audio and speech signals requires the use of special timescale modification algorithms as described in the previous section, for video the situation is much easier. Time compression or dilation of a video signal can be achieved by simply changing the display duration of individual video frames. If the display process supports only fixed duration display of video frames, repetition or dropping of video frames can be used instead. As an example let us assume that we want to scale the playout of the video signal by 20% for a video sequence with 25 frames per second and a fixed display duration of 40 ms per frame. Repetition of every 5th frame makes 30 frames or 1.2 s out of an original 1-s segment of the video signal. For a fixed display duration of 40 ms

per frame the playout duration is now 1.2 s, which corresponds to the desired 20% playout dilation.

16.5.3 Time Compression and Dilation of Silence Periods

As mentioned previously, scaling of up to 100% is tolerable if applied infrequently to short segments of speech. Continuous scaling of speech or audio signals for longer segments, as needed for AMP in video streaming scenarios, however, becomes noticeable and practical scaling factors are much lower. In these cases typically scaling of up to 25% can be used. However, if we detect silence within the packets in our client buffer, then we can overproportionally stretch or compress the silence periods without significantly impairing the perceived quality.

16.6 PLAYOUT SPEED CONTROL MECHANISMS

Playout speed control is a key issue when using Adaptive Media Playout and, in general, the playout scheduler has to be designed individually for a particular application. While, for example, a simple heuristic that chooses a playout speed from a small set of discrete values depending on the occupancy of the playout buffer is suitable for Internet Media Streaming, low-delay conversational services as described in Section 16.4.1 require more sophisticated schemes. Advanced playout speed control mechanisms may, for instance, operate in a way that meets a constraint on expected distortion or finds an optimal trade-off among the distortion due to playout speed variation and the distortion due to expected decoding errors, and latency. In the following we discuss a selection of playout speed control mechanisms.

16.6.1 Heuristic Playout Speed Control for Video Streaming

A simple adaptive playout strategy for video streaming controls the playout frame rate by examining the number of frames of media in the playout buffer. Let $\mu(n)$ be the playout frame rate dictated by the scheme, where n is the number of frames that occupy the playout buffer. In the nonadaptive case this rate is constant and given by $1/t_F$, where t_F is the frame period. With Adaptive Media Playout, $\mu(n)$ varies with n. For example, a simple playout policy for AMP-Live (introduced in Section 16.4.2) is

$$\mu(n) = \begin{cases} \dfrac{1}{s \cdot t_F} & n < N_{\text{low}}, \\ \dfrac{1}{f \cdot t_F} & n > N_{\text{high}}, \\ \dfrac{1}{t_F} & \text{otherwise}, \end{cases} \tag{16.2}$$

where $s \geq 1$ represents a decrease in playout speed, $f \leq 1$ is an increase in playout speed, n is the number of contiguous frames in the playout queue, and N_{high} and N_{low} are threshold values. When there are fewer than N_{low} frames in the playout queue, each frame plays for $s \cdot t_F$ seconds. When the number in the queue exceeds N_{high}, each frame plays for $f \cdot t_F$ seconds.

In general, functions $\mu(n)$ can be defined depending on goals with respect to robustness, mean latency, buffer size, initial preroll delay, and playout speed variability. For instance, in the case of AMP-Initial and AMP-Robust, the controller in (16.2) can be used with $N_{\text{high}} = \infty$. Alternative heuristic controllers may, for example, allow playout speed to vary continuously as a function of buffer occupancy.

16.6.2 Distortion-Latency Optimized Playout Speed Control

This section discusses a scheme [7] that controls playout in a way that attempts to yield an optimal trade-off among distortion of decoded media, distortion due to playout variation, and latency.

In this scheme, a scaling factor for the playout speed of a frame is determined by an algorithm that jointly optimizes a vector of scaling factors $\nu \in \Re^M$ for the playout speed of a window of M future frames. The scheme finds a ν that gives an approximately optimal trade-off between the expected distortion $D(\nu)$ and functions $G_i(\nu)$ that assess perceptual costs due to playout speed and latency variation. The optimal trade-off is determined to be the one that minimizes the Lagrangian cost function

$$J(\nu) = D(\nu) + \sum_i \lambda_i G_i, \tag{16.3}$$

where the λ_i are user-defined weights for the perceptual costs. Cost functions $G_i(\nu)$ that assess the perceptual impact of playout speed and latency variation can be defined, for example, as

$$G_1(\nu) = \sum_{i=1}^{M} (\nu_i - 1)^2, \tag{16.4}$$

$$G_2(\nu) = \sum_{i=1}^{M} (\nu_i - \nu_{i-1})^2, \tag{16.5}$$

$$G_3(\nu) = \left(t_F \sum_{i=1}^{M} (\nu_i - 1) + t_{\text{accum}} \right)^2, \tag{16.6}$$

where M is the length of vector ν. In these examples, G_1 assesses a cost for deviation of the frame-period scaling factors from 1, G_2 assesses a cost on variations in the playout rate from one frame to the next, and G_3 assigns a cost to latencies that do not match the initial buffering delay. G_3 is useful in cases such as live or interactive streams in which latency must be constrained. In (16.6), t_{accum} is the amount of delay that will have accumulated by the end of the currently playing frame, less the initial buffering delay.

16.6.3 Adaptive Media Playout and R–D Optimized Media Streaming

The control scheme described in Section 16.6.2 can be integrated with the framework for rate-distortion optimized streaming described in Chapter 10.2.1 [7].

The scheme uses receiver-driven streaming in which the receiver calculates optimal requests to transmit to the sender and the sender simply transmits the requested packets.

If we combine Adaptive Media Playout with receiver-driven RD optimized streaming, transmission requests and playout speeds have both to be computed at the receiver and can therefore be computed jointly. In this case, the Lagrangian optimization objective of (16.3) is augmented to be a function not only of the playout speeds ν, but also of the transmission policy π. The goal in this case is to minimize the Lagrangian

$$J(\pi, \nu) = D(\pi, \nu) + \lambda R(\pi, \nu) + \lambda_1 G_1(\nu) + \lambda_2 G_2(\nu) + \lambda_3 G_3(\nu), \qquad (16.7)$$

where $R(\pi, \nu)$ is the expected transmission rate as a function of the transmission policy and playout speeds. The iterative descent algorithm used to minimize the Lagrangian with respect to the transmission policy π remains mostly as described in Section 10.2.1 of this book. The Lagrangian is iteratively minimized one variable at a time while the others are held fixed. The step of minimizing the Lagrangian with respect to the playout speed vector while the transmission policy variable is held fixed is added, however.

16.6.4 Playout Speed Control for Low-Delay Applications

As discussed in Section 16.4.1 the success of Adaptive Media Playout for low-delay conversational services mainly depends on how accurately the receiver can estimate the arrival time of future packets. A key component of playout speed control for this kind of application is therefore packet transfer delay estimation. Several delay estimation techniques have been proposed, including linear recursive filtering with stochastic gradient algorithms [16], histogram-based approaches [14,18], normal approximation [3], and event counting [23]. The playout speed

control mechanism used for VoIP shown in Figure 16.8 is based on the delay of the most recent packets [12]. In this scheme, the user specifies a desired rate ε_l of errors due to late loss. The algorithm estimates the late loss probability of a packet as a function of its playout deadline by examining the delays of a window of recent transmissions. The algorithm scales the playout speed of each frame so that the next packet's deadline yields an expected loss probability that meets the constraint set by the user.

To find the expected loss probability for a packet as a function of its deadline, the algorithm makes use of order statistics compiled from the delays of a window of the last w received packets. Let $D^1, D^2, \ldots, D^w$ denote the delays of the last w received packets in ascending order such that $D^1 \leq D^2 \leq \cdots \leq D^w$. Let the random network delay of a packet be denoted d_n. The expected cumulative distribution function (CDF) for the delay can be written as

$$E\left[\Pr\left\{d_n \leq D^r\right\}\right] = \frac{r}{w+1}, \quad r = 1, 2, \ldots, w \tag{16.8}$$

and represents the expected probability that a packet with the same delay statistics can be received with delay D^r or less. Interpolation can be used to evaluate the expected CDF at values between the D^r.

Using this estimate for the delay CDF, the deadline for a packet can be chosen so as to meet the late loss rate constraint ε_l. The playout speed of a speech segment is scaled to control the deadline of the packet containing the following segment.

One important feature of this history-based estimation is that the user can specify the acceptable loss rate, and the algorithm adjusts the playout schedule accordingly. Therefore, the trade-off between buffering delay and late loss can be controlled explicitly.

Over IP networks, it is common to observe sudden high delays ("spikes") incurred by voice packets, as shown by packets 113–115 in Figure 16.8. Delay spikes usually occur when new traffic enters the network and a shared link becomes congested, in which case past statistics are not useful to predict future delays. In this case, the scheduler switches from a normal mode to the *rapid adaptation mode* when the current delay exceeds the previous one by more than a threshold value. In rapid adaptation mode, the first packet with an unpredictable high delay has to be discarded. Following that, the delay estimate is set to the last "spike delay" without considering or further updating the order statistics. The rapid adaptation mode is switched off when delays drop to the level before the spike and the scheduler returns to its normal operation, reusing the state of order statistics before the spike occurred. This rapid adaptation helps to mitigate burst loss as illustrated in Figure 16.8.

16.7 ADAPTIVE MEDIA PLAYOUT AND PACKET PATH DIVERSITY

In Chapter 17, path diversity is described for reliable communication over lossy networks using multiple description coding. It has been observed that for multipath transmission the end-to-end application sees a virtual average path, which exhibits a smaller variability in quality than any of the individual paths. In this section, packet path diversity is revisited for low-delay applications when Adaptive Media Playout is employed.

16.7.1 Packet Path Diversity for Low-Delay Conversational Services

In the context of delay-sensitive applications, such as interactive VoIP, the largely uncorrelated characteristics of the delay jitter on multiple network paths can be exploited using Adaptive Media Playout techniques [10]. The multiple streams to be delivered via different paths are formed by multiple description coding (MDC), which generates multiple descriptions of the source signal of equal importance. These descriptions can be decoded independently at the receiver. If all descriptions are received, the source signal can be reconstructed in full quality. If only a subset of the descriptions is received, the quality of the reconstruction is degraded, but is still better than the quality resulting from losing all descriptions. Depending on the MDC scheme selected, the overall data rate of the payload does not necessarily increase as a result of transmitting multiple streams. The data rate only increases if redundancy is introduced between the multiple streams.

In order to maximize the benefits of multipath transmission, paths that exhibit largely uncorrelated jitter and loss characteristics are preferred. Sending streams along different routers from source to destination naturally leads to path diversity, which could include streams traversing different ISPs or even streams being sent in different directions around the globe. For a detailed discussion on how to practically realize multipath transmission, refer to Chapter 17.

16.7.2 Adaptive Two-Stream Playout Scheduling

To exploit the characteristics of multipath transmission, the playout scheduler is again a key component, as in the previous discussion in this chapter. An adaptive scheduler for two-stream playout is similar to the single stream case described in Section 16.4.1. Before the arrival of each packet i, the playout deadline for that packet has to be set according to the most recent delays recorded. The playout deadline of packet i is denoted by d^i_{play}, which is the time from the moment the packet is delivered to the network until it has to be played out. When determining the playout deadlines, the trade-off among delay, losing both MDC descriptions (referred to as *packet erasure*), and losing only one description has to be considered. This trade-off can be stated as the following constrained problem: given

a certain acceptable signal distortion, minimize the average delay $\mathcal{E}\{d^i_{\text{play}}\}$. This constrained problem can be formulated as an unconstrained problem by introducing a Lagrange cost function for packet i,

$$\begin{aligned} C^i &= d^i_{\text{play}} + \lambda_1 \cdot Pr(\text{both descriptions lost}) \\ &\quad + \lambda_2 \cdot Pr(\text{only one description lost}) \\ &= d^i_{\text{play}} + \lambda_1 \hat{\varepsilon}^i_{S_1} \hat{\varepsilon}^i_{S_2} + \lambda_2 \big(\hat{\varepsilon}^i_{S_1}\big(1 - \hat{\varepsilon}^i_{S_2}\big) + \hat{\varepsilon}^i_{S_2}\big(1 - \hat{\varepsilon}^i_{S_1}\big)\big), \end{aligned} \tag{16.9}$$

where $\hat{\varepsilon}^i_{S_1}$ and $\hat{\varepsilon}^i_{S_2}$ are the estimated loss probabilities of the descriptions in Streams 1 and 2, respectively, given a certain d^i_{play}. The estimate of $\hat{\varepsilon}^i_{S_1}$ and $\hat{\varepsilon}^i_{S_2}$ can be based on past delay values recorded for the two streams. The higher d^i_{play} is, the lower the loss probabilities since the likelihood of playing out delayed packets is higher. The Lagrange multipliers λ_1 and λ_2 are predefined parameters used to trade off delay and the two loss probabilities.

The playout deadline can be optimized by searching for the d^i_{play} that minimizes the cost function (16.9). Multiplier λ_1 is used to trade off total delay and packet erasure probability. Greater λ_1 results in a lower erasure rate at the cost of higher delay. Multiplier λ_2 is introduced to penalize distortion as a result of playing out only one description. The greater λ_2 is, the better the quality of the reconstructed signal at the cost of higher delay. Note that although packet erasure [the second term in (16.9)] and quality degradation due to the loss of one MDC description [the third term in (16.9)] are different perceptual experiences, they are not independent measures. From (16.9), increasing λ_2 also leads to lower erasure probability. However, with zero or very small λ_2, only packet erasure is considered. In this case, good reconstruction quality is not a priority, but lower latency is given more emphasis, with the trade-off between delay and erasure rate determined mainly by λ_1.

When switching between streams during speech playout, the playout schedule has to be dynamically adjusted and adapted to the delay statistics of each individual stream. Adaptive Media Playout in combination with packet path diversity has first been demonstrated in [10] for two-stream VoIP. The experimental results in [10] demonstrate that this combination has the potential to significantly reduce the application latency of low-delay conversational services over the Internet while preserving the user-perceived signal quality.

16.8 SUMMARY AND FURTHER READING

This chapter discussed Adaptive Media Playout as a receiver-based technique to adapt multimedia communication applications to varying transmission conditions.

The main idea of AMP is to consider the time axis as a rubber band that can be locally stretched or compressed. In the context of video streaming delivery, AMP can successfully be employed to decrease the initial as well as the average latency of the application. For VoIP the flexibility to adjust the playout duration of individual speech segments allows us to adjust playout deadlines even within talkspurts and to dynamically optimize the trade-off between latency and late loss rate. While for video the change of playout duration is straightforward, for audio and speech signals special signal processing techniques have to be employed, which lead to noticeable quality impairments if the scaling factor becomes too large. Finally, it should be mentioned that AMP is particularly beneficial in multicast or broadcast scenarios where every user sees a different channel and can individually adapt to the current transmission properties without getting the sender involved.

REFERENCES

[1] J. G. Apostolopoulos. "Reliable Video Communication over Lossy Packet Networks Using Multiple State Encoding and Path Diversity," *Proceedings Visual Communication and Image Processing*, pp. 392–409, January 2001.

[2] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. "The Case for Resilient Overlay Networks," *Proceedings of the 8th Annual Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Online at: http://nms.lcs.mit.edu/projects/ron/, May 2001.

[3] J. F. Gibbon and T. D. C. Little. "The Use of Network Delay Estimation for Multimedia Data Retrieval," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1376–1387, September 1996.

[4] David J. Goodman, Gordon B. Lockhart, Ondria J. Wasem, and Wai-Choong Wong. "Waveform Substitution Techniques for Recovering Missing Speech Segments in Packet Voice Communications," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 6, pp. 1440–1448, December 1986.

[5] M. Kalman, B. Girod, and E. Steinbach. "Adaptive Playout for Real-Time Media Streaming," *International Symposium on Circuits and Systems, ISCAS 2002*, Scottsdale, Arizona, May 2002.

[6] M. Kalman, E. Steinbach, and B. Girod. "R-D Optimized Media Streaming Enhanced with Adaptive Media Playout," *International Conference on Multimedia and Expo, ICME 2002*, Lausanne, August 2002.

[7] M. Kalman, E. Steinbach, and B. Girod. "Rate-Distortion Optimized Video Streaming with Adaptive Playout," *International Conference on Image Processing, ICIP 2002*, Rochester, New York, September 2002.

[8] M. Kalman, E. Steinbach, and B. Girod. "Adaptive Media Playout for Low Delay Video Streaming over Error-Prone Channels," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 6, pp. 841–851, June 2004.

[9] Y. Liang, N. Faerber, and B. Girod. "Adaptive Playout Scheduling Using Time-Scale Modification in Packet Voice Communications," *Proc. IEEE Intern. Conf. Acoustics, Speech, and Signal Processing, ICASSP-2001*, Salt Lake City, UT, May 2001.

[10] Yi J. Liang, E. Steinbach, and B. Girod. "Real-Time Voice Communication over the Internet Using Packet Path Diversity," *Proc. ACM Multimedia 2001*, pp. 431–440, Ottawa, Canada, September/October 2001.

[11] Yi J. Liang, E. Steinbach, and B. Girod. "Multi-Stream Voice over IP Using Packet Path Diversity," *Proc. IEEE Workshop on Multimedia Signal Processing*, pp. 555–560, Cannes, France, October 2001.

[12] Yi. J. Liang, N. Faerber, and B. Girod. "Adaptive Playout Scheduling and Loss Concealment for Voice Communication over IP Networks," *IEEE Transactions on Multimedia*, vol. 5, no. 4, pp. 532–543, December 2003.

[13] Wen-Tsai Liao, Jeng-Chun Chen, and Ming-Syan Chen. "Adaptive Recovery Techniques for Real-Time Audio Streams," *Proceedings IEEE INFOCOM 2001*, vol. 2, pp. 815–823, Anchorage, AK, April 2001.

[14] S. B. Moon, J. Kurose, and D. Towsley. "Packet Audio Playout Delay Adjustment: Performance Bounds and Algorithms," *Multimedia Systems*, vol. 6, no. 1, pp. 17–28, January 1998.

[15] C. Perkins, O. Hodson, and V. Hardman. "A Survey of Packet Loss Recovery Techniques for Streaming Audio," *IEEE Network*, vol. 12, no. 5, pp. 40–48, September/October 1998.

[16] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne. "Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks," *Proceedings IEEE INFOCOM '94*, vol. 2, pp. 680–688, June 1994.

[17] H. Sanneck, A. Stenger, K. Ben Younes, and B. Girod, "A New Technique for Audio Packet Loss Concealment," *IEEE GLOBECOM*, pp. 48–52, November 1996.

[18] Cormac J. Sreenan, Jyh-Cheng Chen, Prathima Agrawal, and B. Narendran. "Delay Reduction Techniques for Playout Buffering," *IEEE Transactions on Multimedia*, vol. 2, no. 2, pp. 88–100, June 2000.

[19] E. Steinbach, N. Faerber, and B. Girod. "Adaptive Playout for Low-Latency Video Streaming," *Proc. International Conference on Image Processing, ICIP-2001*, pp. 962–965, Thessaloniki, Greece, October 2001.

[20] A. Stenger, K. Ben Younes, R. Reng, and B. Girod. "A New Error Concealment Technique for Audio Transmission with Packet Loss," *Proc. EUSIPCO '96*, Trieste, Italy, September 1996.

[21] W. Verhelst and M. Roelands. "An Overlap-Add Technique Based on Waveform Similarity (WSOLA) for High Quality Time-Scale Modification of Speech," *Proc. ICASSP '93*, pp. 554–557, April 1993.

[22] Ondria J. Wasem, David J. Goodman, Charles A. Dvorak, and Howard G. Page. "The Effect of Waveform Substitution on the Quality of PCM Packet Communications," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 3, pp. 342–348, March 1988.

[23] Y. Xie, C. Liu, M. Lee, and T. N. Saadawi. "Adaptive Multimedia Synchronization in a Teleconference System," *Multimedia Systems*, vol. 7, no. 4, pp. 326–337, July 1999.

[24] M. C. Yuang, S. T. Liang, and Y. G. Chen. "Dynamic Video Playout Smoothing Method for Multimedia Applications," *Multimedia Tools and Applications*, vol. 6, pp. 47–59, 1998.

[25] ITU-T Recommendation P.862. "Perceptual Evaluation of Speech Quality (PESQ), an Objective Method for End-to-End Speech Quality Assessment of Narrow-Band Telephone Networks and Speech Codecs, February 2001.

PART **F**

ADVANCED TOPICS

This page intentionally left blank

17

Path Diversity for Media Streaming

The Use of Multiple Description Coding

John Apostolopoulos, Mitchell Trott, and Wai-Tian Tan

17.1 INTRODUCTION

Media streaming over best-effort packet networks such as the Internet is quite challenging because of the dynamic and unpredictable delay, loss rate, and available bandwidth. Streaming over multiple paths to provide path diversity, coupled with careful co-design of the media coding and packetization to exploit path diversity, has emerged as an approach to help overcome these problems. This chapter provides an overview of path diversity, of complementary media coding techniques such as multiple description coding, and of their benefits and uses for improved media streaming.

Path diversity is a transmission technique that sends data through two or more paths in a packet-based network. A path diversity system may use multiple paths at the same time or may perform path selection to switch between them. This is in contrast to the conventional approach where all packets are sent over a single path between sender and receiver, and this path does not vary with time under the direct or indirect control of the application. The paths may originate from single or multiple sources. An example of the single-source case in a video streaming application is shown in Figure 17.1. A more complex multiple-source scenario is illustrated in Figure 17.2.

Using multiple paths through the transport network for streaming can help overcome the loss and delay problems that afflict streaming media and low-latency communication. In addition, it has long been known that multiple paths can improve fault tolerance and link recovery for data delivery, as well as provide larger

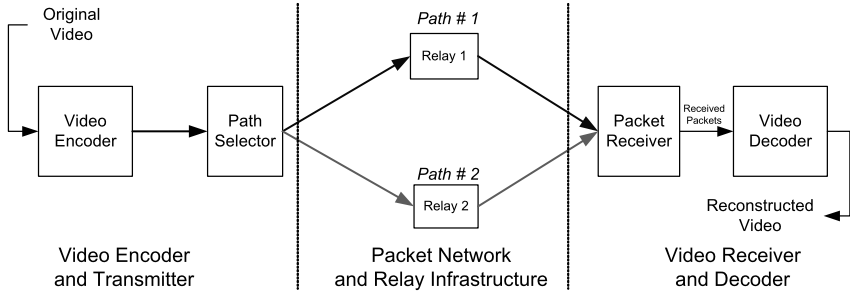


FIGURE 17.1: Media packets are sent over multiple paths in a packet network. In this case packets from a single source are directed over different paths via relays.

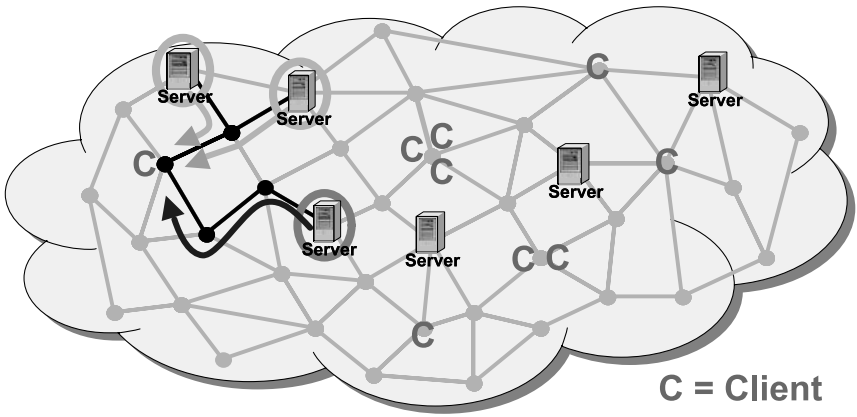


FIGURE 17.2: Path diversity in a content delivery network using multiple sources. A basic question: which two of the three circled servers should be selected to stream to the client? The two nearby servers that share a link or a nearby server and the more distant server?

aggregate bandwidth, load balancing, and faster bulk data downloads. The benefits of path diversity are examined in detail in Section 17.3.

Diversity techniques have been studied for many years for wireless communication, for example, frequency, time, and spatial diversity. However, path diversity over packet networks has received limited attention until relatively recently. The early work in this area, for example, [4,5,22], focuses on reliable data delivery

through error correction coding or retransmission and is generally performed in the context of multiple virtual circuits on connection-oriented systems such as ATM.

A number of thorough experimental studies, for example, [29], have demonstrated the great variability in the end-to-end performance observed over the Internet. This variability is analogous to the variability of wireless links and motivates the application of wireless diversity techniques to wired and wireless Internet communication. Further motivation for the use of multiple paths is that the default (single) path between two nodes on a network is often not the best path. For example, the measurement study [35] comparing the paths between two hosts on the Internet found that "in 30–80% of the cases, there is an alternate path with significantly superior quality" to the default path, where quality is measured in terms of metrics such as round-trip time, loss rate, and bandwidth.

This chapter provides a survey of the benefits, architectures, system design issues, and open problems associated with streaming media delivery using path diversity. Complementary media coding techniques such as multiple description coding are also reviewed. We begin in Section 17.2 by describing two basic components of the path diversity systems we consider: media streaming and media coding. Section 17.3 surveys the benefits of path diversity in a streaming-media context. Many of these benefits accrue only for media streaming applications and have no direct analogy, for example, to benefits associated with classical wireless diversity. Section 17.4 provides an overview of multiple description (MD) coding and its application to different types of media, including speech, audio, image, and video. Section 17.5 examines the design, analysis, and operation of media streaming systems that use path diversity and highlights some techniques for analyzing and modeling path diversity that are beneficial for selecting the best paths or best servers in a path diversity system. In Section 17.6 we describe various architectures that support and benefit from path diversity, including overlay networks, low-delay applications, peer-to-peer networks, and wireless networks. Finally, Section 17.7 provides a summary and pointers for further reading.

17.2 BUILDING BLOCKS FOR MEDIA STREAMING

In this section we introduce the basic components of media streaming, independent of any path diversity enhancements. We first describe how media streams differ from ordinary file transfer. We then describe how source material, such as audio or video streams, is transformed into a sequence of packets for transmission over a network. This encoding architecture is used in essentially all nondiversity applications and is extended in subsequent sections to support path diversity.

17.2.1 Media Streaming Characteristics

The problem of streaming media such as voice and video over best-effort packet networks is complicated by a number of factors. Unlike static content delivery or file download, streaming involves the simultaneous delivery and playback of media and is characterized by delay constraints on each transmitted packet. Packets that arrive after their decoding or display deadline are generally useless. These inevitable packet losses are not necessarily fatal; most media streams have some tolerance to packet loss, albeit limited by the use of temporally predictive compression. Delay constraints and tolerance to loss are primary factors that distinguish media streaming from ordinary data transport. Conversational or interactive applications have particularly tight delay constraints, typically 100 ms or less.

Conventional approaches for overcoming packet loss and network congestion in data delivery, such as reliable transport via TCP, generally are not possible for streaming because the persistent retransmissions cause too much delay and the rate adaptation is inappropriate for a constant-rate streaming source. Moreover, many applications (e.g., multicast or broadcast) lack a back channel or other means for retransmissions. Thus, meeting tight delay constraints in the presence of packet losses, queuing delays and network outages is quite challenging and provides motivation for some of the path diversity techniques that follow.

17.2.2 From Media to Packets

A representative packetization scheme is depicted in Figure 17.3. In most applications, the "media encoder and packetizer" module operates by first compressing the media frames into blocks of data, where the block boundaries are selected in some sensible manner to limit catastrophic parsing errors and error propagation at the decoder if blocks are lost. The media codec is optionally followed by some combination of forward error correction coding (FEC) and interleaving into transport packets. As the transport packets traverse the network some are lost or

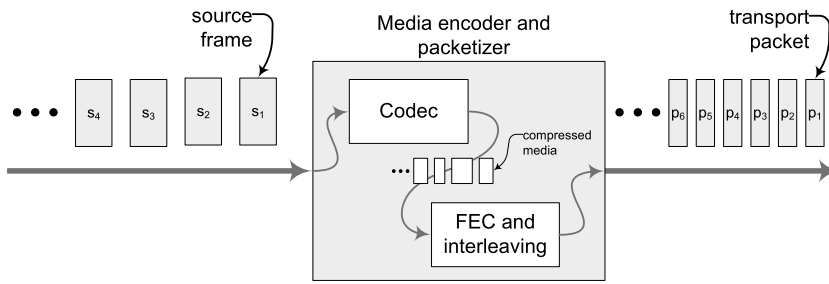


FIGURE 17.3: Generic media coding and packetization scheme.

delayed. The client must reconstruct and play out each frame of the source media by its play out deadline using the packets it has received thus far.

The separation of source and channel coding shown in Figure 17.3 is tremendously convenient for a host of reasons. For example, separate coder and error correction modules can be implemented in different hardware, can be designed by different standardization teams, can be upgraded, replaced, or reused separately, and so on. Even if the modules are implemented separately, however, significant benefits can accrue from designing them jointly. In particular, the precise manner in which coded media is distributed across multiple transport packets greatly impacts the quality of the reconstructed media after packet erasures. This point will be amplified in Section 17.4.

From a theoretical standpoint it's known that fully separating source and channel coding can hurt performance in some circumstances. The rather general scheme in Figure 17.3 has therefore already compromised potential performance to arrive at a simpler architecture. Nevertheless, essentially all modern packet streaming systems follow the block diagram shown in Figure 17.3, which is the structure on which we will concentrate.

17.3 BENEFITS OF PATH DIVERSITY

Streaming over multiple paths can mitigate three basic problem areas in networks: bandwidth, loss, and delay. In this section we characterize these benefits.

Path diversity to a single receiver can arise when streaming from one source or when streaming from several sources. Examples of the latter include streaming from multiple servers in a content delivery network or from multiple peers in a peer-to-peer system. In addition to path diversity, these applications also provide something that can be termed "source diversity," the benefits of which persist even on an uncongested network. For example, if hosts in a peer-to-peer system enter and exit then source diversity reduces the probability of service outage. Similarly, source diversity provides a load balancing benefit that can be important when the hosts are disk or CPU limited. Therefore, the use of multiple hosts, and associated multiple paths, leads to both host diversity and path diversity, where the largest benefit depends on where the bottleneck is: paths or hosts. While the discussion that follows is framed in terms of path diversity, it's useful to keep in mind the complementary benefits that can arise from using multiple hosts.

Leaving aside questions of fairness and resource consumption for the moment, a straightforward way to realize performance gains with multiple paths is to use all of them at once. For example, if in a peer-to-peer system the hosts are connected via cable modems or DSL connections with limited uplink bandwidth, path diversity with multiple sources provides much-needed bandwidth aggregation. This is shown in Figure 17.4a. Bandwidth aggregation can also be achieved with a single

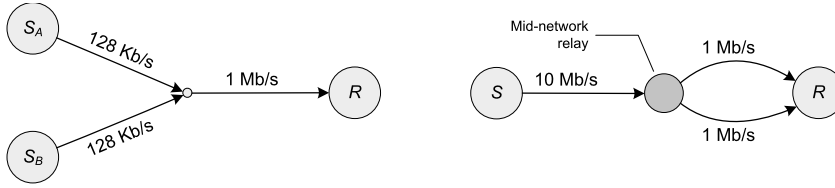


FIGURE 17.4: (a) Bandwidth is aggregated across paths using two streaming sources S_A and S_B. (b) Bandwidth is aggregated using one source and a mid-network relay. Paths are labeled with their available bandwidths.

source node, as in Figure 17.4b. A complementary benefit to bandwidth aggregation is traffic load balancing, that is, decreased per-path bandwidth by splitting a stream across multiple paths.

If bandwidth is not a primary concern, delay and loss may be reduced by replicating each packet across the paths. The receiver then sees the minimum of the path delays (effectively chopping off the long tails in the end-to-end delay distribution) and is immune to packet losses unless they occur on all paths simultaneously.

A more challenging problem—and the main focus of this chapter—is to realize performance gains in the presence of delay and loss without doubling the number of packets transmitted. How this is done depends on the amount of information the sender has about prevailing link conditions.

Perhaps the simplest way to realize gains without consuming extra bandwidth is through path selection. Selection diversity arises when there are multiple paths available, when the sender knows which path has the most favorable characteristics, and when the sender can respond in a timely manner to direct packets along the currently most favorable path. A simple example is shown in Figure 17.5a. The benefits of selection diversity are often quite large, so large that even approximate knowledge of link conditions suffices to realize gains. Thus, when a single-path system is unreliable, selection diversity should be one of the first techniques considered.

Often, however, either the sender does not have detailed knowledge about the current state of the paths or it cannot take advantage of its knowledge. This occurs in a variety of situations, including:

1. time-invariant paths that lack feedback,
2. time-varying paths where the path measurement system lags the variation, and

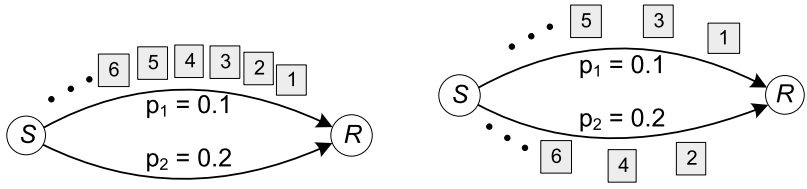


FIGURE 17.5: (a) Selection diversity directs packets over the path with the smallest loss rate. (b) The sender responds to its uncertainty by sending odd packets on the upper path and even packets on the lower path. The average loss rate is more predictable than the loss rate in a single-path system that selects a path at random.

3. broadcast or multicast scenarios in which a single transmission strategy serves all users at once.

Selection diversity alone is not a viable approach in these cases, and the sender can take advantage of path diversity only through more sophisticated approaches.

An example of path diversity in which two time-invariant paths have differing loss rates is shown in Figure 17.5b. If the sender is unsure about which path is best it can hedge its bets by dispersing media packets across the paths. The receiver then sees the average $(p_1 + p_2)/2$ of the loss rates.

Averaging loss rates illustrates a significant benefit of path diversity: reduced uncertainty. Reduced uncertainty enables methods for combating losses, such as forward error correction, to become more effective [25,26]. FEC adds specialized inter-packet redundancy that enables data recovery up to a loss threshold. Compared to a single-path system that selects from a set of paths at random, averaging loss rates improves the probability that the overall loss rate lies below the critical threshold (in most cases of interest).

For time-varying links, for example, links that shift between periods of no loss and high loss that are common on the Internet, the amount of redundancy can be adjusted dynamically to compensate for changes in the loss rate. However, adaptation is problematic when the network changes quickly: the FEC is inevitably overdesigned and therefore inefficient or underdesigned and therefore ineffective. Combining FEC with long interleaving also helps combat loss variability, but the added delay often makes interleaving unsatisfactory for media streaming. When used with FEC, path diversity provides benefits similar to time interleaving with a smaller associated delay.

Path diversity can dramatically decrease the probability of outage, where an *outage* is an extreme form of time variation in which all communication along a network path is lost for a sizable length of time. A single-path system necessarily

fails during an outage. However, with two-path diversity the average loss rate becomes 50%, which is tolerable for certain codecs (see Section 17.4). A service outage then occurs only if both paths fail at once. In heuristic terms, path diversity improves the probability of outage from p to p^2.

Problems of delay and delay jitter may also be addressed using path diversity. If the sender distributes packets across paths the receiver sees the mixture of the delay distributions of the paths. This effect is termed *queue diversity* because network delays are often due to backlogged queues, and to emphasize the benefit of multiple parallel queues. Queue diversity offers little benefit for loss-intolerant data because the receiver must wait for all packets from all paths. In contrast, loss-tolerant delay-intolerant voice or video streams remain useful when only some packets arrive promptly. Similar to loss-rate averaging, queue diversity is helpful for time-invariant paths that have different but unknown delay characteristics. It's also helpful when the paths are time varying, for example, when cross traffic causes episodes of high delay that strike the paths at independent times. A typical application of queue diversity is to allow end-to-end delay constraints to be tightened while maintaining quality. Examples of such delay reductions may be found in voice over IP (VoIP) [20], as described in Chapter 16, and video over 802.11 wireless networks [23], as described in 17.6.4.

Path diversity can also reduce the length of burst losses, that is, losses of consecutive packets. Distributing packets across multiple paths increases the interpacket spacing on each path, and therefore for a network congestion event of a given duration fewer packets are lost. This is illustrated in Figure 17.6. If all paths

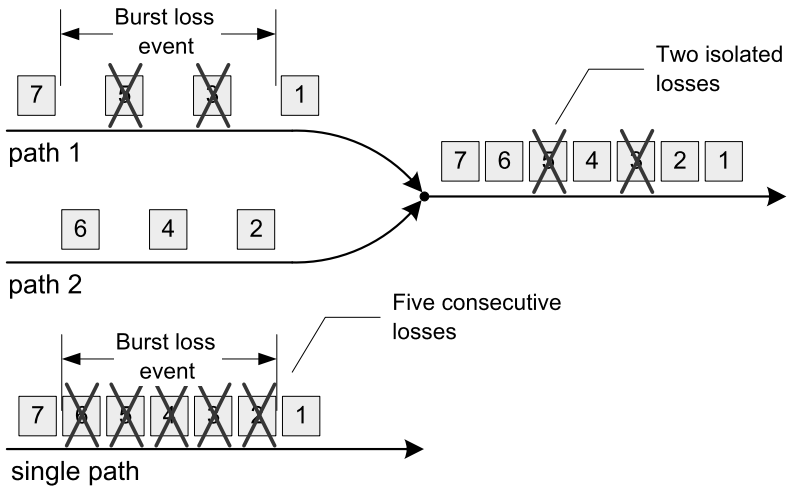


FIGURE 17.6: Path diversity reduces the length of burst losses.

are subject to burst losses, however, long burst events are in effect replaced with a larger number of shorter events. Whether this trade-off is advantageous depends on the media coding. If latency is not critical, then using FEC over a sufficiently long block will make the decoder largely indifferent to burst vs. isolated losses. In contrast, we will see in Section 17.4.3 an example of a simple low-latency speech coding strategy that is resistant to isolated but not burst losses. In addition, for video codecs it can sometimes be easier to recover from multiple isolated losses than from an equal number of consecutive losses [1]; for example, a gain of 0.5–1.0 dB in video quality is achievable in certain situations [23]. Still greater gains are possible by carefully codesigning the encoder, decoder, and path diversity system, as we shall see in Section 17.4.6.

17.4 MULTIPLE DESCRIPTION CODING

In this section we introduce multiple description coding, a media coding technique that is quite appealing when multiple transport paths are available. An excellent review of MD coding, both history and theory, is given in [13].

Multiple description coding produces two or more sets of compressed data, referred to as *descriptions*, as in Figure 17.7. In broad terms, MD coding permits a usable reproduction of the original signal to be reconstituted when only some of the descriptions are available, at the decoder. The more descriptions available, the better the quality of the reproduction. For example, a simple MD video coder can be achieved by splitting a video stream into even and odd frames and coding them separately.

One way to apply multiple description coding to packet networks is for each packet to contain one of the descriptions in the MD code. The resulting system can be tuned to exhibit a graceful degradation of media quality with the number

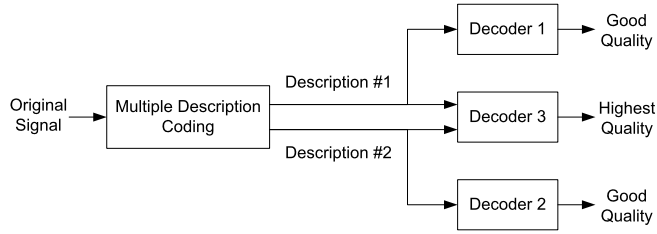


FIGURE 17.7: A basic type of MD coder produces two descriptions of roughly equal importance. If a decoder receives either description it can reconstruct a good quality signal, whereas if it receives both it can reconstruct the highest quality signal.

of packets lost. Used in this manner, MD coding exemplifies a design philosophy that treats unpredictable variations in packet loss rate as inevitable and aims to make systems robust to a range of loss conditions. A contrasting approach, which does not employ MD coding, is to compress media into a loss-sensitive packet stream and use retransmission or forward error correction to recover from losses. The MD approach is particularly advantageous when the loss rate is varying or unknown and when latency constraints hinder the use of retransmissions.

MD coding is also useful for systems that employ path diversity. The source is coded into two or more data streams and each stream is viewed as a description to be sent along a different path. With this approach, the coding system may be designed to tolerate losses that differentially impact one stream, for example, where one stream experiences 20% packet losses while the other is largely loss free. Examples illustrate MD codes that are effective in this scenario.

Note that in some contexts a "description" implies a single packet, while in other contexts it refers to an entire stream of packets. Both cases exist in the literature. In the case of path diversity, the two viewpoints can be unified by treating each packet as a description while recognizing that the packet loss patterns have a special structure. In particular, for path diversity, packet losses are concentrated on the subset of packets transmitted over a failing path, while for systems with a single path, packet losses are distributed more uniformly.

We continue by providing a high-level comparison of MD coding versus conventional single description (SD) and scalable coding techniques, which were described in detail in Chapters 2–6. We then give an overview of the information theory results that apply to MD coding, all of which take the one-description-per-packet approach. We next examine the salient features of several practical examples of MD coding for speech, audio, image, and video. These examples illustrate how the one-description-per-packet and one-description-per-stream approaches lead to different coding techniques. We will also see that MD coding can be implemented through special source codecs, through a joint design of the source codec with interleaving, or through a joint design of codec, interleaver, and FEC.

17.4.1 Comparing MD Coding with SD and Scalable Coding

Conventional SD coding algorithms, such as MPEG-1/2/4 and H.261/3/4 video coding standards, compress media into a single bit stream. In scalable (also called layered) coding, media is coded into multiple bit streams, beginning with a base layer that provides low but usable quality, and one or more enhancement layers that improve quality. The scalable coding bit streams are partially ordered, and different subsets can, for example, represent video at different spatial, temporal, or amplitude resolutions and fidelities.

Applications and networks that support prioritization can exploit scalable coding by assigning higher delivery priorities to coding layers that are more important. However, in best-effort networks, such as the Internet, all packets are equally likely to be lost or delayed. This fundamental mismatch—prioritized data on a nonprioritized network—makes scalable coding by itself difficult to exploit using the Internet. One solution to this problem is to combine scalable coding with channel coding. For example, erasure-correction coding can be used to make the base layer more tolerant to packet loss than the enhancement layer(s). This approach will be described in more detail in Section 17.4.4.

Multiple description coding differs from scalable coding in at least one important way: scalable coding has a base layer that is critically important and if lost renders the other bit stream(s) useless. MD coding enables a useful reproduction of the signal when any description is received (when following the one-description-per-stream approach) or when any sufficiently large set of descriptions is received (when following the one-description-per-packet approach).

17.4.2 MD Coding: Information Theory Perspective

The multiple description coding problem has received significant attention in the information theory literature, beginning in 1980 with [27,44–46]. For recent perspectives and comprehensive bibliographies, see, for example, [13,18,30,31].

The basic information theory problem in multiple description coding is that of compressing a block of independent identically distributed (i.i.d.) random variables, typically Gaussian or binary, into a set of packets for transmission over an erasure channel. The simplest case, which turns out to be far from trivial, encodes into just two packets. Encoding and decoding are done without regard to delay, hence the theory is more directly relevant to problems such as still image compression than to streaming media. The generalization of multiple description information theory to problems that incorporate delay constraints is an important (and challenging) area for future research.

A general theme of multiple description information theory research is to characterize the achievable distortion as a function of erasure patterns. In the case of a source encoded into two packets—or equivalently, into two descriptions—we'd like the quality of the recovered source to be good when just one of the two packets survives the network and to improve when both packets survive. One bound on this problem comes from classical rate-distortion theory: the distortion–rate function $D(R)$ for the source gives the minimum achievable average distortion D when the source is described using R bits per source symbol. For example, the distortion–rate function for a unit variance i.i.d. Gaussian source where distortion is measured by mean-squared error is $D(R) = 2^{-2R}$. Applied to the multiple description problem with two packets, assuming each packet contains R bits per

source symbol, the best that can be achieved when one packet survives is 2^{-2R}. The best that can be achieved when both packets survive is 2^{-4R}.

But are these naive bounds useful? That is, can a multiple description compression algorithm simultaneously achieve both single- and dual-description upper bounds $D(R)$ and $D(2R)$? The answer is no: these bounds are wildly optimistic when the rate R is large (i.e., much above 1 bit per symbol). For example, if in the Gaussian case the single description meets the $D(R)$ bound, the joint description can be no better than roughly $D(R + 1/2)$, which is quite a bit worse than $D(2R)$ at high rates.

A sensible engineering compromise in this situation is to leave some slack in the system. Rather than trying to meet either the $D(R)$ or the $D(2R)$ bounds exactly, one should instead aim for reasonable but not ideal performance for both single- and dual-description cases. The exact set of available trade-offs in the Gaussian case with mean-squared distortion was determined by Ozarow [27]; useful plots of this trade-off and a more detailed discussion may be found in Goyal [13].

The Gaussian case is noteworthy for being the only nontrivial example where the multiple description trade-off curve is known exactly. Even for the apparently simple case of binary source with Hamming distortion the exact curve is unknown. Simple arguments suggest, however, that real-world sources will have the same characteristic trade-offs as the Gaussian source, hence the Gaussian case remains a useful benchmark.

Generalizations of multiple description coding from two descriptions to many descriptions are treated in [30,31,39]. A complete solution to this problem would allow us to answer the following type of question: suppose a channel has either 30% packet erasures or 60% packet erasures, but the encoder doesn't know which. How should the source be coded to prepare for these two contingencies, and what distortions can be achieved? Broadly speaking, a system designed aggressively to do as best as possible with 30% erasures will fare quite poorly when confronted with 60% erasures. Conversely, a conservative design that does as well as possible with 60% erasures will see only modest performance improvements when there are only 30% erasures. The achievable trade-offs for this problem have been bounded but are not precisely known.

The relative lack of theoretical results about the trade-offs inherent in multiple description coding presents no obstacle to a practical exploration of the space. There is growing literature on practical schemes for multiple description media coding for both streaming and nonstreaming applications, as discussed next. While many of these schemes have good performance or insightful architectures, one should keep in mind that they represent particular achievable points in the space of trade-offs; only in special scenarios is the ideal performance known, let alone approached closely. Thus there remain many interesting opportunities to find improved solutions.

17.4.3 MD Speech and MD Audio Coding

The earliest MD coders were applied to speech signals. For example, in [16,17] speech is partitioned into speech frames of (say) 16 ms duration, and each speech frame is split into even and odd samples, which are then coded independently and sent in separate packets. This coding method protects better against certain loss patterns than others. For example, losing both the even and the odd packet from one speech frame is likely to sound worse than losing an even and an odd packet from different frames. The even/odd coding strategy is therefore most effective when the two packet streams are sent along different paths that suffer independent outages.

A variety of other simple MD techniques exist. For example, each packet can contain the current speech frame and a low-quality copy of the previous speech frame, as in Figure 17.8. In another variation, a packet containing the coded even (odd) samples may also contain a coarsely coded version of the odd (even) samples. This is illustrated in Figure 17.9, where the packets containing the even samples are sent over one path and the packets containing the odd samples are sent over a second path. In this way, as long as either of the two packets for each speech frame is received the decoder can reconstruct a good version of half of the samples and a degraded version of the other half. Further discussion on the use of MD speech coding is given in Chapter 16 and in [20].

In most speech and audio coders the source frames (as in Figure 17.3) are coded largely independently of the neighboring speech or audio frames. This is true for most SD, scalable, and MD speech and audio coders. The loss of one speech or

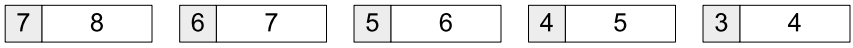


FIGURE 17.8: Packets for audio frames 4–8 contain low-quality copies of frames 3–7, respectively. Loss of audio frame 5 will not cause a playback gap if frame 6 is received. However, a playback gap will result if both frames 5 and 6 are lost.

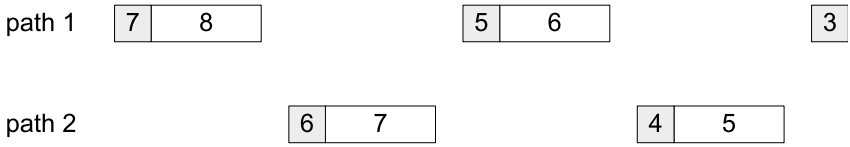


FIGURE 17.9: Burst loss in either path 1 or path 2 alone will not cause a gap in playback. This is an example in which path diversity does not reduce loss rate, but provides a more usable set of data for the application.

audio frame therefore affects a small portion of the reconstructed signal. This is in contrast to conventional video coding where temporal prediction is applied between video frames, and a single packet loss can lead to significant error propagation, which affects many frames.

17.4.4 MD Image Coding

MD codes for images can be constructed using methods from source coding or by combining source coding with FEC. A common source coding approach is to subsample in the spatial or frequency domains. Straightforward subsampling, however, provides no control over the trade-off between single- vs. multiple-description image quality. In MD transform coding (e.g., [41]) a correlating transform introduces controlled redundancy between the subsets of coefficients, which enables such trade-offs.

MD codes may also be created by coupling a scalable coder with FEC to provide unequal error protection (UEP) for the scalable layers, in a manner sometimes referred to as MD-FEC [32]. While MD-FEC was first proposed in the context of MD image coding and is appealing in this context because high-quality scalable image coders such as JPEG-2000 are available, it's applicable to any type of media that's scalably coded. In this technique, the source coder does not produce multiple descriptions directly, but the combination of scalable coding and UEP produces a set of packets that have the MD property. This allows one to turn a scalable, prioritized bit stream into a nonprioritized one that is better matched to a best-effort packet network such as the Internet.

An example of MD-FEC is shown in Figure 17.10. In Figure 17.10 a scalable code with two layers—a base layer and a single enhancement layer—is used to construct an M-description MD code. Each of the M descriptions is contained in a separate packet. The base layer is split into m equal-sized blocks, and $M - m$ parity blocks are computed using, for example, an (M, m) systematic Reed–Solomon code. In this manner the base layer is expanded into M descriptions, any m of which allow the receiver to recover the layer perfectly. The enhancement layer in this example has no FEC, hence all M descriptions are needed to recover it perfectly. An important feature of MD-FEC is its flexibility: it can be used with any number of scalable layers, variable amounts of FEC per layer, and an arbitrary number of descriptions.

17.4.5 MD Video Coding

Multiple description video coding has some attributes of the audio coding and image coding cases considered earlier. As with audio, coded video consists of a sequence of packets, and the manner in which burst losses interact with this sequence has important consequences for encoder and decoder design. As with

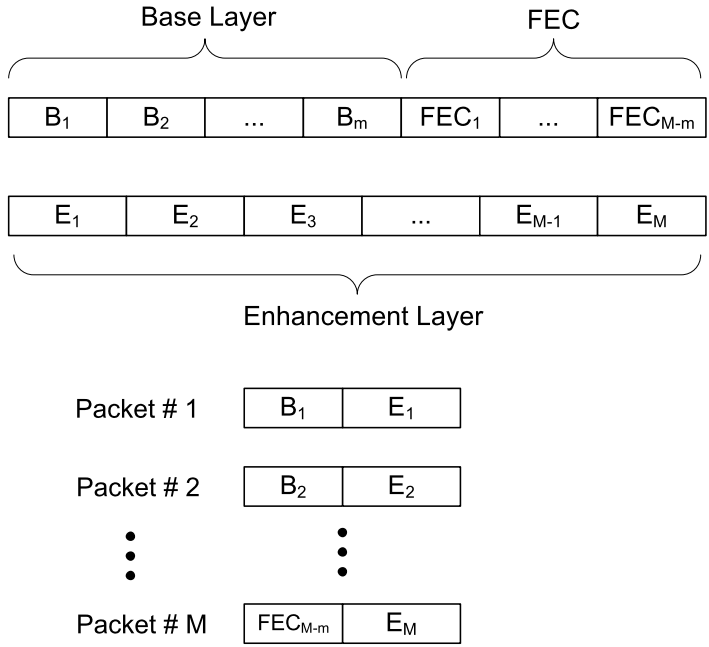


FIGURE 17.10: In MD-FEC a signal is scalably coded into two or more layers that are spread across M packets, with unequal cross-packet FEC for each layer. In this example, if any m packets are received, the base layer can be recovered perfectly; if all M packets are received, both layers can be recovered perfectly.

images, each packet can be thought of as a description, and one can trade off performance when few descriptions are lost vs. performance when many descriptions are lost.

Video sequences do, however, have special characteristics not present in image and audio sources. Consecutive video frames tend to be similar, and the most common approach to exploit this redundancy for compression is to apply some form of predictive coding along the temporal direction. For example, the MPEG-1/2/4 and H.261/3/4 video compression standards all employ motion-compensated prediction between frames. Predictive coding is based on the assumption that the encoder and decoder are able to maintain the same state, that is, that the frames used for forming the prediction are the same at the encoder and decoder. However, packet losses can cause a mismatch between the states at the encoder and decoder. This mismatch can lead to significant error propagation into subsequent frames, even if the packets corresponding to those frames are correctly received.

For example, a single lost packet can impair the video quality of tens or hundreds of subsequent frames, until the prediction is reinitialized with an I frame.

The design of an MD video codec depends on the loss patterns one expects to encounter. In the case of path diversity, a conservative design goal is to maintain good quality even when half the data is lost, for example, when a path permanently fails. A more aggressive target that leads to greater compression efficiency is to provide resilience to burst or isolated losses, provided they do not occur on all paths at the same time. The latter scenario commonly arises when the paths have independent losses or when the loss statistics on the paths are different, for example, when one suffers isolated 1% packet loss while another suffers burst losses at an overall rate of 10%.

A variety of MD video coding techniques suitable for these and other packet loss scenarios are based on varying the prediction between frames, for example, [1,33,40,43], where these techniques are motivated by the importance of predictive coding in most video coders today. MD video coding techniques that apply motion-compensated filtering between frames, as often used for scalable video coding, have been proposed, and these techniques can provide the advantages of both MD and scalable coding, for example, [38]. An excellent review of MD video coding is given in [42]. For simplicity, the following discussion assumes that we code the video into two packet streams to be transmitted over two paths in a packet network.

One type of predictive MD coding emphasizes the case where a packet stream decodes with reasonable quality even when the other stream is completely lost. This can be accomplished by using independent prediction loops for each stream. For example, two separate prediction loops may be used so that even frames are predicted based on past even frames and odd frames are predicted based on past odd frames. (A standard-compliant way to achieve this is to use multiple reference frames, as supported in H.263v2, MPEG-4, and H.264.) Independence between streams can also be achieved by having a single prediction loop, but duplicating the information required to form the prediction in each stream to ensure that if packets from either stream are received the receiver can form the required prediction. These approaches allow a good quality reconstruction when one packet stream is received and one is lost (e.g., the video is reconstructed at half the original frame rate); however, they suffer a sizable penalty in compression performance since the two streams are coded independently of each other.

On the other extreme are predictive MD coders that try to maximize the coding efficiency when both packet streams are received. In particular, they use a single prediction loop in a manner similar to single description coding. If both streams are received they provide excellent quality; however, the prediction requires information from both packet streams and if either is lost then mismatch occurs with subsequent error propagation. There are also predictive MD coders that try to operate between these two extremes, providing the ability to trade off compres-

sion efficiency (when both streams are received) versus resilience to the full or partial loss of a stream. An active area of research is designing MD coders that automatically adjust the compression efficiency versus resiliency as a function of the channel characteristics to maximize the expected quality at the receiver, for example, [15,40].

Video, like audio, is frequently used in situations that require low latency. If latency is not a primary concern, one approach to MD video coding is to carefully combine a scalable video coder with UEP in a manner analogous to MD image coding (Section 17.4.4). This technique provides a straightforward way to produce many packet streams. In contrast, the predictive MD video coding methods described earlier are generally limited to a small number of description streams (typically two) because compression efficiency decreases with the number of streams. It is worth noting that while predictively coded video has historically provided significant compression benefits over scalable coding, and hence the widespread use of predictive coded video for both practical applications and MD video coder design, this may change in the future with some of the emerging scalable video coding techniques, as discussed in Chapter 5.

17.4.6 Repairable MD Coding

Repairable MD coding (also known as *state recovery* in MD coding) is an extension of conventional MD coding beneficial for temporally predictively coded media such as video [1]. Predictive coding causes errors in one frame, due, for example, to lost packets, to propagate to potentially many subsequent frames. In repairable MD coding the decoder attempts to stop error propagation by repairing the corrupted frames in one description using uncorrupted frames from the other description. This technique can maintain usable quality even when *both* descriptions suffer losses, as long as both descriptions are not simultaneously lost.

Figure 17.11 illustrates the impact of packet loss on a conventional SD MPEG-type video coder and on a repairable MD coder. The particular MD video coder in Figure 17.11 forms the first description by predicting even frames from previously coded even frames and forms the second description likewise from odd frames. For both SD and MD examples, the packet(s) that describes how frame 3 is used to predict the subsequent coded frame is assumed lost. For SD, the packet that predicts frame P_4 from P_3 is lost, and the decoder estimates the missing frame 4 using the last correctly decoded frame (frame 3), where the inaccuracy leads to error propagation that continues until the next I frame. The repairable MD decoder estimates the lost frame by bidirectionally predicting it (that is, interpolating it) from neighboring frames in both descriptions. The use of both previous and future frames to estimate the lost frame provides significant improvements in accuracy as compared to the use of only previous frames.

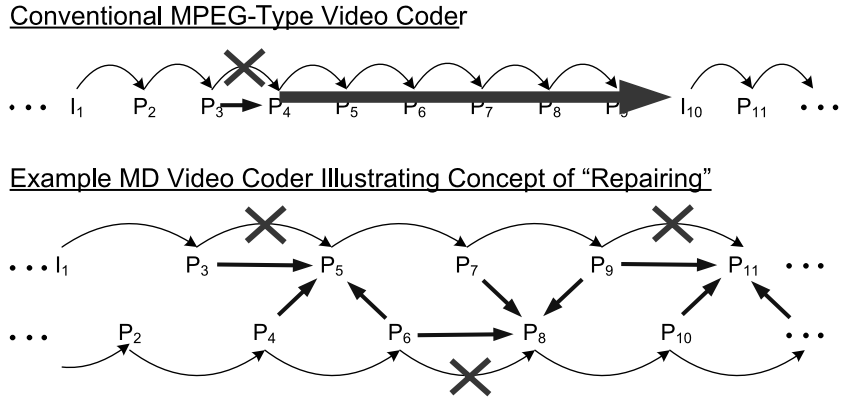


FIGURE 17.11: Benefits of repairable MD coding to reduce error propagation. The arrows show the prediction dependencies between frames, and each "X" denotes packet loss that leads to an error in the following predicted frame. The conventional SD coder exhibits error propagation that continues until the next I frame. The repairable MD decoder limits error propagation by repairing the missing or damaged frame by estimating it using neighboring previous and future frames from both descriptions.

Repairable MD coding and path diversity complement each other to improve the effectiveness of MD coding: the path diversity transmission system reduces the probability that both descriptions are simultaneously lost, and the MD decoder enables recovery from losses as long as both descriptions are not simultaneously lost [1]. This is illustrated in Figure 17.11 where the MD coder is afflicted by three losses affecting both descriptions—without repairing then both descriptions would be corrupted and the MD video quality would likely be no better than the SD quality. This example also highlights the importance of considering the effect of partial losses of both descriptions, in particular since even a single lost packet can lead to significant error propagation that corrupts a description for a long length of time. This comparatively recent decoder-side optimization is complementary to the ongoing efforts to improve the rate vs. quality trade-off of MD encoders, which typically do not consider partial losses of both descriptions.

17.4.7 Comments on SD, Scalable, and MD Coding for Media

The relative compression efficiencies of SD, scalable, and MD coding depend on the media type. For image and audio sources, scalable coding can be as efficient as SD coding, and many recent coding standards, for example, JPEG-2000, are

scalable. In addition, scalable image or audio coding may be combined with unequal error protection to provide an efficient MD-like system: more redundancy is allocated to high-priority data relative to low-priority data, improving the probability that high-priority data are correctly received. For video sources both scalable coding and MD coding are currently less efficient than SD coding; however, they provide valuable properties for streaming and, as a result, there is intense research underway to improve their performance and reduce the compression efficiency gap.

17.5 DESIGN, ANALYSIS, AND OPERATION OF MULTIPATH STREAMING SYSTEMS

The prior sections discuss the potential benefits of path diversity for media streaming and describe media coding techniques useful for path diversity. This section discusses issues that arise in effectively designing and using path diversity in streaming media systems.

17.5.1 Joint and Disjoint Paths

Multiple paths are not guaranteed in practice to be independent and may, for example, share links. The benefits of path diversity do not depend on whether paths are completely disjoint, but rather on whether bottlenecks occur on shared or disjoint portions. Shared bottlenecks reduce the impact of path diversity, while disjoint bottlenecks do not. Identifying bottlenecks and avoiding them if possible are important elements in effective use of path diversity. Joint bottleneck detection is an active area of research, for example, [34].

17.5.2 How Many Paths to Use?

How do diversity benefits scale with the number of paths and when is it worthwhile to use more than one path? These complicated questions depend on the specifics of the application, the path diversity benefits that one is trying to exploit, and the characteristics of the available paths. For example, for repairable MD video coding, the improved error recovery arises with two paths, and while increasing the number of paths helps there are no further jumps in performance.

End-to-end network characteristics improve in different ways with the number of paths. Total aggregated bandwidth (in principle) increases additively with the number of paths. Probability of outage decreases exponentially with the number of independent paths. Delay variability, measured in terms of standard deviation, decreases as $1/\sqrt{N}$ where N is the number of independent paths. For media streaming in a real-world scenario with reasonably reliable networks and servers, we would expect that typically a small number of paths would provide

a good balance between complexity and performance. In a peer-to-peer scenario in which hosts frequently enter and leave, a much larger number of paths could prove useful.

17.5.3 Selecting the Best Paths or Best Servers

As discussed earlier, the effectiveness of using multiple paths depends less on their lengths than on their shared vs. disjoint topology. Of still greater importance is whether bottlenecks occur on shared portions. Thus, it may be preferable to have longer paths or longer shared portions of paths if the resulting bottlenecks are not shared. This observation hints at the complexity of the important basic question: How to select the best paths to use?

Determining the best paths is important for point-to-point multipath streaming, but its impact becomes even more apparent when streaming from multiple servers. Which of several available servers should be selected? For a single-path system one typically selects a server that is in some sense nearby. However, a multiple server system requires fundamentally different metrics: minimizing distance from the client to multiple servers while maximizing path diversity, two generally conflicting objectives. Accurately evaluating these metrics is necessary for designing and operating a multiple server system that uses path diversity. In addition, the server selection problem must be solved for every client request, necessitating efficient solutions. Further characteristics of multiple server systems are provided in Section 17.6.2.

17.5.4 Modeling Path Diversity Performance

The aforementioned discussion highlights the importance of accurately modeling the performance of a path diversity system. An accurate model is needed for selecting the best subset of paths from a set of possible paths, selecting the best subset of servers from a set of possible servers, comparing path diversity scenarios, or simply evaluating the merits of a path diversity system relative to a conventional single path system. Determining an appropriate performance model depends on the benefits (reviewed in Section 17.3) one intends to capture and on the particular characteristics of the media and the network.

In the following we describe a model for predicting SD and MD video quality over a lossy packet network as a function of path diversity and loss characteristics. The model has two parts, one that provides a probabilistic model of the loss patterns that occur in the network and another that quantifies how packet losses reduce video quality. See [2,3] for complete details.

The sender(s) and receiver in a path diversity system may be connected through a wide range of complex topologies that include joint and disjoint links. It can be shown, however, that under certain assumptions the end-to-end properties of

the network are captured using the dramatically simplified three-link topologies shown in Figure 17.12. The bursty losses on each link are modeled using, for example, a two-state Gilbert model.

Packet losses affect SD and MD video differently. The model must therefore distinguish among isolated packet losses, burst losses of various lengths, and for MD whether the loss(es) occurs on one or both descriptions at the same time. For example, for SD we may have a finite-state model that accounts for the packet loss burst length, and for MD a four-state model that represents for each pair of packets transmitted whether both descriptions are correctly received, one is correctly received and one is lost, or both are simultaneously lost. Each state transition causes an incremental change in video quality determined by the details of the particular codec. The state transition probabilities are derived from the loss model described in the previous paragraph.

An application-level performance model of this form provides insights into fundamental questions in path diversity, for example, the relative performance of conventional SD over a single path versus MD with two-path diversity. Using the model, one can determine that MD with path diversity is beneficial for video streaming when the packet losses fall predominantly on disjoint links, while SD sometimes performs better when the losses are on joint links (see [2,3]).

Additional work includes [10], which investigates rate-distortion optimized packet transmission schedules across multiple paths, [8], where fast heuristics are presented for quickly performing path selection when a large number of candidate path pairs are available while accounting for the loss and on-time arrival probability of each path, and [42], which provides an overview of MD video coding and its transport over multiple paths.

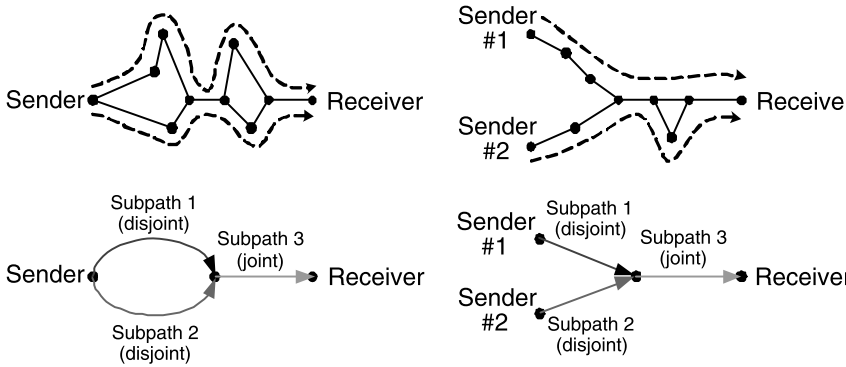


FIGURE 17.12: Complex path diversity topologies (top) and the simplified network models (bottom), which preserve the end-to-end characteristics important for accurately predicting video distortion.

17.5.5 Streaming and Packet Scheduling Across Asymmetric Paths

Different paths may offer different bandwidths, loss rates, and delay characteristics. A path diversity system should compensate for and exploit these asymmetries. The first step is to estimate the characteristics of each path. Path measurement is an active area of research, and many techniques developed for single paths may be applied to the multiple path case.

When paths have different, potentially time-varying, available bandwidths, it is important to adapt the rate across paths. In the point-to-point case, rate adaptation can be performed in a centralized manner. However, in the multipoint-to-point case, a distributed algorithm is generally required. The distributed rate allocation should not only perform the appropriate adaptation, but also for SD coding should ensure that the different senders send disjoint sets of packets (no duplication) to the receiver [25].

Paths with different packet loss rates or delays provide the opportunity to perform packet scheduling across the paths [10]. For example, more important packets may be sent over the path with the lower packet loss rate. Similarly, packets with tighter delay constraints may be sent over the path with shorter delay. Furthermore, real-time video encoding can be adapted to react to the time-varying path characteristics and losses [19].

17.6 APPLICATIONS AND ARCHITECTURES

In this section we examine a variety of application areas for which path diversity provides performance benefits: low-delay applications, content delivery networks, peer-to-peer networks, and wireless networks. These application areas also illustrate architectures for realizing path diversity, for example, by using mid-network relays, distributed content, or explicit routing control. We summarize these architectures at the end of the section.

17.6.1 Low-Delay Applications

Many applications, such as video conferencing and VoIP, require low latency for effective interactive communication. One possible approach used to achieve a latency guarantee is through network quality-of-service mechanisms. Nevertheless, schemes such as Skype [7] that operate over best-effort networks have shown to be effective most of the time. As illustrated in Figure 17.13, in order to bypass firewalls and network address translators (NAT), Skype uses relay nodes in the public Internet. The existence of multiple available relay nodes improves its ability to choose a low-latency relay path. This scheme exploits the existence of multiple paths via *selection*, or choosing a good or the best path. Generally, knowledge of network states is imprecise, and selection works best when network statistics

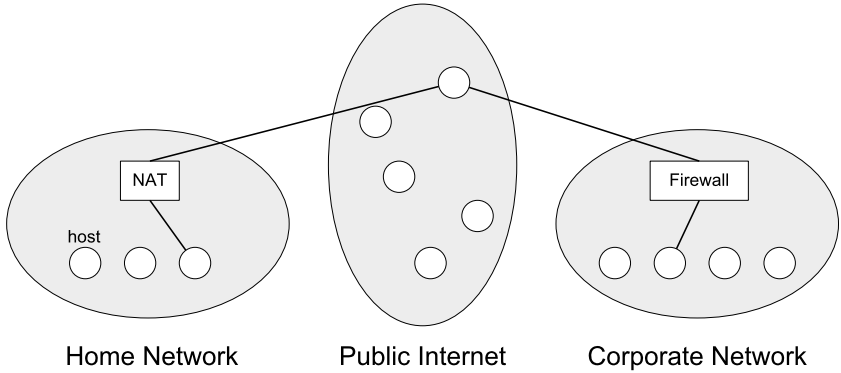


FIGURE 17.13: Skype uses nodes in the public Internet to bypass firewalls and NATs. Multiple relay paths are available for selection.

do not change rapidly. These issues are discussed in more detail in Sections 17.3 and 17.5.

Another consequence of low-latency requirement is the limitation in choices of error recovery mechanisms. Specifically, common and effective techniques such as retransmissions, interleaving, and forward error correction tend to introduce additional latency and may be unacceptable. One practical approach for error recovery in voice communication is to append to each voice packet a low-quality version of the previous packet, as illustrated in Figure 17.8. This technique avoids gaps in audio playback for isolated losses; however, by itself it is ineffective for burst losses. By simply alternating packets across multiple paths, or transmitting the low-quality version on a separate path, burst losses can be effectively reduced to isolated losses. This scheme exploits the *independence* of multiple paths. Notice that from the network's perspective, the total number of packets sent and delivered remains the same. However, from the application's perspective, the set of received packets from multiple paths is far more usable. Chapter 16 examines the problem of low-latency communication in more depth and shows how the combination of path diversity and adaptive media playout, where the media playout rate is varied as a function of the receiver's playout buffer fullness, can lead to sizable improvements in user-perceived quality.

17.6.2 Content Delivery Networks

A content delivery network (CDN) is a set of hosts inside a network that cooperate to improve content delivery by performing functions such as caching, content serving, and traffic relaying. CDNs have been widely used to provide low latency, scalability, fault tolerance, and load balancing for the delivery of web content

and more recently streaming media. Figure 17.14 illustrates the operation of a CDN. By locating content close to the users, a CDN improves access latency, lowers packet loss rate, and reduces traffic demand. CDNs are described in detail in Chapter 19.

The replication of content at multiple servers at different locations provides a number of benefits. First, it eliminates the bandwidth bottleneck associated with a single server or the network at a single location, thereby improving scalability to support a large number of clients. Second, it naturally allows delivery of content using multiple paths from multiple servers. Unlike the relay example, where multiple paths are provided between a single sender and a single receiver via multiple relays, in the context of a CDN, the different paths correspond to different senders, as shown in Figure 17.2.

The currently prevalent manner in which CDNs exploit multiple paths is via selection, that is, choosing a good or best server. Nevertheless, simultaneous use of independent paths may provide additional gains. For example, appropriately coupling MD coding with a CDN can provide improved reliability to packet losses, link outages, and temporary server overload or server failures. This system is referred to as a Multiple Description Streaming Media Content Delivery Network or an MD-CDN for short.

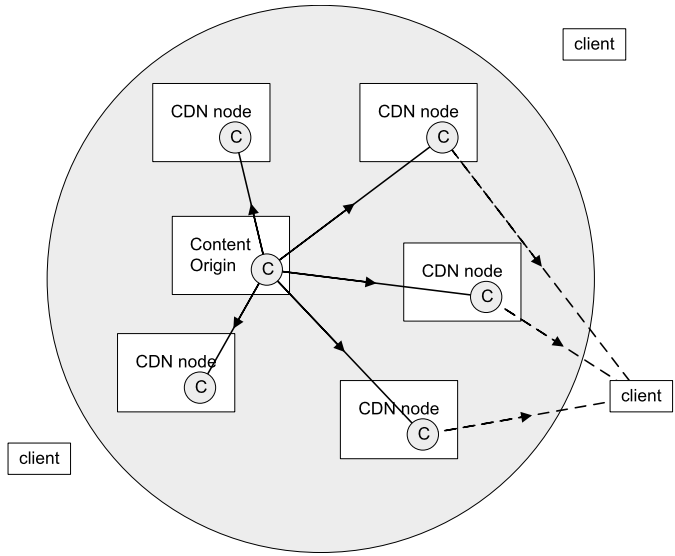


FIGURE 17.14: Content C is distributed to multiple CDN nodes. As a result, a client has the benefits associated with closer access, as well as the benefits of choosing one or more locations and paths to serve the content.

An MD-CDN operates in the following manner [3]. The media is coded into multiple complementary descriptions, which are distributed across different servers in the CDN. When a client requests a stream, complementary descriptions are simultaneously sent from different nearby servers through different network paths to the client. This architecture simultaneously reaps the benefits of path diversity, of source diversity, and of CDNs.

The multiple servers in a CDN can also provide path diversity with SD coding and FEC [25]. In this case, the content is replicated on multiple mirror servers and multiple servers stream disjoint sets of SD and FEC packets to the client. As discussed earlier, the reduced variability of losses afforded by path diversity makes FEC more efficient.

It is worth noting that the fundamental problems that arise when designing and operating a CDN are changed in important ways when using multipath streaming. Three key CDN problems are:

1. Where to deploy the servers? (Server placement problem)
2. How to distribute the content? (Content distribution across server problem)
3. How to select the best server for each client? (Server selection problem)

In a conventional CDN where each client receives a stream from a single server over a single path, the problems just given are solved by minimizing some notion of distance between a client and a server. In contrast, an MD-CDN should use a different metric that accounts for both distance and path diversity as discussed in Section 17.5.3. MD-CDN design and operation is discussed in [3].

17.6.3 Peer-to-Peer Networks

Peer-to-peer systems such as BitTorrent [11] are widely used for the distribution of large data files. The use of such systems for streaming is starting to appear in large-scale deployments (e.g., [47]) and is an active area of research.

Similar to a CDN, in a peer-to-peer distribution system every piece of content is stored by multiple peers. This allows scalability: the number of peer servers and peer clients increases at the same rate, overcoming the bottleneck of a single central server or a fixed number of CDN servers. Unlike the typical CDN, peers or locations may only have a portion of the content. This complicates the mechanism needed to locate content and generally necessitates communication with multiple peers to complete a transfer. Also, unlike a CDN, successful peer-to-peer systems usually enforce fairness constraints that enable peers who contribute more serving bandwidth to receive more in return.

Streaming using a peer-to-peer system, and in particular streaming of live events, is more difficult than distributing large data files. The essential challenge lies in providing an uninterrupted flow of data to each client for the duration of the streaming session. In comparison, for file distribution the data can flow in fits

and starts, and the users are broadly tolerant of fluctuations in total transfer time so long as visible progress occurs.

Various structures may be imposed on the relationships between peers to improve performance, for example, to reduce the end-to-end system delay or to increase robustness to peer failures. Peers may be taken as nodes in tightly organized distribution structures such as trees, sets of trees, or directed acyclic graphs (DAGs) (e.g., [6,28,37]) or in more flexible distribution nets (e.g., [47]). A comparison of various tree-based and DAG-based systems can be found in [6].

The critical need to avoid service outages in the face of uncertain peer and network conditions makes path diversity particularly relevant to peer-to-peer streaming. In many applications the peers are dispersed geographically and are unlikely to share bottlenecks (such as a DSL or cable modem uplink), hence an environment suitable for path diversity arises automatically. A system using path diversity with MD coding for peer-to-peer streaming is described in [28].

17.6.4 Path Diversity over Wireless Networks

Wireless networks are quite challenging for streaming since they generally offer time-varying and unpredictable behavior caused by multiple users, interference, propagation effects, and mobility. At the network level, these effects appear as variable delays, losses, and bandwidth. Wireless links, rather than the wired infrastructure, are typically the bottlenecks. As mentioned in the introduction, various forms of diversity have been used for decades to overcome these problems, for example, in the cellular environment.

Wireless LANs such as IEEE 802.11 are becoming widespread as they provide simple connectivity and data delivery. One approach to reliable streaming over 802.11 connections, in particular for low-latency or interactive communication, is to exploit the potential path diversity between each mobile client and multiple 802.11 access points (APs) in the infrastructure [23,24]. A mobile 802.11 wireless client is often in range of multiple APs, each offering a different relationship to the client with respect to distance, obstructions, multipath, signal strength, contention, available bandwidth, neighboring interferers, and potential hidden nodes. Figure 17.15 illustrates the variability seen by a mobile client in an 802.11 network with two access points.

If a router on the wired side of the network were clever enough to select the best access point for each packet at each time—or, more boldly, to schedule all the traffic flowing to all clients through the entire wireless system—significant performance gains would result. This is in fact the goal of some of the "switched WLAN" products currently on the market (e.g., those of Meru Networks and others). The scheduling in switched WLANs cannot, of course, be ideal because the current state of the wired and wireless network cannot be perfectly known even

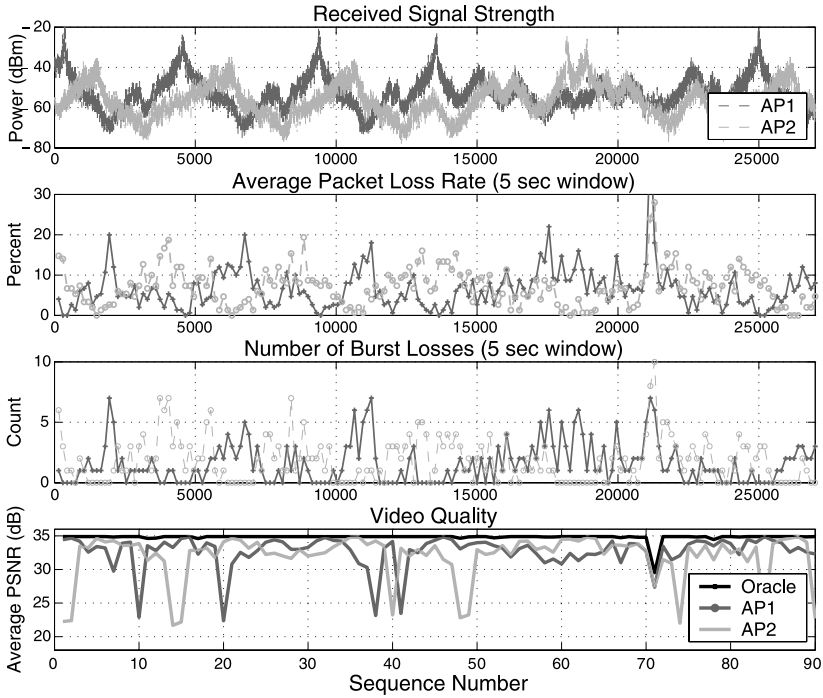


FIGURE 17.15: Performance statistics for path diversity between a mobile 802.11 wireless client and two APs in the distributed infrastructure [23]: received packet signal strength variation (top), average packet loss rate (second), and number of loss events of burst length ≥ 2 (third) as a function of packet sequence number for a 15-min packet trace. The bottom plot illustrates video quality when using only AP1 or only AP2, and the upper bound on achievable performance by selecting the best AP for every packet.

with a centralized controller. When some uncertainty about the best path at any given moment exists, path diversity techniques become useful.

By using multiple paths simultaneously or by switching between multiple paths (site selection) as a function of channel characteristics, results indicate that significant benefits may be achieved by using a wireless path diversity system compared to a conventional single access point system. In addition, these benefits may be achieved using a single client radio on a single channel [23]; there is no need to perform physical layer combining or multichannel operation or require multiradios.

Path diversity is also beneficial in ad-hoc wireless networks, where the topology may change or nodes may come or go. Many ad-hoc routing protocols identify multiple paths between the sender and the receiver, and the use of multiple paths can improve the reliability of the connection between the two wireless hosts. The effectiveness of path diversity over an ad-hoc wireless network for image and video communication is examined in [12] and [21].

17.6.5 Controlling Packet Routes

Path diversity requires, by definition, that packets destined for a receiver traverse different paths through the network. Even if the set of all possible paths was known (which would require learning the underlying network topology) and all the links in these paths were statistically characterized (another difficult measurement problem) and it was known how to both generate and apportion packets across paths to maximize some measure of performance (the subject of previous sections), there remains the problem of achieving the required degree of control over packet routing.

Indeed, the end-to-end structure of IP network is designed to achieve the opposite effect: packets are routed independently based on the destination address and reach their destination at the whim of a variety of midnetwork routers. It would seem that path diversity requires this structure to be circumvented, for example, using source routing. As shown earlier this turns out to be partially but not entirely true; there are a variety of scenarios where path diversity arises "for free" as a consequence of other architectural features of the system. For example, path diversity occurs naturally when the desired content is available in multiple physical locations. As discussed previously, this can be arranged in a distributed caching infrastructure such as in a content delivery network or in a peer-to-peer delivery system.

When the content is available at only one location and cannot be cached, for example, real-time speech or video, path diversity must be realized using network or infrastructure support. One approach for directing different streams over different paths is to send each stream to a different relay host placed at a strategic (or merely convenient) node in the network. The relays forward the streams to their final destination(s) [1]. In a large corporation, for example, the relay hosts can be installed at the corporation's various points of presence scattered throughout the country or the world. The relay infrastructure forms an application-specific overlay network on top of the conventional Internet, thereby providing a service of improved reliability while leveraging the infrastructure of the Internet.

Source routing, when available, can be an attractive option for achieving path diversity. In certain circumstances it is possible for the packet source to specify the set of nodes or "source route" for each packet to traverse. Path diversity can then be achieved by explicitly specifying different source routes for different subsets

of packets. While IP source routing is theoretically straightforward, there are a number of problems that limit its use in the current Internet, although it may be useful within a private network. It has been argued that IPv6 may allow source routing to be adopted more widely.

17.7 SUMMARY AND FURTHER READING

Media streaming with path diversity has gained significant interest in the last few years, as it provides valuable benefits for overcoming some of the challenges that afflict best-effort packet networks, including dynamic and unpredictable available bandwidth, delay, and loss rate. Studies have shown promise for both the single sender to single receiver case and the multiple sender to single receiver case as in a streaming media CDN, as well as for both wired and wireless networks. A media streaming system may use multiple paths at the same time or may perform path selection where it chooses the best path to use at any point in time. Multiple description coding combined with path diversity can enhance the benefits of each and, in certain circumstances, can lead to sizable improvements over media coded with single description coding and sent over a single path. Path diversity helps us take a step closer to feedback-free video streaming, which could simplify a range of applications from low-latency communication to multicast or broadcast streaming. The combination of media streaming and path diversity provides significant promise, and it is likely to see continued evolution and adoption in the future.

A variety of excellent sources exist for further reading. For a description of the early work on path diversity for data delivery see [4,5,22] and the overview paper [14]. The idea that multiple paths can improve fault tolerance and link recovery for data delivery, as well as provide larger aggregate bandwidth and load balancing, has long been known. The combination of using multiple paths and sophisticated rateless FEC codes can provide faster bulk data downloads, without the transmission of redundant data [9]. The use of path diversity for media streaming gained attention more recently. The paths may originate from a single source (e.g., [1,8, 12,19–21,23]) or from multiple sources (e.g., [3,10,25]). Dynamic path selection for streaming, from a set of possible paths, is also an important problem [36]. While path diversity can be exploited with many different types of media coding, the combination of multiple description coding and path diversity is conceptually particularly appealing. Excellent reviews of MD coding, both history and theory, and of MD video coding are available in [13] and [42], respectively.

REFERENCES

[1] J. G. Apostolopoulos. “Reliable Video Communication over Lossy Packet Networks Using Multiple State Encoding and Path Diversity,” *Visual Communications and Image Processing (VCIP)*, January 2001.

[2] J. G. Apostolopoulos, W. Tan, S. J. Wee, and G. W. Wornell. "Modeling Path Diversity for Multiple Description Video Communication," *IEEE ICASSP*, May 2002.

[3] J. G. Apostolopoulos, T. Wong, W. Tan, and S. J. Wee. "On Multiple Description Streaming with Content Delivery Networks," *IEEE INFOCOM*, June 2002.

[4] E. Ayanoglu, C.-L. I, R. D. Gitlin, and J. E. Mazo. "Diversity Coding: Using Error Control for Self-Healing in Communication Networks," *Proc. IEEE INFOCOM'90*, 1:95–104, June 1990.

[5] A. Banerjea. "Simulation Study of the Capacity Effects of Dispersity Routing for Fault-Tolerant Real-Time Channels," *Computer Communications Review (ACM SIGCOMM'96)*, 26(4):194–205, October 1996.

[6] M. Bansal and A. Zakhor. "Path Diversity Based Techniques for Resilient Overlay Multimedia Multicast," in *Picture Coding Symposium*, December 2004.

[7] S. Baset and H. Schulzrinne. "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," *Columbia University Technical Report CUCS-039-04*, September 15, 2004.

[8] A. Begen, Y. Altunbasak, and O. Ergun. "Fast Heuristics for Multi-Path Selection for Multiple Description Encoded Video Streaming," *IEEE ICME*, July 2003.

[9] J. Byers, M. Luby, and M. Mitzenmacher. "Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed up Downloads," *IEEE INFOCOM*, 1999.

[10] J. Chakareski and B. Girod. "Rate-Distortion Optimized Packet Scheduling and Routing for Media Streaming with Path Diversity," *IEEE DCC*, April 2003.

[11] B. Cohen. "Incentives Build Robustness in BitTorrent," in *1st Workshop on Economics of Peer-to-Peer Systems*, May 2003.

[12] N. Gogate, D. Chung, S. S. Panwar, and Y. Wang. "Supporting Image/Video Applications in a Mobile Multihop Radio Environment Using Route Diversity and Multiple Description Coding," *IEEE Trans. Circuits and System for Video Technology*, September 2002.

[13] V. Goyal. "Multiple Description Coding: Compression Meets the Network," *IEEE Signal Processing Magazine*, September 2001.

[14] E. Gustafsson and G. Karlsson. "A Literature Survey on Traffic Dispersion," *IEEE Network Magazine*, 1997.

[15] B. Heng, J. G. Apostolopoulos, and J. S. Lim. "End-to-End Rate-Distortion Optimized MD Mode Selection for Multiple Description Video Coding," *EURASIP Journal on Applied Signal Processing special issue on Video Analysis and Coding for Robust Transmission*, 2006.

[16] N. S. Jayant. "Subsampling of a DPCM Speech Channel to Provide Two 'Self-contained' Half-Rate Channels," *Bell Syst. Tech. J.*, 60(4):501–509, April 1981.

[17] N. S. Jayant and S. W. Christensen. "Effects of Packet Losses in Waveform Coded Speech and Improvements Due to an Odd-Even Sample-Interpolation Procedure," *IEEE Transactions on Communications*, COM-29(2):101–109, February 1981.

[18] J. N. Laneman, E. Martinian, G. W. Wornell, and J. G. Apostolopoulos. "Source-Channel Diversity for Parallel Channels," *IEEE Transactions on Information Theory*, October 2004.

[19] Y. Liang, E. Setton, and B. Girod. "Channel-Adaptive Video Streaming Using Packet Path Diversity and Rate-Distortion Optimized Reference Picture Selection," *IEEE Fifth Workshop on Multimedia Signal Processing*, December 2002.

[20] Y. J. Liang, E. G. Steinbach, and B. Girod. "Real-Time Voice Communication over the Internet Using Packet Path Diversity," *Proc. ACM Multimedia*, September/October 2001.

[21] S. Mao, S. Lin, S. Panwar, Y. Wang, and E. Celebi. "Video Transport over Ad Hoc Networks: Multistream Coding with Multipath Transport," *IEEE Journal on Selected Areas in Communications*, pages 1721–1737, December 2003.

[22] N. F. Maxemchuk. *Dispersity Routing in Store and Forward Networks*. Ph.D. thesis, University of Pennsylvania, May 1975.

[23] A. Miu, J. Apostolopoulos, W. Tan, and M. Trott. "Low-Latency Wireless Video Over 802.11 Networks Using Path Diversity," *IEEE ICME*, July 2003.

[24] A. Miu, G. Tan, H. Balakrishnan, and J. G. Apostolopoulos. "Divert: Fine-Grained Path Selection for Wireless LANs," *ACM MobiSys*, June 2004.

[25] T. Nguyen and A. Zakhor. "Distributed Video Streaming with Forward Error Correction," *Packet Video Workshop*, April 2002.

[26] T. Nguyen and A. Zakhor. "Path Diversity with Forward Error Correction (pdf) System for Packet Switched Networks," *IEEE INFOCOM*, April 2003.

[27] L. Ozarow. "On a Source Coding Problem with Two Channels and Three Receivers," *Bell Syst. Tech. J.*, 59:1909–1921, December 1980.

[28] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai. "Distributing Streaming Media Content Using Cooperative Networking," *ACM NOSSDAV*, May 2002.

[29] V. Paxson. "End-to-End Internet Packet Dynamics," *Proc. of the ACM SIGCOMM*, pages 139–152, September 1997.

[30] S. Pradhan, R. Puri, and K. Ramchandran. "n-Channel Symmetric Multiple Descriptions. Part I: (n, k) Source-Channel Erasure Codes," *IEEE Transactions on Information Theory*, pages 47–61, January 2004.

[31] R. Puri, S. Pradhan, and K. Ramchandran. "n-Channel Symmetric Multiple Descriptions. Part II: An Achievable Rate-Distortion Region," *IEEE Transactions on Information Theory*, April 2005.

[32] R. Puri and K. Ramchandran. "Multiple Description Source Coding Using Forward Error Correction Codes," *IEEE Asilomar Conference on Signals, Systems, and Computers*, October 1999.

[33] A. R. Reibman, H. Jafarkhani, Y. Wang, M. T. Orchard, and R. Puri. "Multiple Description Video Coding Using Motion-Compensated Temporal Prediction," *IEEE Trans. Circuits and Systems for Video Technology*, March 2002.

[34] D. Rubenstein, J. Kurose, and D. Towsley. "Detecting Shared Congestion of Flows Via End-to-End Measurement," *ACM SIGMETRICS*, 2000.

[35] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. "The End-to-End Effects of Internet Path Selection," *ACM SIGCOMM*, October 1999.

[36] S. Tao and R. Guerin. "Application-Specific Path Switching: A Case Study for Streaming Video," *ACM Multimedia*, October 2004.

[37] D. Tran, K. Hua, and T. Do. "ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming," in *IEEE INFOCOM*, April 2003.

[38] M. van der Schaar and D. S. Turaga. "Multiple Description Scalable Coding Using Wavelet-Based Motion Compensated Temporal Filtering," *IEEE ICIP*, September 2003.

[39] R. Venkataramani, G. Kramer, and V. Goyal. "Multiple Description Coding with Many Channels," *IEEE Transactions on Information Theory*, 49(9):2106–2114, September 2003.

[40] Y. Wang and S. Lin. "Error Resilient Video Coding Using Multiple Description Motion Compensation," *IEEE Trans. Circuits Systems for Video Tech*, June 2002.

[41] Y. Wang, M. T. Orchard, V. Vaishampayan, and A. R. Reibman. "Multiple Description Coding Using Pairwise Correlating Transforms," *IEEE Transactions on Image Processing*, pages 351–366, March 2001.

[42] Y. Wang, A. Reibman, and S. Lin. "Multiple Description Coding for Video Communications," *Proceedings of the IEEE*, January 2005.

[43] S. Wenger, G. Knorr, J. Ott, and F. Kossentini. "Error Resilience Support in H.263+," *IEEE Transactions on Circuits and Systems for Video Technology*, pages 867–877, November 1998.

[44] H. Witsenhausen. "On Source Networks with Minimal Breakdown Degradation," *Bell Syst. Tech. J.*, 59:1083–1087, July–August 1980.

[45] H. Witsenhausen and A. D. Wyner. "Source Coding for Multiple Descriptions II: A Binary Source," Technical Report TM-80-1217, Bell Labs, December 1980.

[46] J. Wolf, A. Wyner, and J. Ziv. "Source Coding for Multiple Descriptions," *Bell Syst. Tech. J.*, 59:1417–1426, October 1980.

[47] X. Zhang, J. Liu, and B. Li. "On Large-Scale Peer-to-Peer Live Video Distribution: CoolStreaming and Its Preliminary Experimental Results," in *IEEE Multimedia Signal Processing Workshop*, October 2005.

18

Distributed Video Coding and Its Applications

Abhik Majumdar, Rohit Puri, Kannan Ramchandran, and Jim Chou

18.1 INTRODUCTION

Contemporary digital video coding architectures have been driven primarily by the "downlink" broadcast model of a complex encoder and a multitude of light decoders. However, with the current proliferation of video devices ranging from hand-held digital cameras to low-power video sensor networks to camera-equipped cellphones, the days of typecasting digital video transmission as a downlink experience are over. We expect future systems to use multiple video input and output streams captured using a network of distributed devices and transmitted over a bandwidth-constrained, noisy wireless transmission medium, to either a peer (as in a peer-to-peer network) or a central location for processing. This new emerging class of "uplink"-rich media applications places a new set of architectural requirements that include:

- robustness to packet/frame loss caused by channel transmission errors;
- low-power and light-footprint encoding due to limited battery power and/or device memory; and
- high compression efficiency due to both bandwidth and transmission power limitations.

In addition, certain applications may impose very stringent end-to-end delay requirements. Current video coding paradigms fail to simultaneously address these requirements well. Predictive or full-motion inter-frame video coding approaches that are part of popular standards, such as H.26x and MPEGx [1,3,4,10] achieve

state-of-the-art compression efficiency, but fail to meet the other two criteria, as they are fragile to packet losses[1] while being computationally heavy at the encoder (primarily due to motion search). Alternatively, intra-frame video coding (motion-JPEG) methods, where individual frames are encoded as still images, are robust to packet drops and have low computational complexity but they take a high hit in compression efficiency.

One approach to overcoming these limitations and designing a new video coding solution that has *inbuilt robustness* to channel losses, a *flexible distribution of computational complexity between encoder and decoder*[2] depending upon the device constraints and the channel conditions, and *high compression efficiency* is to have a more *statistical* rather than a *deterministic* mindset. It is in this context that we introduce PRISM, a video coding paradigm founded on the principles of distributed source coding (also called source coding with side information) [38, 40]. Recently, there has been a spate of research activity in the area of video coding based on the ideas of distributed source coding (see Section 18.3.6). In this chapter, however, we will limit our discussion to the PRISM codec [32].

This chapter is organized as follows. Sections 18.2 and 18.3 provide background information motivating the PRISM framework. Section 18.4 lists the key architectural goals underlying the proposed PRISM framework. With a view to quantify the key architectural traits of PRISM, we present the information–theoretic performance limits of prediction-based and side information-based video codecs in Section 18.5. These theoretical insights guide the practical implementation of PRISM described in detail in Sections 18.6 and 18.7 along with experimental results presented in Section 18.8.

With the continued expansion of high-speed wireless networks, such as third-generation cellular networks, 802.11a/b/g (WiFi), and 802.16 (WiMAX), we expect the proliferation of "video sensor networks" in the near future. Typical video sensor networks would be made up of multiple cameras with varying degrees of spatially and temporally overlapping coverage, generating correlated signals that need to be processed, compressed, and exchanged in a loss-prone wireless environment in order to facilitate real-time decisions. Since there would be a high degree of spatiotemporal correlation in the data gathered by a video sensor network, distributed source coding principles can provide useful tools for efficiently exploiting this correlation. In Section 18.9 we discuss briefly how the PRISM architecture can be scaled to scenarios involving multicamera applications.

[1] The loss of predictor information in inter-frame coding renders the residue information useless from the point of view of decoding leading to fragility.

[2] In this work, by complexity we refer to motion search complexity.

18.2 CONVENTIONAL VIDEO CODING BACKGROUND

This section quickly overviews the conventional video inter-frame motion-compensated predictive coding (MCPC) architecture that underlies current video coding standards such as the MPEGx and H.26x. Video is a temporal sequence of two-dimensional images (also called frames). For the purpose of encoding, each of these frames is partitioned into regular spatial blocks. These blocks are encoded primarily in the following two modes.

1. **Intra-Coding (I) Mode:** The intra-coding mode exploits the spatial correlation in the frame that contains the current block by using a block transform such as the Discrete Cosine Transform. It typically achieves *poor compression*, since it does not exploit the temporal redundancies in video.
2. **Inter-Coding or Motion-Compensated Predictive (P) Mode:** This mode exploits both spatial and temporal correlations present in the video sequence, resulting in *high compression*. The *high-complexity* motion estimation operation uses the frame memory to infer the best predictor block for the block being encoded. Motion compensation provides the residue between the predictor block and the block in question, which is then transformed and encoded. Inter-coding is illustrated in Figure 18.1.

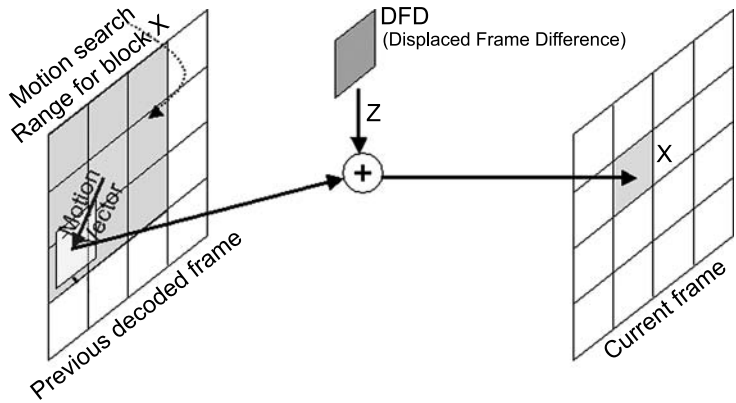


FIGURE 18.1: P-Frame coding (motion-compensated predictive video coding): The current frame is divided up into blocks of n pixels. $\mathbf{X}$ is the current block being encoded. $\mathbf{Y}_1, \ldots, \mathbf{Y}_M$ are M candidate predictor blocks for $\mathbf{X}$ in the previous decoded frame within a search range. $\mathbf{Y}_T$ is the best predictor for $\mathbf{X}$. $\mathbf{Z}$ corresponds to the prediction error (or innovations noise).

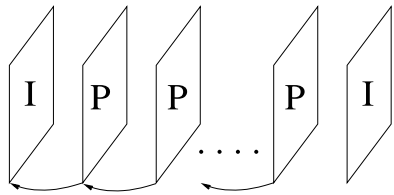


FIGURE 18.2: A Group of Frames. I, intra-mode coded frames; P, motion-compensated predictive mode coded frames.

Typically, the video sequence is grouped into a Group of Frames (see Figure 18.2) where the first frame in the group is coded in Intra-mode only while the remaining frames in the group are usually coded in Inter-mode.

Intra-coding has *low encoding complexity* and *high robustness* (being a self-contained description of the block being encoded) but has *poor compression efficiency*. To offset this, the MPEGx and H.26x standards use MCPC to achieve the compression needed to communicate over bandwidth-constrained networks. However, MCPC suffers from two major drawbacks.

(a) Fragility to synchronization or "drift"[3] between encoder and decoder in the face of prediction mismatch, primarily due to channel loss, is a major drawback of the current paradigms. This is a major problem in wireless communication environments, which are characterized by noise and deep fades.

(b) These frameworks are hampered by a rigid computational complexity partition between encoder (heavy) and decoder (light) where the encoding complexity is dominated by the motion search operation.

18.3 BACKGROUND ON SOURCE CODING WITH SIDE INFORMATION

We now introduce the concept of source coding with side information (distributed source coding) by examining the following illustrative example [31] (see Figure 18.3).

[3]Difference in frame memories at the encoder and the decoder results in the residue error being encoded at the encoder off some predictor and decoded at the decoder off some other predictor causing drift. Scenarios such as transmission losses can lead to nonidentical encoder and decoder frame memories.

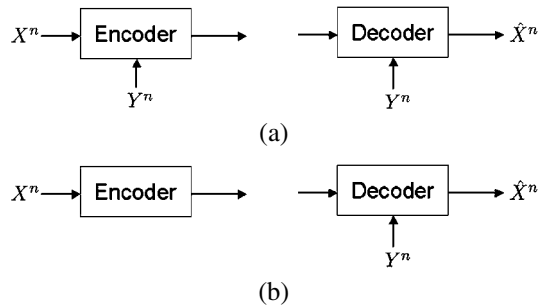


FIGURE 18.3: (a) Source coding with side information at both encoder and decoder. (b) Source coding with side information only at the decoder.

18.3.1 Example for Source Coding with Side Information

Let X and Y be length 3-bit binary data that can equally likely take each of eight possible values. However, they are correlated such that the Hamming distance between them is at most 1. That is, given Y (e.g., [0 1 0]), X is either the same as Y ([0 1 0]) or different in the first ([1 1 0]) or the middle ([0 0 0]) or the last bit ([0 1 1]). The goal is to efficiently encode X in the following two scenarios (see Figure 18.3) so that it can be perfectly reconstructed at the decoder.

Scenario 1: In the first scenario (see Figure 18.3a), Y is present both at the encoder and at the decoder. Here X can be predicted from Y. The residue $(X \oplus Y)$ or the error pattern of X with respect to Y takes four distinct values and hence can be encoded with 2 bits. The decoder can combine the residue with Y to obtain X. We note that X is analogous to the current video block that is being encoded, Y is analogous to the predictor from the frame memory, the correlation between X and Y is analogous to the temporal correlation, hence this method corresponds to **predictive coding** (Section 18.2).

Scenario 2: Here Y is made available to the decoder but the encoder for X does not have access to Y as illustrated in Figure 18.3b. However, it does know the correlation structure and also knows that the decoder has access to Y. This scenario being necessarily no better than the first scenario, its performance is limited by that of the first scenario. However, even in this seemingly worse case, we can achieve the same performance as in the first scenario (i.e., encode X using 2 bits)!

This can be done using the following approach. The space of codewords of X is partitioned into four sets each containing two codewords, namely **Coset1** ([0 0 0] and [1 1 1]), **Coset2** ([0 0 1] and [1 1 0]), **Coset3** ([0 1 0] and [1 0 1]), and **Coset4** ([1 0 0] and [0 1 1]). The encoder for X identifies the set containing the codeword for X and sends the index for the set (which can be described in 2 bits),

also called syndrome, instead of the individual codeword. The decoder, in turn, on the reception of the coset index (syndrome), uses Y to disambiguate the correct X from the set by declaring the codeword that is closest to Y as the answer. Note that the distance between X and Y is at most 1, and the distance between the two codewords in any set is 3. Hence, decoding can be done perfectly.

Such an encoding method, where the decoder has access to correlated side information, is known as **coding with side information** (also called distributed source coding). We note that **Coset1** in the aforementioned example is a repetition channel code [27] of distance 3 and the other sets are cosets [12,13] of this code in the codeword space of X. In channel coding terminology, each coset is associated with a unique *syndrome*. We have used a channel code that is "matched" to the correlation distance (equivalently, noise) between X and Y to partition the source codeword space of X (which is the set of all possible 3 bit words) into cosets of the 3-bit repetition channel code. The decoder here needs to perform *channel decoding* since it needs to guess the source codeword from among the list of possibilities enumerated in the coset indicated by the encoder. To do so, it finds the codeword in the indicated coset closest to Y. Since the encoder sends the label or *syndrome* for the coset containing the codeword for X to the decoder, we sometimes refer to this operation as **syndrome coding**.

The general source coding with side-information problem, where the source X and decoder side information Y are discrete random variables and X is to be communicated losslessly to the decoder (as in the aforementioned example), has been solved in literature [38] and is known as the Slepian–Wolf theorem. This result, which gives the smallest rate required for communicating X, is summarized in Section 18.3.2.

18.3.2 Source Coding with Side Information: Lossless Case

Consider the problems depicted in Figure 18.3. In Figure 18.3a, the side-information Y^n is available only to the decoder, while in Figure 18.3b it is available to both encoder and decoder. In both cases, let $\{X_i, Y_i\}_{i=1}^n$ be i.i.d. $\sim p(x, y)$, where X and Y are discrete random variables drawn from finite alphabets $\mathcal{X}$ and $\mathcal{Y}$, respectively. The decoder is interested in recovering X^n perfectly with high probability, that is,

$$P_e^{(n)} = P\left(\hat{X}^n \neq X^n\right) \to 0 \quad \text{as } n \to \infty.$$

Now, from information theory [11] we know that the rate region for the problem of Figure 18.3b, when the side information is available to both encoder and decoder, is $R \geq H(X|Y)$. The surprising result of Slepian and Wolf [38] is that the rate region for the problem of Figure 18.3a, when the side information is only

available to the decoder, is also $R \geq H(X|Y)$.[4] *Thus one can do as well when the side information is available only to the decoder as when it is available to both encoder and decoder.*

We now turn to the case when we are interested in recovering X^n at the decoder to within some distortion. This is the subject matter of the Wyner–Ziv theorem [40] presented later (see Section 18.3.3), which extends distributed source coding to the more general case of lossy coding with a distortion measure, which is the case of interest for video.

18.3.3 Source Coding with Side Information: Lossy Case

Consider again the problem of Figure 18.3a. We now remove the constraint on X and Y to be discrete and allow them to be continuous random variables as well. We are now interested in recovering X^n at the decoder to within a distortion constraint D for some distortion measure $d(x, \hat{x})$. Let $\{X_i, Y_i\}_{i=1}^n$ be i.i.d. $\sim p(x, y)$ and let the distortion measure be $d(x^n, \hat{x}^n) = \frac{1}{n}\sum_{i=1}^n d(x_i, \hat{x}_i)$. Then the Wyner–Ziv theorem [40] states that the rate–distortion function for this problem is

$$R(D) = \min_{p(u|x)\,p(\hat{x}|u,y)} I(X; U) - I(Y; U),$$

where

$$p(x, y, u) = p(u|x)\,p(x, y)$$

and the minimization is under the distortion constraint

$$\sum_{x,u,y,\hat{x}} p\big(\hat{x} \mid u, y\big)\,p(u|x)\,p(x, y)\,d\big(x, \hat{x}\big) \leq D.$$

Here U is the active source codeword and the term $I(Y; U)$[5] is the rate rebate due to the presence of side information at the decoder. For the case when X and Y are jointly Gaussian and the mean-squared error (MSE) is the distortion measure, it can be shown [40], using the Wyner–Ziv theorem, that the rate–distortion performance for coding X^n is the same whether or not the encoder has access to Y^n (i.e., the encoder cannot use the knowledge of Y^n for improving the rate–distortion performance in coding X^n). Later in [30] it was shown that this also holds true for the case of $\mathbf{X} = \mathbf{Y} + \mathbf{Z}$, where $\mathbf{Z}$ is independent and identically distributed Gaussian, and the distortion measure is the MSE. In general, however,

[4] The notation $H(A)$ stands for the Shannon entropy of random variable A [11].

[5] The notation $I(A; B)$ stands for the Shannon mutual information between two random variables A and B [11].

when compared with a predictive coding approach, the side information-based approach has a small loss in performance [42]. This loss is often termed *Wyner–Ziv rate loss*. Let us now consider the following illustrative example from [23] for the Wyner–Ziv problem.

18.3.4 Illustrative Example for Wyner–Ziv Coding

In this example, X is a real-valued number. The encoder will first quantize X to $\hat{X}$ with a scalar quantizer with step size δ (Figure 18.4). Clearly, the distance between X and $\hat{X}$ is bounded as $|X - \hat{X}| \leq \delta/2$. We can think of the quantizer as consisting of three interleaved quantizers (cosets), each of step size 3δ. In Figure 18.4 we have labeled the reconstruction levels of the three quantizers as 'A', 'B', and 'C', respectively. The encoder, after quantizing X, will note the label of $\hat{X}$ and send this label to the decoder, which requires $\log_2(3)$ bits on average.

The decoder has access to the label transmitted by the encoder and the side information Y. In this example, we assume that X and Y are correlated such that $|Y - X| < \delta$. Thus, we can bound the distance between $\hat{X}$ and Y as

$$\left|\hat{X} - Y\right| \leq \left|\hat{X} - X\right| + |X - Y| < \frac{\delta}{2} + \delta = \frac{3\delta}{2}.$$

Because $\hat{X}$ and Y are within a distance of $\frac{3\delta}{2}$ of each other and the reconstruction levels with the same label are separated by 3δ, the decoder can correctly find $\hat{X}$ by selecting the reconstruction level with the label sent by the encoder that is closest to Y. This can be seen in Figure 18.4, which shows one realization of X and Y.

In this example, the encoder has transmitted only $\log_2(3)$ bits per sample, and the decoder can correctly reconstruct $\hat{X}$, an estimate within $\delta/2$ of the source X. In the absence of Y at the decoder, the encoder would need to quantize X on an m-level quantizer of step-size δ. Thus, by exploiting the presence of Y at the

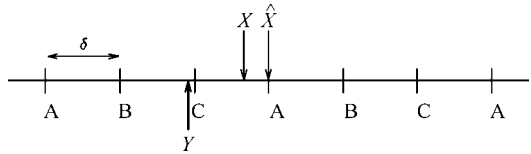


FIGURE 18.4: Distributed compression example: The encoder quantizes X to $\hat{X}$ and transmits the label of $\hat{X}$, an 'A'. The decoder finds the reconstruction level labeled 'A' that is closest to the side information, which is equal to $\hat{X}$.

decoder, the encoder saves $(\log_2(m) - \log_2(3))_+$ bits—this can be quite large if m is large, which should be the case if the variance of X is large.

Intuitively, what is happening here is that the source quantizer is partitioned into cosets of a channel code. We can think of the side-information Y as a free (but noisy) version of the source X available at the decoder. The decoder decodes this noisy version of X in a channel codebook (the specific codebook used will be the coset specified by the encoder). Just as in channel coding, the decoder needs to guess the source codeword from among a set of possible codewords. If the channel code is strong enough, Y will be decoded correctly to $\hat{X}$. Thus, *the goal is to partition a source codebook into good channel codes*. We see that the Wyner–Ziv problem requires a combination of source and channel coding. Note that this was also true for the lossless case (the Slepian–Wolf problem) as illustrated in the example of Section 18.3.1.

Let us now take a more formal look at Wyner–Ziv encoding and decoding.

18.3.5 Wyner–Ziv Encoding and Decoding

As in regular source coding, encoding proceeds by first designing a rate–distortion codebook of rate R' (containing $2^{nR'}$ codewords) constituting the space of quantized codewords for X. Each n-length block of source samples $\mathbf{X}$ is first quantized to the "nearest" codeword in the codebook. As in the illustrative example given earlier, the quantized codeword space (of size $2^{nR'}$ codewords) is further partitioned into 2^{nR} cosets or bins ($R < R'$) so that each bin contains $2^{n(R'-R)}$ codewords. This can be achieved by the information theoretic operation of random binning. The encoder only transmits the index of the bin in which the quantized codeword lies and thereby only needs R bits/sample.

The decoder receives the bin index and disambiguates the correct codeword in this bin by exploiting the correlation between the codeword and the matching n-length block of side-information samples $\mathbf{Y}$. This operation is akin to channel decoding. Once the decoder recovers the codeword, if MSE is the distortion measure, it forms the minimum MSE estimate of the source to achieve an MSE of D.

The optimal codec structure for this problem is illustrated in Figure 18.5 and can be briefly described as follows. The reader is referred to [11] for details.

- Codebook Construction: As in regular source coding, we first construct a codebook for quantization of source X to a random variable U. This is done by drawing $2^{nR'}$ n-length vectors, each of whose components is independent and identically distributed according to the marginal distribution $p_U(u)$, where $p_{U,X}(u, x) = p(x)p(u|x)$. For $\mathbf{X} = \mathbf{Y} + \mathbf{Z}$, where $\mathbf{Z}$ is independent and identically distributed Gaussian of variance σ_Z^2, and MSE distortion measure, we have $p(u|x) = \mathcal{N}(\alpha x, D\alpha)$, and $\alpha := \frac{1}{D}(\sigma_Z^2 - D)$.

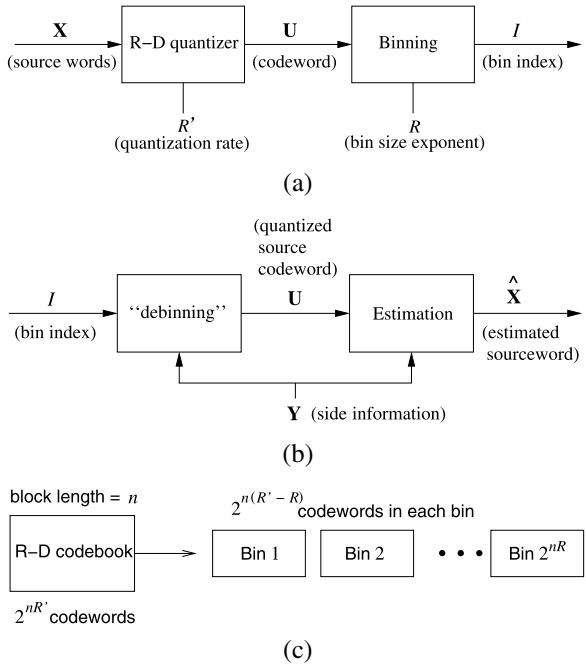


FIGURE 18.5: (a) Structure of distributed encoders. Encoding consists of quantization followed by a binning operation. (b) Structure of distributed decoders. Decoding consists of "de-binning" followed by estimation. (c) Structure of the codebook bins. The R–D codebook containing approximately $2^{nR'}$ codewords is partitioned into 2^{nR} bins each with approximately $2^{n(R'-R)}$ codewords.

This constitutes the space of quantized codewords for X. As in the illustrative example given earlier, this quantized codeword space (of size $2^{nR'}$ codewords) is further partitioned into 2^{nR} cosets or bins ($R < R'$) so that each bin contains $2^{n(R'-R)}$ codewords. This can be achieved by the information theoretic operation of random binning.

- Encoding: Each n-length block of source samples $\mathbf{X}$ is first quantized to the "nearest" codeword in the codebook. By standard rate–distortion theory arguments [11], this can be ensured by choosing a rate $R' = I(X;U)$. The encoder then transmits the index of the bin in which the quantized codeword lies and thereby only needs R bits/sample.
- Decoding: The decoder receives the bin index and attempts to disambiguate the correct codeword in this bin by exploiting the correlation between the

codeword and the matching n-length block of side-information samples $\mathbf{Y}$. This operation succeeds (with high probability) if the bin size is not too large to cause irrecoverable confusion: quantitatively, if $R' - R \leq I(U;Y)$. Hence, $R \geq I(X;U) - I(U;Y)$. For the choice of $p(u|x)$ given earlier, it can be shown that for $\mathbf{X} = \mathbf{Y} + \mathbf{Z}$, where $\mathbf{Z}$ is independent and identically distributed Gaussian, and the MSE distortion measure, this is the same rate distortion function as would be obtained if the side-information $\mathbf{Y}$ was available to both encoder and decoder. As mentioned earlier, *the decoding operation is akin to channel decoding since the decoder needs to guess the correct source codeword from all the possible codewords in the bin*.

- Reconstruction: Once the decoder recovers the codeword, it forms the best estimate of the source word using the recovered codeword and the side information. For the MSE distortion case, the decoder forms the minimum MSE estimate of the source-word $\hat{X}(j)$ to achieve a distortion D, where $\hat{X}(j) = f(U(j), Y(j)) := E[X(j) \mid U(j), Y(j)]$, $j = 1, \ldots, n$. Here $f(u, y) = u + (1 - \alpha)y$ and the MSE is $E(X - f(U, Y))^2 = D$.

Thus, we see that Wyner–Ziv encoding is an interplay of both source and channel coding, requiring us to design a good source codebook that can be partitioned into cosets of a good channel code.

18.3.6 Related Work

Before we describe the PRISM video compression framework, we briefly describe some of the related research activity. As mentioned before, PRISM is founded on the principles of distributed source coding, the information theory for which was established in the Slepian–Wolf [38] and the Wyner–Ziv [40] theorems for lossless and lossy cases, respectively. In fact, PRISM represents a generalization of the latter to the case where there is uncertainty in side information [20].

The first instance of distributed compression ideas for video coding can be found in [39], which pointed out the feasibility of distributed compression for video coding. However, [39] merely offers a conceptual treatment; there are no codec details and it does not address critical issues such as motion compensation. More recently, [32–34] proposed the PRISM distributed video codec system based on the framework of [20] in a block-level setup with motion search at the decoder. The distributed video coding problem was also independently studied in [6]. Further, [22,37] study the robustness property associated with distributed source coding and its application to video.

The idea of moving the computational burden away from the encoder was first presented in [35] in an MCPC setup, where the task of motion estimation was essentially transferred from the video encoder to the network terminal. In the distributed video coding context, the concept of moving motion estimation to the decoder was first presented in PRISM [32] and later in [17].

Further, there has been a spate of research activity in the area of distributed video coding addressing issues such as standards-compatible distributed video coding architectures [29,37] and scalability in the distributed video coding framework [36,41]. Finally, extensions of the PRISM framework from the point-to-point single camera case (as in the distributed video coding application) to the multicamera case (as in the video sensor network application) have been considered in [16,19,43].

18.4 ARCHITECTURAL GOALS OF PRISM

We now discuss the three major architectural goals of PRISM.

18.4.1 Compression Performance

As discussed before, it has been shown that in general, source coding with a side–information-based approach can have a small loss in compression performance when compared with a predictive coding approach [42]. However, it was shown in [30,38,40] that in many situations of interest, the performance of a side-information coding system can match that of one based on predictive coding, as in the example given in Section 18.3.1.

We note that **Coset1** in this example is a repetition channel code [27] of distance 3 and the other sets are cosets [12,13] of this code in the codeword space of X. We have thus used a channel code that is "matched" to the unit correlation distance (equivalently, noise) between X and Y to partition the source codeword space of X into cosets. This reduces the encoding rate for X and enables a side–information encoding system to give a **high compression** performance, comparable to a predictive coding system.

We now revisit the video coding problem. Let $\mathbf{X}$ denote the current macro-block to be encoded. Let $\mathbf{Y}$ denote the best (motion-compensated) predictor block for $\mathbf{X}$ and let $\mathbf{Y} = \mathbf{X} + \mathbf{N}$. Using the insight from the aforementioned example, we can encode X by finding a channel code that is matched to the correlation noise $\mathbf{N}$ (also called innovations process from $\mathbf{X}$ to $\mathbf{Y}$) and use that to partition the codeword space of $\mathbf{X}$.

18.4.2 Robustness

A major goal of PRISM is in-built robustness to packet and frame drops in contrast to what is possible with today's video codecs. PRISM targets this by using the "universally robust" side–information-based coding framework. The partitioning of X in the example of Section 18.3.4 is *universal*, that is, the same partitioning of X works for all Y regardless of the value of Y as long as both X and Y satisfy

the correlation structure. Note that in this example, as long as $|Y - X| < \delta$, the decoder is guaranteed to recover the correct $\hat{X}$.

Essentially, in the predictive coding framework the encoding for the current unit hinges on a *single* deterministic predictor, the loss of which results in erroneous decoding and error propagation. A side-information coding-based paradigm encodes the current unit, in principle, with respect to the correlation statistics between the current unit and the predictor only. At the decoder, the availability of *any* predictor that satisfies the correlation statistics enables correct decoding. A valid question to ask is: how does PRISM differ from the conventional use of Forward Error Correction (FEC) codes [27] on top of a MCPC compressed bit stream? Primarily, we note that FEC-based solutions serve to minimize the probability of error, thereby reducing the likelihood of mismatch between the encoder and the decoder *but they cannot fix the mismatch when there is one* (which is almost inevitable). Second, FEC-based solutions usually need large block lengths of data for attaining good performance, thus adding to the overall end-to-end delay while PRISM offers a low-latency solution.

18.4.3 Moving Motion-Search Complexity to the Decoder

Another architectural goal is to allow for a much more flexible distribution of the computational complexity (motion search) between encoder and decoder, depending upon device constraints, than is possible today. For example, in uplink-rich media applications, it is desirable to move the bulk of the complexity from the battery-power constrained encoder to the more capable decoder. PRISM facilitates this by allowing for partly or wholly *moving the computationally intensive motion search module to the decoder*. That is, in addition to the conventional high-complexity encoder and low-complexity decoder setting, PRISM also allows for the reverse possibility comprising a low-complexity encoder and a high-complexity decoder. The underlying theoretical paradigm for this is based on a generalization of the Wyner–Ziv side-information coding framework to the case where there is uncertainty at the receiver about the exact state of the side information [20], as will be described in Section 18.5.1.

In this context, we would like to point out that in conventional MCPC systems (where only the encoder performs motion search), a low-complexity encoding solution can be realized by limiting the amount of motion estimation performed at the encoder. This, however, decreases the overall compression efficiency of the system. In the case of PRISM, while in theory motion complexity can be completely moved to the decoder with little loss of compression performance (as is detailed in Section 18.5.1), in practice, however, we observe a similar complexity-compression trade-off in the PRISM video codec with a no-motion PRISM encoder generally taking a hit in compression performance relative to an inter-frame codec (see Section 18.8).

However, when we consider the end-to-end performance of PRISM for the case of transmission over a loss-prone channel, we can show that (see Section 18.5.3) as the channel noise increases, doing a motion search at the encoder gives diminishing marginal utility. At the same time, as the channel degrades, the decoder will need to search more among the list of available predictors to find one that enables successful decoding.

18.5 A THEORY FOR DISTRIBUTED VIDEO CODING

In Section 18.4, we proposed the distributed compression paradigm as a feasible approach for addressing the architectural goals of PRISM. In this section, we first describe a generalization of the concepts of lossy source coding with side information to the case when there is uncertainty in the side information at the decoder [20] (Section 18.5.1). This corresponds to the exact theoretical paradigm that underlies the PRISM framework. We then present an analysis that showcases the advantages of the side–information-based framework over the predictive coding framework when transmitting over a lossy channel (Section 18.5.2). We also discuss the complexity performance trade-offs for a side–information-based video codec when there are losses on the transmission channel (Section 18.5.3).

While the relatively simple models used in the following do not capture the rich and complex video phenomenon in its entirety, they are powerful enough to capture the essence of the problem at hand and offer valuable insights into developing the practical PRISM solution.

18.5.1 Sharing Motion Complexity Between Encoder and Decoder

In [20], it was shown that for an interesting class of signal models, motion complexity can be arbitrarily shared between encoder and decoder. Specifically, in [20] the information-theoretic rate–MSE performance of encoding $\mathbf{X}$ when the encoder *does not have access* to the decoded blocks in the previous frame(s), that is, motion compensation *is not possible* at the encoder, is compared with the performance when the encoder *has access* to all the correlated decoded blocks in the previous frame(s) to encode $\mathbf{X}$, as is done in contemporary video codecs that are based on MCPC. The surprising answer is that both systems have the same performance (when the innovations process has Gaussian statistics). In this section we give a short overview of the results of [20].

18.5.1.1 A Motion-Compensated Video Model

A model for video signals is depicted in Figures 18.1 and 18.6. Here, a block of pixels $\mathbf{X}$ in the current frame is modeled as the sum of a block of pixels $\mathbf{Y}_T$ in the previous decoded frame that is spatially close to $\mathbf{X}$ and a block of independent

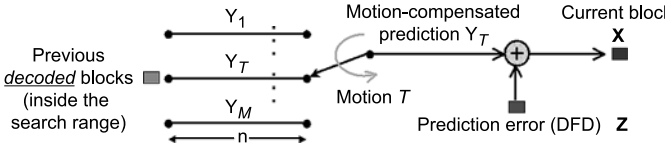


FIGURE 18.6: Motion-indexed additive–innovations model for video signals. $\mathbf{X}$ denotes a block of size n pixels in the current frame to be encoded and $\{\mathbf{Y}_i\}_{i=1}^{M}$ the set of blocks (each of size n) in the previous decoded frame corresponding to different values of the motion vector indexed by T.

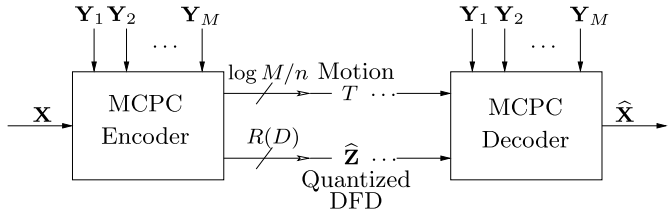


FIGURE 18.7: Motion-Compensated Predictive Coding (MCPC). Encoder sends motion using $\frac{1}{n}\log M$ bpp and the quantized residual $\mathbf{Z}$ using $R(D)$ bpp.

and identically distributed (i.i.d.) white Gaussian "innovations noise" $\mathbf{Z}$. That is, $\mathbf{X} = \mathbf{Y}_T + \mathbf{Z}$ where the parameter $T \in \{1, \ldots, M\}$ (called motion index) accounts for any motion that has occurred in consecutive frames.

18.5.1.2 Motion-Compensated Predictive Coding

We first derive the rate–MSE performance for the MCPC approach. Conventional MCPC is done in the following two steps (see Figure 18.7).

(a) The encoder estimates and transmits the index of the estimated motion vector to the decoder. The rate (bits per pixel or bpp) needed to specify T is given by $\frac{\log M}{n}$.

(b) Once the decoder knows T, the video coding problem is reduced to the problem of compressing the "source" $\mathbf{X}$ using the correlated side-information $\mathbf{Y}_T$ now available to *both* the encoder and the decoder. The solution to this problem is well known: for a target MSE value of D, the minimum rate $R(D)$ (in bpp) needed to encode $\mathbf{X}$ is given by the smallest rate that is needed to quantize $\mathbf{Z}$ to the nearest

codeword $\widehat{\mathbf{Z}}$ within a distortion D. That is, $R(D)$ is given by [11]

$$R(D) = \min\bigl(0, 0.5\log_2(\sigma_Z^2/D)\bigr). \tag{18.1}$$

The decoder receives the quantized codeword and reconstructs the source block as $\mathbf{Y}_T + \widehat{\mathbf{Z}}$ whose MSE is D. The total rate needed is $\frac{\log M}{n} + R(D)$ bpp.

18.5.1.3 Distributed Video Coding

In this case, due to severely limited processing capability (or some other reason), the encoder is disallowed from performing the complex motion-compensated prediction task. This is in effect pretending that the encoder does not have access to the previous decoded blocks $\mathbf{Y}_1, \ldots, \mathbf{Y}_M$.

Help from an Oracle:
As an intermediate step, consider the situation depicted in Figure 18.8 where an oracle reveals the true value of T only to the decoder (the encoder still does not know T). This is precisely the setup of the source coding with the decoder side information problem [40]. As mentioned in Section 18.3.3, for the special case of i.i.d. Gaussian statistics for $\mathbf{Z}$, the performance in the coding with side–information case is identical to MCPC, that is, the minimum bit rate needed to achieve an MSE D is given by (18.1). The encoding and decoding operations are as described in Section 18.3.3.
Reality: no oracle is available:
In the absence of the oracle, T is not available to the decoder and represents an additional source of uncertainty (see Figure 18.9). However, it turns out that this additional uncertainty can be overcome by decreasing the size of the bins or equivalently by having more bins. *This incurs an additional bit budget of* $\frac{1}{n}\log M$ *bpp, which is precisely the bit budget needed by motion-compensated predictive video codecs to convey the motion index to the decoder.* The encoder uses the same rate–distortion codebook as before. Whereas earlier each bin contained $2^{n(R'-R)}$

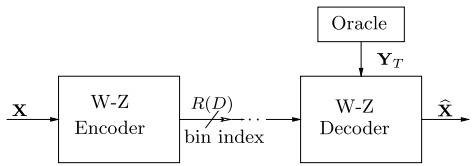


FIGURE 18.8: Wyner–Ziv codec with Oracle. Oracle reveals $\mathbf{Y}_T$ to the decoder only. Encoder does not have access to or is constrained from using $\mathbf{Y}_1, \ldots, \mathbf{Y}_M$.

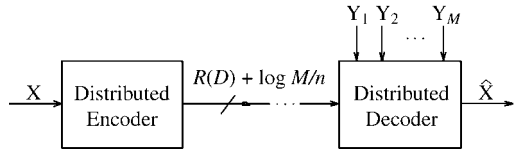


FIGURE 18.9: Low encoding complexity-distributed video codec. The encoder is incapable of using the previous decoded blocks. The decoder does not know the hidden motion index T.

codewords, now each bin will contain $2^{n(R'-R-\frac{1}{n}\log M)}$ codewords. Upon receiving a bin index, the decoder tries each $\mathbf{Y}_i$ in turn and stops as soon as it has found a codeword in the bin with which it is "sufficiently strongly correlated" according to the joint component statistics expected of $\mathbf{Y}_T$ and the quantized representation of $\mathbf{X}$.[6] This is like a "block-matching" motion-compensation operation but done at the decoder. It can be rigorously demonstrated that this algorithm not only finds the correct quantized codeword of $\mathbf{X}$, but also recovers the correct motion index T with high probability [20].

We have thus summarized an information-theoretic construction that enables shifting the motion complexity to the decoder without losing any performance relative to MCPC.

18.5.2 Robustness to Transmission Errors

In this section, we present an information-theoretic analysis for a very simple mismatched side-information problem that clarifies the nature of the drift problem associated with predictive coding and the value of distributed coding. This will highlight the superior robustness properties of the distributed video coding approach.

Consider the setup depicted in Figure 18.10. Here, $X = Y + Z$ stands for the data source that needs to be transmitted, Y denotes the predictor for X available at the encoder with associated independent innovations Z, and $Y' = Y + W$ is the predictor for X available at the destination. Here, W denotes the accumulated drift noise that cannot be observed at the encoder. We shall compare the performance of the predictive and distributed approaches for communicating X to the destination for two cases detailed later. In this analysis, we will consider that the encoder does a motion search and finds the best block to use from the previous frame(s) (which is the predictor Y) and sends the motion vector to the decoder. Hence, the decoder knows what side information to use, unlike in Section 18.5.1.3. At the end of this

[6] This is also referred to as "jointly typical decoding" [11].

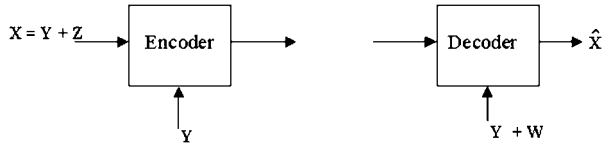


FIGURE 18.10: Problem setup: The encoder needs to compress the source $X = Y + Z$. Y is the predictor available to the encoder while $Y' = (Y + W)$ is the "drifted" predictor available to the decoder.

section, we will return to the model of Section 18.5.1.3, where the decoder needs to perform a motion search to find the correct side information to use.

18.5.2.1 Discrete Data, Lossless Recovery

The encoder does not have access to the realization of the drift random variable W, only to the joint statistics of Y, Z, and W. We assume that Z is independent of Y and Y'. The goal is to ensure that $\hat{X}$ is equal to X with high probability. It can be shown (see [28]) that the optimal solution is to ignore Y at the encoder and hence, using the Slepian–Wolf theorem [38], the smallest encoding rate in bits per sample needed to convey X losslessly to the decoder is given by

$$R_{dsc} = H\big(X|Y'\big).$$

Here the subscript "dsc" stands for distributed source coding.

We now outline the derivation of a lower bound on the rate required by a predictive coding system. The predictive coding system first forms the innovations Z and sends it to the decoder, incurring a total rate of $H(Z)$ bits per sample. If there is no drift between the encoder and the decoder ($W = 0$), the decoder can recover Z and use Y to recover X. However, with a nonzero drift between the encoder and the decoder, an additional drift correction rate needs to be incurred. But the encoder does not have access to the realization of the drift random variable W. To correct for the drift, the best that the encoder can do is to use a distributed source coding approach to convey the missing information needed to recover X. This incurs an extra rate of $H(X|Z, Y')$ bits per sample. Note, however, that this method of using distributed source coding to correct for the drift is *not* followed by the MPEGx or H.26x standards and so we can term this extra rate to correct for drift as a *lower bound* for the extra rate required by the predictive coding system to correct the drift. Hence, the predictive coding approach needs a total bit rate not smaller than

$$R_{pc}^{lb} = H(Z) + H\big(X \mid Z, Y'\big) = H(Z) + H\big(Y + Z \mid Z, Y'\big) = H(Z) + H\big(Y|Y'\big),$$

where the subscripts "pc" and "lb" stand, respectively, for predictive coding and lower bound. The last equality follows from the fact that Z is independent of Y. The last expression suggests an alternative interpretation for R_{pc}. The term $H(Y|Y')$ can be interpreted as the rate required to "correct" the side information, that is, resynchronize the encoder and the decoder frame memories.

Note that $H(Z|Y') = H(Z)$ since Z is independent of Y'. So, we have

$$\begin{aligned} R_{pc}^{lb} &= H(Z) + H\big(X \mid Z, Y'\big) \\ &= H\big(Z|Y'\big) + H\big(X \mid Z, Y'\big) \\ &= H\big(Z, X \mid Y'\big) \\ &= H\big(X|Y'\big) + H\big(Z \mid X, Y'\big) \\ &= R_{dsc} + H\big(Z \mid X, Y'\big). \end{aligned}$$

The rate penalty due to drift for the predictive coding approach over the distributed source coding approach is then

$$R_{pc}^{lb} - R_{dsc} = H\big(Z \mid X, Y'\big) = H\big((X - Y) \mid X, Y'\big) = H\big(Y \mid X, Y'\big),$$

which is always nonnegative and zero only when $Y = Y'$ (i.e., $W = 0$) or if $Z = 0$.

The rate penalty $R_{pc}^{lb} - R_{dsc}$ evaluated earlier is significant only if the "innovations" component $H(Z)$ and the "channel noise" component $H(Y|Y')$ are both large [since $H(Z \mid X, Y') \leq H(Z)$ and $H(Y \mid X, Y') \leq H(Y|Y')$]. This is not surprising since when $H(Z)$ is small the two-step approach is not "wasting" a lot of rate by sending $H(Z)$ simply because it does not require a lot of bits to send $H(Z)$. However, when $H(Y|Y')$ is small, the drift is small and it does not take too many bits to correct it (in the extreme case when there is no drift, predictive coding is, of course, optimal). However, the interesting case is really when the drift *is* significant and it is in this case that the aforementioned rate penalty is also significant. Hence, when there is significant drift, the predictive coding framework can be quite suboptimal.

18.5.2.2 Jointly Gaussian Data, Recovery with MSE $\leq D$

We now present a rate–distortion analysis for the two approaches in a jointly Gaussian setting. Random variables Y, Z, W are jointly Gaussian and mutually independent Gaussian random variables with variances $\sigma_y^2, \sigma_z^2, \sigma_w^2$, respectively. Let U denote the quantization random variable (the output of the encoder) and $\hat{X}$ the reconstruction random variable. We are interested in recovering X to a target distortion D.

It can be shown [28] that the optimal solution here is to ignore Y at the encoder and hence, using the Wyner–Ziv theorem [40], the smallest encoding rate in bits per sample needed is

$$R_{dsc}(D) = \min_{p(u|x)p(\hat{x}|u,y')} I\big(X; U \mid Y'\big) = \min_{p(u|x)p(\hat{x}|u,y')} I(X;U) - I(Y;U),$$

where $Y' = Y + W$ and the minimization is under the constraint that the overall expected distortion is at most D where MSE is the distortion measure. Then we have the following theorem:

Theorem 18.5-1. *For jointly Gaussian and mutually independent random variables, Y, Z, W with variances $\sigma_y^2, \sigma_z^2, \sigma_w^2$, respectively, and an MSE distortion measure, let $X = Y + Z$ be the source to be encoded with Y available at the encoder and $Y' = Y + W$ available at the decoder. Then the rate–distortion function for the distributed source coding approach is*

$$R_{dsc}(D) = \max\left(0, \frac{1}{2}\log_2 \frac{\sigma_z^2 + \sigma_y^2\sigma_w^2/(\sigma_y^2 + \sigma_w^2)}{D}\right). \tag{18.2}$$

For the proof of Theorem 18.5-1 please refer to [28].

As in Section 18.5.2.1, we lower bound the rate required by the predictive coding system using a two-step approach. The predictive coding system quantizes Z to $\hat{Z}$ with a distortion D. We assume $D < \sigma_z^2$ since otherwise the predictive coding system will not encode the innovations at all. If the encoder and the decoder use identical predictor information, $\hat{X}$ can be recovered from $\hat{Z}$ as $\hat{X} = \hat{Z} + Y$. In the general case, however, there is a drift between the encoder and the decoder. As in Section 18.5.2.1, the encoder needs to spend an additional rate (using distributed coding techniques) to correct for the drift, thus resulting in a total rate

$$R_{pc}^{lb}(D) = I\big(Z; \hat{Z}\big) + \min_{p(u|x)p(\hat{x}|u,y',\hat{z})} I\big(X; U \mid Y', \hat{Z}\big), \tag{18.3}$$

where $Y' = Y + W$ and the minimization is under the constraint that the overall expected distortion is at most D where MSE is the distortion measure. The rate required to correct the drift (using Wyner–Ziv techniques) is given by $\min_{p(u|x)p(\hat{x}|u,y',\hat{z})} I(X; U|Y', \hat{Z})$. Then we have the following theorem.

Theorem 18.5-2. *For jointly Gaussian and mutually independent random variables, Y, Z, W with variances $\sigma_y^2, \sigma_z^2, \sigma_w^2$, respectively, and an MSE distortion measure, let $X = Y + Z$ be the source to be encoded with Y available at the encoder and $Y' = Y + W$ available at the decoder. Then for target distortion*

$D < \sigma_z^2$, the rate distortion function for the two-step predictive coding and drift correction method is

$$R_{pc}^{lb}(D) = \frac{1}{2}\log_2 \frac{\sigma_z^2(D + \sigma_y^2\sigma_w^2/(\sigma_y^2 + \sigma_w^2))}{D^2}. \tag{18.4}$$

For the proof of Theorem 18.5-2 please refer to [28].

Note that when $D < \sigma_z^2$,

$$R_{dsc}(D) = \frac{1}{2}\log_2 \frac{\sigma_z^2 + \sigma_y^2\sigma_w^2/(\sigma_y^2 + \sigma_w^2)}{D}.$$

The rate penalty due to drift for the predictive coding approach over the distributed source coding approach is then

$$R_{pc}^{lb} - R_{dsc} = \frac{1}{2}\log_2 \frac{1 + A/D}{1 + A/\sigma_z^2},$$

where $A = \sigma_y^2\sigma_w^2/(\sigma_y^2 + \sigma_w^2)$. For the range of interest $D < \sigma_z^2$, the difference is positive. In fact, $R_{pc}^{lb} \geq R_{dsc}$, with $R_{pc}^{lb} = R_{dsc}$ if $D \geq \sigma_z^2$, or $\sigma_y^2 = 0$, or $\sigma_w^2 = 0$. In the context of video coding, $D \geq \sigma_z^2$ is akin to the case of not sending the block at all (the "skip" mode), $\sigma_y^2 = 0$ is like not having any useful predictor available (the "intra" mode), and $\sigma_w^2 = 0$ implies that the encoder and decoder are in sync (no drift). For all other cases, $R_{pc}^{lb} > R_{dsc}$.

Further, we note in the high-quality regime (i.e., $D \to 0$), we have

$$\lim_{D\to 0} \frac{R_{pc} - R_{dsc}}{R_{dsc}} \to 1, \tag{18.5}$$

that is, *the predictive coding system needs nearly double the rate as compared to the distributed coding system.*[7]

The analysis of this section is readily extended to the multiple predictors case of Section 18.5.1 where the side information at the decoder is not known exactly and is only known to be one among a set of predictors $\{\mathbf{Y}_i\}_{i=1}^n$.

To see this, note that the rate–distortion functions for the case when the side information is known at the decoder (the motion vector is known) and the case when it is not (the motion vector is unknown) are identical (as $n \to \infty$) [20]. That is, the problems depicted in Figures 18.9 and 18.8 have the same rate–distortion function. Thus it is sufficient to consider the case when the side information is known at the decoder, which is the case considered here.

[7] Equation (18.5) can be proved using L'Hospital's rule.

18.5.3 Complexity Performance Trade-Offs

In the previous sections of this chapter, we had assumed perfect knowledge of the correlation statistics at both the encoder and the decoder. However, real-world video encoding algorithms involve an "online" learning of the correlation statistics through the process of motion estimation. Typically, the more the complexity invested in the motion estimation process, the more accurate is the estimate of the statistics leading to better compression performance. While this is true for the case of transmission over a clean channel, in the following we show that, using a distributed video coding approach over a lossy channel, the *marginal value* of accurately learning the correlation statistics at the encoder diminishes as the channel noise increases.

Let X be the block to be encoded. Let $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_M)$ be the set of predictors available to the encoder for the block X. Let $\mathbf{Y}' = (Y'_1, Y'_2, \ldots, Y'_M)$ be the set of predictors available to the decoder for the same block. Here $\mathbf{Y}'$ is a noisy version of $\mathbf{Y}$ (due to previous transmission errors). We assume that $X \leftrightarrow \mathbf{Y} \leftrightarrow \mathbf{Y}'$ form a Markov chain, that is, the set of predictors $\mathbf{Y}'$ is a degraded version of the set of predictors $\mathbf{Y}$. As in Section 18.5.2.1, we are interested in communicating X losslessly to the decoder.

Note that the minimum rate required to losslessly communicate X to the decoder is $R_{dsc} = H(X|\mathbf{Y}')$, since $\mathbf{Y}'$ is the side information available to the decoder. It can be shown [21] that this rate is

$$R_{dsc} = H\big(X|\mathbf{Y}'\big) = H\big(\mathbf{Y}|\mathbf{Y}'\big) + H(X|\mathbf{Y}) - H\big(\mathbf{Y} \mid X, \mathbf{Y}'\big).$$

Since we are interested in observing the effect of channel noise, we will upper bound R_{dsc} by neglecting the last term (since $H(\mathbf{Y} \mid X, \mathbf{Y}')$ can at most increase to $H(\mathbf{Y}|X)$ as the noise increases). So

$$R_{dsc} \leq H\big(\mathbf{Y}|\mathbf{Y}'\big) + H(X|\mathbf{Y}). \tag{18.6}$$

Note that the term $H(X|\mathbf{Y})$ is a measure of the source correlation while $H(\mathbf{Y}|\mathbf{Y}')$ is a measure of the effect of channel noise. Further, also note that

$$H(X|Y_1) \geq H(X \mid Y_1, Y_2) \geq \cdots \geq H(X|\mathbf{Y}).$$

$H(X|Y_1)$ can be thought of as the encoder estimate of $H(X|\mathbf{Y})$ if the encoder only looked at one predictor from the previous frame. Similarly, $H(X \mid Y_1, Y_2)$ would be the encoder estimate of $H(X|\mathbf{Y})$ if it looked at two predictors from the previous frame, and so on. Thus, we will get better estimates of $H(X|\mathbf{Y})$ as we search the list of predictors (motion search). However, as (18.6) shows, the rate R_{dsc} depends on the sum of the correlation noise and the channel noise. For a

fixed correlation noise, the reduction in R_{dsc}, as we find the correlation noise more accurately (through more motion search), diminishes as we increase the channel noise.

18.6 PRISM: ENCODING

We have thus summarized a system-level information theory for distributed video coding with the goal of addressing the feasibility of the architectural goals of PRISM. In extending these methods to the real-world video scenario, we recognize that we are dealing with sources having complex correlation noise structures that are imprecisely known, requiring estimation models. In this context, we would like to point out a principal reason underlying the success of current video coding standards (H.26x and MPEGx), namely their ability to model, estimate, and process motion as a *local block-level phenomenon*. Realizing that accurate motion modeling is the key to success, we have based our implementation of the PRISM on the local block-motion, block-DCT, block-coding framework.

We now list the main aspects of the PRISM encoding process. We note that the video frame to be encoded is divided into nonoverlapping spatial blocks (we choose blocks of size 8×8 in our implementation).

18.6.1 Decorrelating Transform

We first apply a DCT on the source block. The transformed coefficients $\mathbf{X}$ are then arranged in a one-dimensional order (size 64) by doing a zig-zag scan on the two-dimensional block (size 8×8).

18.6.2 Quantization

Following this, we apply a scalar quantizer. The transformed coefficients are quantized with a target quantization step size chosen based on the desired reconstruction quality (as in [10]).

18.6.3 Classification

The next step involves the design of a Wyner–Ziv codebook in order to exploit the correlation between the source and the side information. In this context, it is convenient to view individual quantized coefficients in a block in terms of bit planes, as shown in Figure 18.11. Correlation between a source coefficient X_i ($i \in \{0, 1, \ldots, 63\}$) and the corresponding side-information Y_i can be interpreted in terms of the number of most-significant bit planes of the quantized representation of X_i that can be inferred from side-information Y_i (Figure 18.11 illustrates this with the bits corresponding to the white color being predictable using the

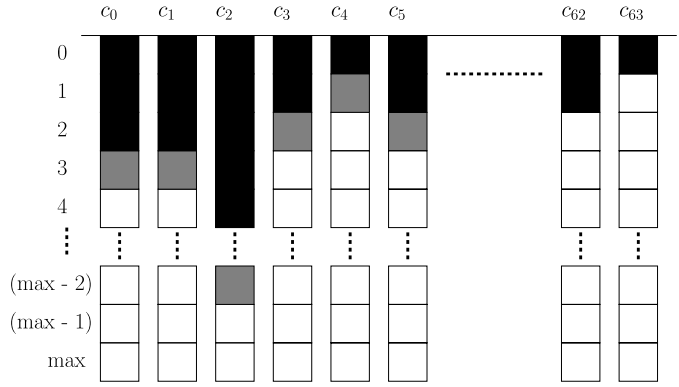


FIGURE 18.11: A bit plane view of a block of 64 coefficients. Bit planes are arranged in increasing order with 0 corresponding to the least-significant bit.

decoder side information). The remaining least-significant bit planes (shown in gray and black in Figure 18.11) are not inferable at the decoder and need to be encoded. These bits constitute the Wyner–Ziv encoding (syndrome) for source coefficient X_i.

This is also illustrated in Figure 18.12. Here U_i corresponds to the quantized representation for X_i with a target step-size δ. Starting from the least-significant bit plane of U_i, each successive bit plane identifies a partition of codewords containing U_i. The number of least-significant bits that need to be communicated to the decoder is given by the tree depth for which the distance between successive codewords in the partition is greater than twice the correlation noise magnitude between U_i and Y_i. This would enable correct decoding (in other words, inferring the remaining most-significant bit planes) of U_i at the decoder using Y_i. Clearly, the higher the correlation, the smaller the correlation noise, the greater the number of most-significant bit planes that can be predicted from the side information. The Wyner–Ziv encoding of a source block $\mathbf{X}$ thus corresponds to a suitable number of least-significant bit planes for each coefficient X_i such that the remaining (most-significant) bit planes can be inferred at the decoder.

While the previous paragraph describes the method for Wyner–Ziv encoding of source block $\mathbf{X}$, one important difference between the aforementioned description and the problem at hand is that the aforementioned description assumes knowledge of the correlation structure between $\mathbf{X}$ and $\mathbf{Y}$ at the encoder. In practice, however, this structure is not known precisely and needs to be estimated for better compression performance. It is for this reason that we use a classification module in our approach, with the goal of estimating the correlation noise between each

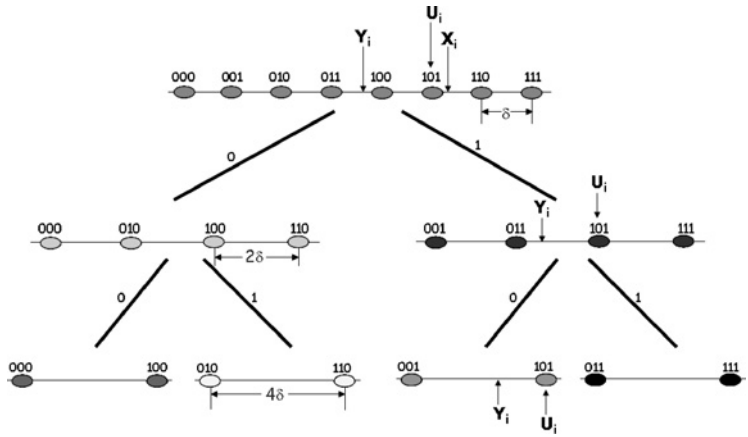


FIGURE 18.12: Partitioning of the quantization lattice into levels. X_i is the source, U_i is the (quantized) codeword, and Y_i is the side information. The number of levels in the partition tree depends on the effective noise between U_i and Y_i.

video block and its temporal predictor from frame memory, as measured in number of most-significant bits that are predictable at the decoder for every quantized coefficient.

Real-world video sources exhibit a spatiotemporal correlation noise structure with highly varying statistics. Spatial blocks that are a part of the scene background or relate to previous frames via simple/regular motion exhibit high correlation with their temporal predictor blocks (i.e., they are associated with correlation noise **N** that is "small"). However, blocks that are a part of a scene change, occlusion, or arise from irregular motion have little correlation with their temporal predictor blocks ("large" **N**). Our classification module deploys block-motion estimation to infer the correlation noise between the current 8×8 spatial block and its temporal predictors. As discussed in Section 18.5.3, depending on the available complexity budget, as well as the prevailing channel conditions, the classification module can perform varying degrees of motion search, ranging from an exhaustive motion search to a coarse motion search to no motion search at all. We now discuss our implementation of the classification module for two extreme configurations—one corresponding to little motion search and the other corresponding to an exhaustive motion search. The compression/robustness performance of these schemes is presented later in Section 18.8.

(a) **No motion search:** In this case, we use the residue information between the current block (considered in the pixel domain) and the colocated block in the pre-

vious frame (corresponding to zero motion) to infer the correlation noise $\mathbf{N}$ and, consequently, the number of least-significant bits that need to be communicated to the decoder. We use a combination of online measurements and off-line training to accomplish this objective.

- We first determine the scalar mean-squared error E corresponding to the residue information between the current block and its zero-motion predictor.
- The appropriate "class" for the current block is then determined by thresholding E using a set of predetermined threshold values. We use a set of 15 thresholds (T_i where $i \in \{0, \ldots, 14\}$) in our implementation corresponding to 16 classes labeled 0 through 15. Class i is chosen when $T_{i-1} \leq E < T_i$.
- Each class i is associated with a block correlation noise $\mathbf{N}^i$ whose statistics were determined using off-line training.[8] The inferred block correlation noise is used to determine the number of least-significant bits for individual coefficients that need to be communicated to the decoder.
- Note that by following this procedure, we have made a conservative determination of the correlation noise between the current block and its best motion-compensated temporal predictor block. As a result of this, due to the need to communicate more bit planes to the decoder, we can incur excessive bit rate. We treat this inefficiency by jointly encoding the most-significant bit planes of individual coefficients that need to be communicated to the decoder with a coset channel code. These bit planes correspond to the gray color in Figure 18.11. The remaining least-significant bit planes, shown in black in Figure 18.11, are encoded using a suitable entropy code. This is further detailed in Section 18.6.4.

(b) **High-complexity motion search:** Similar to conventional video encoders, in this case, we make a precise determination of the correlation noise between the current block and its best motion-compensated temporal predictor. The residue information between the current block and its motion-compensated temporal predictor is used to obtain the number of least-significant bits for individual coefficients that need to be communicated to the decoder. In terms of Figure 18.11, these bit planes correspond to black color (there is no gray color in this case) and are encoded using a suitable entropy code. Optionally, we can also indicate the chosen motion vector at the encoder as a part of the bitstream.

[8] We consider $\mathbf{N}^i$ in the transform domain where the absolute values of its components were modeled as a set of independent Laplacian random variables $\{N_0^i, N_1^i, N_2^i, \ldots, N_{63}^i\}$. The parameters corresponding to the various Laplacian random variables (a Laplacian random variable is completely characterized by its mean value) belonging to different classes were obtained by off-line training. We used a long news clip from the "Euronews" TV channel to derive these statistics. The choice of the Laplacian model was based on its success as reported in the literature [6] and by our experiments on statistical modeling of residue coefficients in the transform domain.

18.6.4 Syndrome Encoding

The syndrome encoding step is assigned the task of encoding the least-significant bits of individual coefficients in a block (as determined by the classification step) in an efficient manner. As described in Section 18.6.3, the least-significant bit planes for individual coefficients in a block fall under two cases—bit planes corresponding to black in Figure 18.11 are encoded using an entropy code and bit planes corresponding to gray are encoded using a coset channel code.

(a) **Entropy Coding:** We note that contemporary video compression standards including [1,3,10] use a (**run**, **level**, **sign**, **last**) 4-tuple[9]-based alphabet for entropy coding the residue information. Given the alphabet, the actual entropy coding is then accomplished by using Huffman coding [11] or arithmetic coding [25].

In our implementation, we use a variant of this method for defining the alphabet used for entropy coding of the least-significant bit planes shown in black in Figure 18.11. Our entropy coding alphabet consists of (**run**, **depth**, **path**, **last**) 4-tuple. Here **run** indicates the number of coefficients prior to the current coefficient for which no bit planes are encoded, **depth** indicates the number of least-significant bit planes encoded for the current coefficient, and **path** indicates the bit path in the binary tree that specifies the coset containing the current coefficient. The number of values taken by **path** is given by $2^{\mathbf{depth}}$. The entry **last** has identical meaning to the corresponding term used in contemporary video compression standards. We use an arithmetic coding engine that operates on this alphabet to efficiently code the syndrome information.

(b) **Coset Channel Coding:** As mentioned in Section 18.6.3, for the case of an encoder with no motion search, we make a conservative determination of correlation noise between the current block and its best motion-compensated temporal predictor block. As a result of this, due to the need to communicate more bit planes to the decoder, we can incur excessive bit rate. We treat this inefficiency by jointly encoding the topmost least-significant bit planes (corresponding to gray in Figure 18.11) using a coset channel code. These bits are encoded using the parity check matrix [27] of an (n, k_i) linear error correction code. $\mathbf{s}_i = H_i \mathbf{b}_i$, where $\mathbf{s}_i$ and H_i represent the syndrome and parity check matrix corresponding to the ith linear channel code and $\mathbf{b}_i$ represents the corresponding input bits. The encoding rate for this case is given by $(n - k_i)/n$ bits per coefficient.

[9] A block of quantized residue coefficients is interpreted as a set of (**run**, **level**, **sign**, **last**) tuples where **run** indicates the number of coefficients with a value equal to zero before a nonzero coefficient residue, **level** indicates the absolute value of the nonzero coefficient residue, **sign** indicates the sign associated with the nonzero coefficient residue, and the binary-valued **last** indicates whether the nonzero coefficient is the last in the block.

Since we have small block lengths at our disposal (640 samples for an 8×8 block), we use the relatively simple BCH [27] block codes, which work well even at reasonably small block lengths (unlike more sophisticated channel codes, such as LDPC codes [15] and turbo codes [7]). The parameter k_i associated with the ith channel code is a function of the class i (see Section 18.6.3) to which the block belongs. The parameters for each class were chosen by using the error probability versus SNR performance curves of the various BCH codes.

18.6.5 Hash Generation

When the encoder is of low complexity and/or the channel is lossy, we require the decoder to perform motion search. For this case, unlike the classical Wyner–Ziv coding setup, we have several side-information candidates Y_i at the decoder corresponding to various motion-predictor choices (as described in Section 18.5.1). The decoder does not know the "best" predictor for the block $\mathbf{X}$. In theory (see Section 18.5.1), it is possible to find the best predictor and decode the block $\mathbf{X}$ through joint-typical decoding. In practice, however, joint-typical decoding is not feasible given the short block lengths and the complexity constraints at the decoder. Accordingly, the encoder needs to transmit not only the syndrome for the side–information-encoded coefficients, but also a hash signature (of sufficient strength) for the quantized sequence codewords.

For this purpose, we use a cyclic redundancy check (CRC) checksum as a "signature" of the quantized codeword sequence. In contrast to the conventional paradigm, it is the decoder's task to do motion search here, and it searches over the space of candidate predictors one by one to decode a sequence from the set labeled by the syndrome. When the decoded sequence matches the CRC check, decoding is declared to be successful. Note that the CRC needs to be sufficiently strong so as to act as a reliable signature for the codeword sequence. For this reason, we use a 16-bit checksum, which has a reasonable error performance, in our implementation.

18.6.6 Summary

The bit stream associated with a block is illustrated in Figure 18.13. Depending on the scenario at hand, different subsets of these fields are used to represent the block. For instance, for the case when we are interested in pure compression

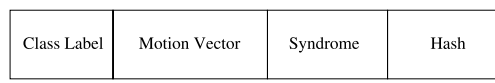


FIGURE 18.13: Bit stream associated with a block.

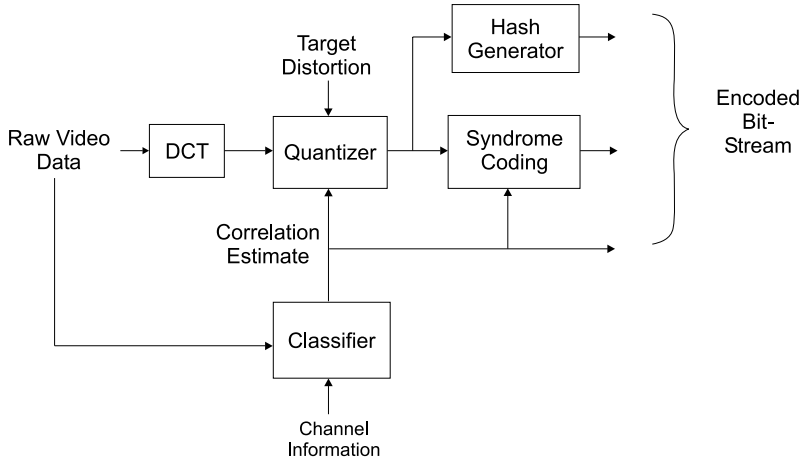


FIGURE 18.14: Functional block diagram of the encoder.

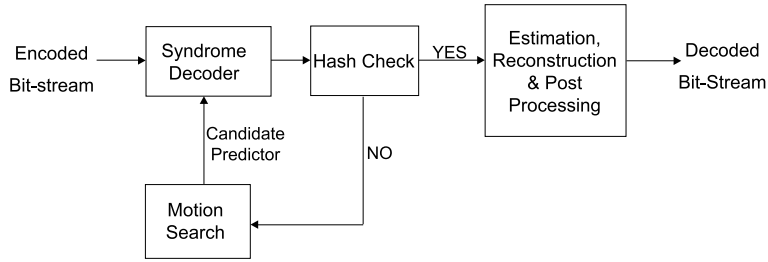


FIGURE 18.15: Functional block diagram of the decoder.

performance, since there is no need for motion search at the decoder, we do not indicate the hash signature associated with a block. Instead, like conventional video compression standards, we indicate the motion vector information for the current block, as determined by the encoder.

Figure 18.14 summarizes the encoding process through the overall encoder block diagram.

18.7 PRISM: DECODING

The block diagram of the PRISM decoder is shown in Figure 18.15. We note that when the decoder does not have to do a motion search, the encoder sends the motion vector that points to the correct side-information block to use and hence

the motion search and CRC check modules in Figure 18.15 are unnecessary. We now describe the main decoder modules.

1. **Generation of Side Information (Motion Search):** The decoder does a motion search to generate candidate predictors, which are tried one by one to decode the sequence of quantized codewords from the set labeled by the received syndrome. In our current implementation, a half-pixel motion search is used to obtain various candidate predictors, as is also done at the encoding side in the standard video algorithms [1,4,10]. We reiterate that the framework is very general so as to accommodate any other sophisticated motion estimation procedures, such as multiframe prediction [18], variable block-sized motion estimation [4], and optical flow [24]. The choice of a more sophisticated algorithm can only serve to enhance the performance of the PRISM paradigm.
2. **Syndrome Decoding:** Each of the candidate predictors generated by the motion search module forms the side-information ($\mathbf{Y}$) for the syndrome decoding step. The syndrome decoding consists of two steps. In the first step, the bits that were entropy coded (the black-colored bit planes in Figure 18.11) are run through an entropy decoder to recover the source bits. If there are no coset channel-coded bit planes (the gray-colored bit plane in Figure 18.11), then the entropy-decoded bits uniquely identify the coset in which the side-information $\mathbf{Y}$ must be decoded. If there is a coset channel-coded bit plane, then the syndrome from the coset channel encoding operation, together with the entropy-decoded bits, specifies the coset in which $\mathbf{Y}$ must be decoded. In the second step, soft decision decoding is performed on the side information, $\mathbf{Y}$, to find the closest codeword within the specified coset. In general, soft decision decoding is computationally intensive for block codes. To reduce the computational burden, we chose to use ordered-statistics decoding [14]. Soft decision decoding based on [14] is near optimal with a loss in performance on the order of 0.2–0.3 dB.
3. **Hash Check:** Since for every candidate predictor, we will decode one codeword sequence from the set of sequences labeled by the syndrome that is nearest to it, the hash signature mechanism is required to infer the codeword sequence intended by the encoder. Thus for every decoded sequence we check if it matches the transmitted hash. If so, then the decoding is declared to be successful. Else using the motion search module, the next candidate predictor is obtained and then the whole procedure is repeated. When correct, the syndrome decoding process recovers the base quantization intervals for the coefficients that are syndrome encoded.
4. **Estimation and Reconstruction:** Once the quantized codeword sequence is recovered, it is used along with the predictor to obtain the best reconstruction of the source. In our current implementation, we use the best mean-squared estimate from the predictor and the quantized codeword

to obtain the source reconstruction. However, any sophisticated signal processing algorithm (e.g., spatiotemporal interpolation) or post processing mechanism can be deployed in this framework, which can only serve to improve the overall performance.

5. **Inverse Transform:** Once all the transform coefficients have been dequantized, the zig-zag scan operation carried out at the encoder is inverted to obtain a two-dimensional block of reconstructed coefficients. The transformed coefficients are then inverted using the inverse transform so as to give reconstructed pixels.

18.8 SIMULATION RESULTS

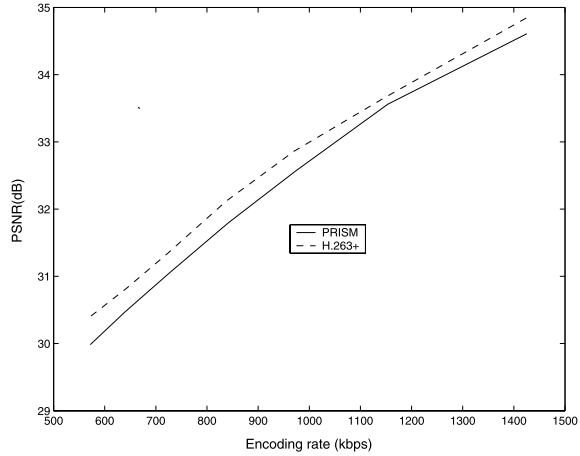
In this section, we present some preliminary simulation results that illustrate the various features of PRISM. As mentioned earlier, we use the block size 8×8 processing primitives for the motion search, DCT, and entropy coding, same as in the H.263+ codec [10] when used in the advanced prediction mode. Thus for the purpose of a fair comparison, we use the H.263+[10] codec as our reference system.

Compression Performance Tests: For tests on pure compression performance, both PRISM and the reference predictive codec use a full-motion search at the encoder. Figure 18.16 shows a comparison of the compression performance of the PRISM video coding algorithm and H.263+ for the first 15 frames of the Football (352×240, 15 fps) and Stefan (176×144, 15 fps) sequences. As can be seen from Figure 18.16, the performance of the proposed scheme nearly matches that of H.263+. This shows that distributed source coding-based video coding can approach the performance of prediction-based coders when it can estimate the correlation structure accurately through the use of good-motion models.

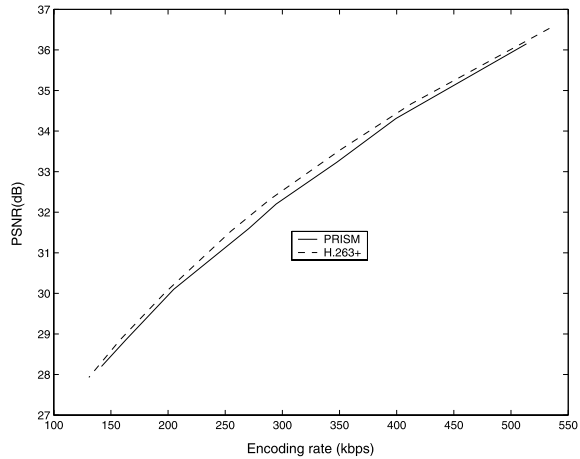
Robustness tests: For robustness tests, PRISM uses the frame difference-based low-complexity classifier. The objective here is to show that even with a low-complexity encoder PRISM can outperform a predictive codec when there are channel losses.

For these robustness tests we used a wireless channel simulator obtained from Qualcomm Inc. This simulator adds packet errors to multimedia data streams transmitted over wireless networks conforming to the CDMA2000 1X standard [2]. (The packet error rates are determined by computing the carrier to interference ratio of the cellular system.) We tested PRISM, H.263+, H.263+ protected with FEC codes (Reed–Solomon codes used, 20% of total rate used for parity bits), and H.263+ with intra-refresh over this simulated wireless channel. Figure 18.17 shows the performance comparison of these four schemes over

[10] Obtained from the University of British Columbia, Vancouver.



(a)



(b)

FIGURE 18.16: Lossless channel results: Comparison of proposed Distributed Video Coding (DVC) algorithm and H.263+ for (a) the Football sequence (352 × 240, 15 fps) and (b) the Stefan sequence (176 × 144, 15 fps).

a range of error rates for the Football (352 × 240, 15 fps, 1700 kbps), Stefan (176 × 144, 15 fps, 720 kbps), Football (176 × 128, 15 fps, 160 kbps), and Flower Garden (176 × 128, 15 fps, 700 kbps) sequences. Figure 18.17 clearly demon-

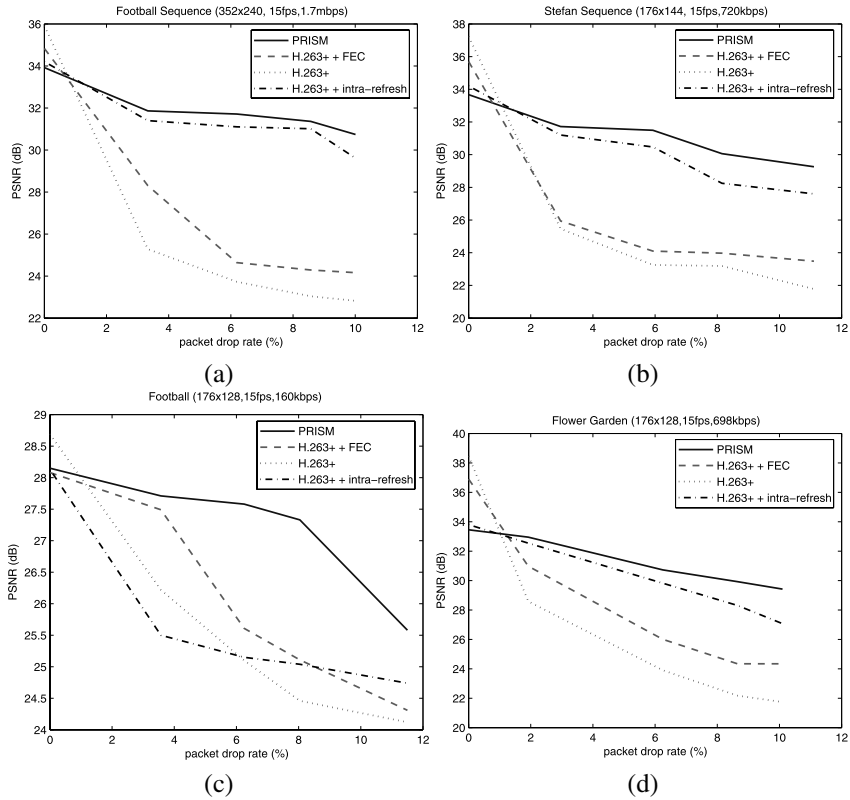


FIGURE 18.17: Lossy channel results: Comparison of the PRISM Distributed Video Coding algorithm, H.263+, H.263+ protected with Forward Error Correcting (FEC) codes (Reed–Solomon codes used, 20% of total rate used for parity bits), and H.263+ with intra-refresh over a simulated CDMA2000 1X channel for (a) the Football sequence (352×240, 15 fps, 1700 kbps), (b) the Stefan sequence (176×144, 15 fps, 720 kbps), (c) the Football sequence (176×128, 15 fps, 160 kbps), and (d) the Flower Garden sequence (176×128, 15 fps, 700 kbps).

strates the superior robustness properties of PRISM. While the decoded quality for the H.263+ system decreases drastically with an increase in packet error rate and saturates at a low value, the decoded quality for PRISM degrades in a graceful fashion.

Figure 18.18 shows the decoded visual quality for the three schemes for the Football (352×240, 15 fps, 1700 kbps) sequence at 8% average error rate. As

(H.263+)

(H.263+ with FEC)

(PRISM)

FIGURE 18.18: Decoded visual quality of the 9th frame of the Football sequence (352 × 240, 15 fps, 1700 kbps) encoded using PRISM, H.263+, and H.263+ protected with Forward Error Correcting (FEC) codes (Reed–Solomon codes used, 20% of total rate used for parity bits). In each case 15 frames were encoded and then sent over a simulated CDMA2000 1X channel. Note that there are very noticeable artifacts for both H.263+ and H.263+ protected with FECs while PRISM has been able to recover from past errors.

can be seen in Figure 18.18, PRISM is able to recover from past errors while error propagation continues to occur for both H.263+ and H.263+ protected with FECs.

As mentioned earlier, PRISM does not do any motion search at the encoder and so loses to H.263+ at a 0% loss rate due to inaccurate modeling of the DFD statistics. However, as channel noise increases, the importance of such accurate modeling diminishes (as described in Section 18.5.3) and the robustness advantages of distributed video coding start to dominate, leading to significant performance gains (over even H.263+ protected with FEC and H.263+ with intra-refresh), as highlighted in Figure 18.17.

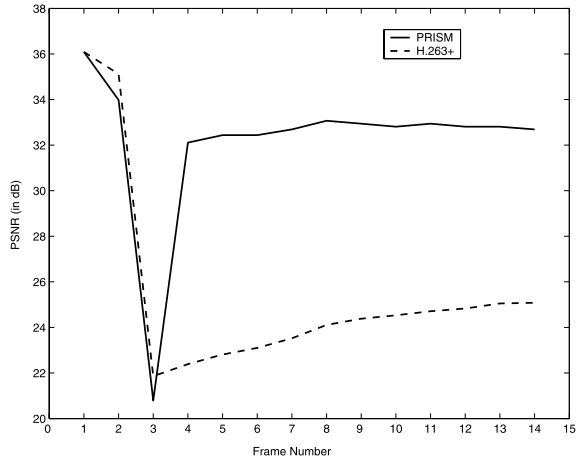


FIGURE 18.19: Effect of frame drops: Comparison of PRISM and H.263+ for the Football sequence (352 $\times$ 240, 15 fps, 1700 kbps) when the third frame was dropped.

Figure 18.19 shows the effect on quality when an entire frame is dropped. For this test the two comparison systems were PRISM and H.263+ and the sequence used was Football (352 $\times$ 240, 15 fps, 1700 kbps). For both PRISM and H.263+, the third frame was dropped. As can be seen from Figure 18.19, the decoded quality in both cases drops drastically for the third frame. However, while the PRISM system recovers quickly and is still able to deliver high quality for subsequent frames, the H.263+ system can only recover to a small extent. This indicates that PRISM can quickly recover from errors.

18.9 SUMMARY AND FURTHER READING

While the simulation results described earlier fuel optimism about the promise of PRISM for uplink-rich media applications, much work remains before a complete codec system can be endorsed. We envisage a scale of effort similar to what has gone into current commercial standards for video compression to make the concepts of PRISM a reality for practical and ubiquitous deployment. There are numerous promising directions for future research.

- More sophisticated motion models need to be integrated into PRISM.
- More sophisticated channel codes based on turbo codes [7,8] and low-density parity check (LDPC) codes [9,15] need to be integrated into the

PRISM fold while keeping intact the block-level motion modeling philosophy of PRISM.

- The classification phase of the codec in estimating the temporal correlation (innovations) noise variance is critical to the performance of PRISM. We plan to direct our future study toward more robust and sophisticated approaches to classification, keying on the complexity versus performance trade-off benefits. The work of [26] on fast motion search appears to be a promising hunting ground for this.

We would like to point out that we have described the PRISM framework primarily in a point-to-point single-camera single receiver setup. Our proposed paradigm, however, scales naturally to scenarios involving multiple-camera applications that we believe will form the cornerstone of emerging video sensor networks. Section 18.9.1 describes an exciting application of multicamera video sensor networks.

18.9.1 Scene Super-Resolution Through the Network

Imagine a dense configuration of cameras conducting surveillance in the parking lot of your office building. These cameras have overlapping coverage and each of these individual cameras is an inexpensive low-resolution device. For instance, each of these cameras can offer a low frame rate (low temporal resolution). An interesting question that arises here is whether all these low-resolution observations can be synergistically combined, providing a "virtual super-resolution" system that allows for enhanced capabilities ranging from novel spatiotemporal viewpoint generation/rendering with robustness to individual camera failures? This is indeed feasible, as is demonstrated by Figure 18.20, which shows three consecutive video frames from two adjacent cameras: A (middle row) and B (top row). Even though the middle frame in stream A (bordered) is missing (e.g., when A operates at half the frame rate of B), sophisticated processing based on camera motion (between A and B), as well as object-motion modeling, enables a near-perfect reconstruction of the missing scene (bottom row).

Additionally, we can also ask if these correlated data can be efficiently compressed for the purpose of archiving/storage. The increasing relevance of this problem can be gauged from the fact that an industry-wide initiative [5] has been launched in the International Standards Organization MPEG group with the purpose of addressing this question.

The caveat here is that our sophisticated processing/compression algorithms require all the frames to be present at one central location. While this is easy to resolve in the high-bandwidth wired network case, where the individual cameras can communicate their respective streams (uncompressed or marginally compressed) to the central processing location, this can be a real daunting task in the low-bandwidth, harsh transmission environment wireless network case. It is here

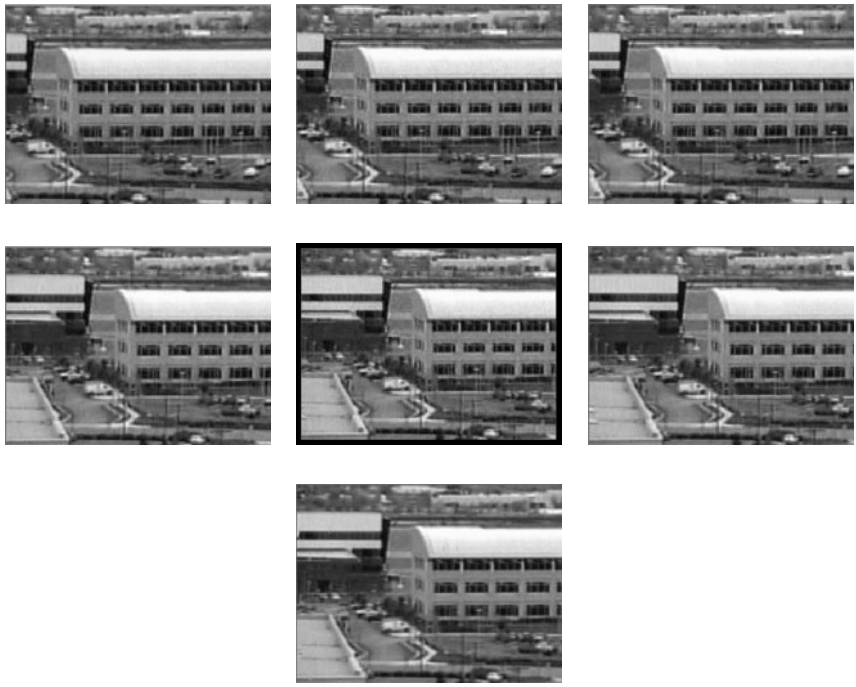


FIGURE 18.20: Top row: video stream from camera B. Middle row: video stream from camera A, including the missing "original" middle frame (bordered). Bottom row: reconstructed missing middle frame from camera A.

that we can use distributed compression algorithms to reduce our transmission bandwidth, as well as provide natural robustness to the vagaries of the wireless transmission environment.

We realize that this problem requires an interdisciplinary approach leveraging the latest advances in the areas of signal and video processing, computer vision, and wireless networking. However, the fundamental architectural features of PRISM, which include robustness as well as an ability to share computational complexity between different network nodes, offer the necessary building blocks that form the core of the solution for this problem. While this problem remains an ongoing challenge for the video networking community at large at this time, promising preliminary efforts are already under way in the research community. The interested reader is referred to [16,19,43] for an overview of the same.

18.9.2 Conclusions

We have introduced PRISM, a practical video coding framework built on distributed source coding principles. Based on a generalization of the classical Wyner–Ziv setup, PRISM is characterized by an inherent system uncertainty about the "state" of the relevant side information that is known at the decoder. The two main architectural goals of PRISM that make it radically different from existing video codecs are (i) flexible distribution of complexity between encoder and decoder (including the special case of reversal of complexities with the decoder picking up the expensive task of motion search, as has been detailed in this work), and (ii) naturally in-built robustness to drift between encoder and decoder caused by a lack of synchronization due to channel loss (as has been demonstrated successfully through simulations in this work). This renders PRISM as an attractive candidate for wireless video applications.

The fundamental architectural traits of PRISM are also well suited for the multicamera regime. Indeed, as the scale of video sensor networks increases in the future, the architectural benefits of PRISM will be magnified. The full potential of large-scale ubiquitous video sensor networks of the future will require an interdisciplinary approach involving signal and video processing, computer vision, multiterminal information theory, and wireless networking. The work presented here represents an important first step toward this goal.

To conclude, our work represents but a tip of the surface of what we believe is an exciting new direction for video coding for a large class of emerging uplink-rich media applications (including broadband wireless video sensor networks). We are optimistic that our work represents a promising start to this exciting journey.

REFERENCES

[1] Information Technology—Generic Coding of Moving Pictures and Associated Audio Information: Video (MPEG-2), 2nd edition. *ISO/IEC JTC 1/SC 29 13818-2*, 2000.

[2] TIA/EIA Interim Standard for CDMA2000 Spread Spectrum Systems. May 2002.

[3] Information Technology—Coding of Audio-Visual Objects – Part 2: Visual (MPEG-4 Visual). *ISO/IEC JTC 1/SC 29 14496-2*, 2004.

[4] Information Technology—Coding of Audio-visual Objects – Part 10: Advanced Video Coding (H.264). *ISO/IEC JTC 1/SC 29 14496-10*, 2005.

[5] Preliminary call for proposals on multi-view video coding. In *ISO/IEC JTC1/SC29/WG11 N7094*, Busan, Korea, April 2005.

[6] A. Aaron, R. Zhang, and B. Girod. "Wyner-Ziv Coding of Motion Video," *36th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, November 2002.

[7] C. Berrou, A. Glavieux, and P. Thitimajshima. "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-codes (1)," *IEEE International Conference on Communications*, 2:1064–1070, May 1993.

[8] J. Chou, S. S. Pradhan, and K. Ramchandran. "Turbo and Trellis-Based Constructions for Source Coding with Side Information," *Proceedings of Data Compression Conference (DCC)*, Snowbird, UT, March, 2003.

[9] S. Y. Chung, G. D. Forney, T. J. Richardson, and R. Urbanke. "On the Design of Low-Density Parity-Check Codes within 0.0045 dB of the Shannon Limit," *IEEE Communications Letters*, 5:58–60, February 2001.

[10] G. Cote, B. Erol, M. Gallant, and F. Kossentini. "H.263+: Video Coding at Low Bit Rates," *IEEE Transactions on Circuits and Systems for Video Technology*, 8(7):849–866, November 1998.

[11] T. M. Cover and J. A. Thomas. *Elements of Information Theory*, John Wiley and Sons, New York, 1991.

[12] G. D. Forney. "Coset Codes-Part I: Introduction and Geometrical Classification," *IEEE Transactions on Information Theory*, 34(5):1123–1151, September 1988.

[13] G. D. Forney. "Coset Codes-Part II: Binary Lattices and Related Codes," *IEEE Transactions on Information Theory*, 34(5):1152–1187, September 1988.

[14] M. P. C. Fossorier and S. Lin. "Soft Decision Decoding of Linear Block Codes Based on Order Statistics," *IEEE Transactions on Information Theory*, 41(5):1379–1396, September 1995.

[15] R. G. Gallager. "Low Density Parity Check Codes," Ph.D. Thesis, MIT, Cambridge, MA, 1963.

[16] N. Gehrig and P. L. Dragotti. "Different-Distributed and Fully Flexible Image Encoders for Camera Sensor Network," in *International Conference on Image Processing*, Genova, Italy, September 2005.

[17] B. Girod, A. Aaron, S. Rane, and D. Rebollo-Monedero. "Distributed Video Coding," in *Proceedings of IEEE*, January 2005.

[18] B. Girod and T. Wiegand. *Multiframe Motion-Compensated Prediction for Video Transmission*. Kluwer Academic Publishers, 2001.

[19] D. A. Hazen, R. Puri, and K. Ramchandran. "Multi-Camera Video Resolution Enhancement by Fusion of Spatial Disparity and Temporal Motion Fields," in *IEEE International Conference on Computer Vision Systems*, New York City, NY, January 2006.

[20] P. Ishwar, V. M. Prabhakaran, and K. Ramchandran. "Towards a Theory for Video Coding Using Distributed Compression Principles," *Proceedings of IEEE International Conference on Image Processing (ICIP)*, Barcelona, Spain, September 2003.

[21] P. Ishwar, R. Puri, A. Majumdar, and K. Ramchandran. "Analysis of Motion-Complexity and Robustness for Video Transmission," in *Proceedings of Wireless-Com*, 2005.

[22] A. Jagmohan, A. Sehgal, and N. Ahuja. "Predictive Encoding Using Coset Codes," *Proceedings of IEEE International Conference on Image Processing (ICIP)*, Rochester, NY, September 2002.

[23] M. Johnson, P. Ishwar, V. Prabhakaran, D. Schonberg, and K. Ramchandran. "On Compressing Encrypted Data," *IEEE Transactions on Signal Processing*, 52:2992–3006, October 2004.

[24] R. Krishnamurthy, J. M. Woods, and P. Moulin. "Frame Interpolation and Bidirectional Prediction of Video Using Compactly-Encoded Optical Flow Fields and Label Fields," *IEEE Transactions on Circuits and Systems for Video Technology*, 9(5):713–726, August 1999.

[25] G. G. Langdon, Jr. "An Introduction to Arithmetic Coding," *IBM Journal of Research and Development*, 28(2):135–149, March 1984.

[26] K. Lengwehasatit and A. Ortega. "Probabilistic Partial Distance Fast Matching for Motion Estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, 11(2):139–152, February 2001.

[27] F. J. Macwilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes*, Elseiver-North-Holland, 1977.

[28] A. Majumdar. "PRISM: A Video Coding Paradigm Based on Source Coding with Side Information," Ph.D. Thesis, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, December 2005.

[29] A. Majumdar, J. Wang, K. Ramchandran, and H. Garudadri. "Drift Reduction in Predictive Video Transmission Using a Distributed Source Coded Side-Channel," *Proceedings of ACM Multimedia*, New York, NY, September 2004.

[30] S. S. Pradhan, J. Chou, and K. Ramchandran. "Duality between Source Coding and Channel Coding and Its Extension to the Side Information Case," *International Transactions on Information Theory*, 49(5):1181–2003, July 2003.

[31] S. S. Pradhan and K. Ramchandran. "Distributed Source Coding Using Syndromes (DISCUS): Design and Construction," *Proceedings of Data Compression Conference (DCC)*, Snowbird, UT, March 1999.

[32] R. Puri and K. Ramchandran. "PRISM: A New Robust Video Coding Architecture Based on Distributed Compression Principles," *40th Allerton Conference on Communication, Control and Computing*, Allerton, IL, October 2002.

[33] R. Puri and K. Ramchandran. "PRISM: A 'Reversed' Multimedia Coding Paradigm," *IEEE International Conference on Image Processing*, Barcelona, Spain, September 2003.

[34] R. Puri and K. Ramchandran. "PRISM: An Uplink-Friendly Multimedia Coding Paradigm," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Hong Kong, April 2003.

[35] W. Rabiner and A. P. Chandrakasan. "Network-Driven Motion Estimation for Wireless Video Terminals," *IEEE Transactions on Circuits and Systems for Video Technology*, 7(4):644–653, August 1997.

[36] A. Sehgal, A. Jagmohan, and N. Ahuja. "Scalable Video Coding Using Wyner–Ziv Codes," *Proceedings of Picture Coding Symposium (PCS)*, San Francisco, CA, December 2004.

[37] A. Sehgal, A. Jagmohan, and N. Ahuja. "Wyner-Ziv Coding of Video: An Error-Resilient Compression Framework," *IEEE Transactions on Multimedia*, 6(2): 249–258, April 2004.

[38] D. Slepian and J. K. Wolf. "Noiseless Coding of Correlated Information Sources," *IEEE Transactions on Information Theory*, 19: 471–480, July 1973.

[39] H. S. Witsenhausen and A. D. Wyner. Interframe coder for video signals. *United States Patent* (4191970), March 1980.

[40] A. D. Wyner and J. Ziv. "The Rate-Distortion Function for Source Coding with Side Information at the Decoder," *IEEE Transactions on Information Theory*, 22(1): 1–10, January 1976.

[41] Q. Xu, V. Stankovic, and Z. Xiong. "Layered Wyner–Ziv Video Coding with IRA Codes for Noisy Channel," in *Proceedings of Visual Communications and Image Processing*, Beijing, China, July 2005.

[42] R. Zamir. "The Rate Loss in the Wyner–Ziv Problem," *IEEE Transactions on Information Theory*, 42(6):2073–2084, November 1996.

[43] X. Zhu, A. Aaron, and B. Girod. "Distributed Compression for Large Camera Arrays," in *IEEE Workshop on Statistical Signal Processing*, St. Louis, Missouri, September 2003.

This page intentionally left blank

19

Infrastructure-Based Streaming Media Overlay Networks

Susie Wee, Wai-Tian Tan, and John Apostolopoulos

19.1 INTRODUCTION

Technology advances are giving people increasingly immersive multimedia experiences in their home entertainment systems, on their portable media players, on their desktop and laptop computers, and even on their mobile phones. While IP networks provide unprecedented connectivity between people and devices for data applications, even today, large fractions of the network are not suitable for the high bandwidths and real-time streaming requirements of multimedia applications. For example, the Internet provides connectivity between any two end nodes on the Internet; however, it only provides best-effort service and therefore provides no guarantees on the available bandwidth, maximum delay or delay jitter, or loss rates. Thus, there remains a challenge for providing high-quality media between people and devices over large portions of the network, and these problems are amplified as the number of people trying to use the network increases.

In this chapter, we describe the basic concepts and architecture of a media overlay, which adds resources to an existing network infrastructure to enhance the media capability of the network. A media overlay can enable new media capabilities in the network, while improving the end-user media performance and the system-wide efficiency of the network for both its media and nonmedia traffic. This is achieved by leveraging the underlying resources and existing connectivity provided by the original network, while enhancing it to improve its ability to deliver real-time media to end users and scaling to support a large number of users.

The term "overlay" refers to the approach of adding resources on top of an existing network infrastructure, as shown in Figure 19.1. This has the advantage

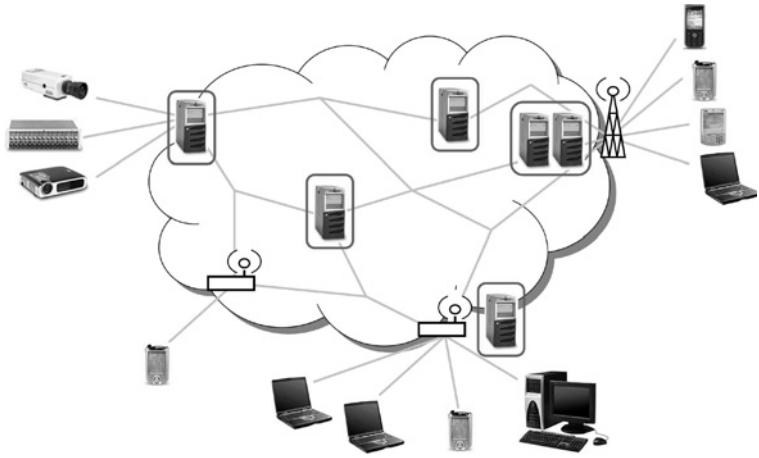


FIGURE 19.1: Media overlays add resources to an existing network to enable new media capabilities and to improve end-user media performance, scaling to support large numbers of users, and system-wide efficiency.

of leveraging the attributes of the existing infrastructure, such as its existing deployment, widespread connectivity, and built-in network services such as domain name services (DNS) and system management. While the existing network has some inherent capabilities, the overlay provides additional capabilities to achieve an extended set of goals, such as enhancing the media capabilities of a network, improving the operational efficiency and system-wide performance of the network itself, and improving the user-perceived performance of media applications.

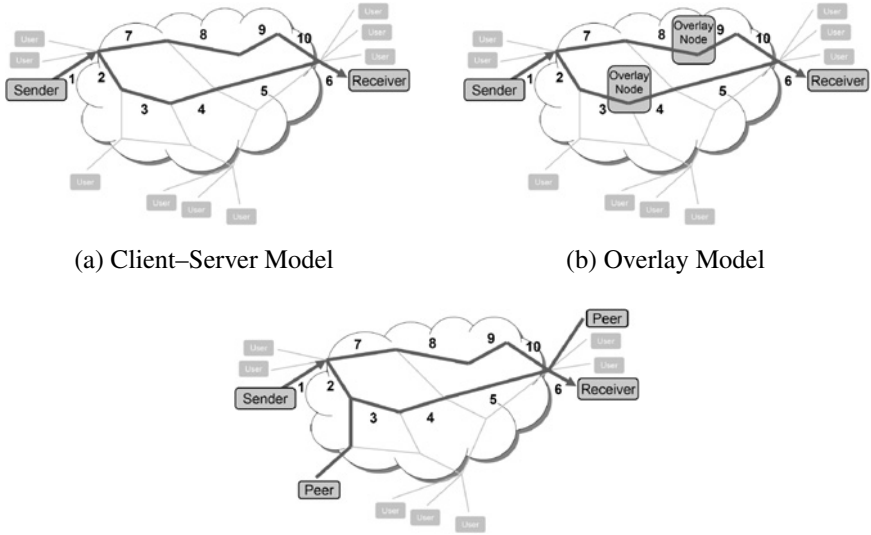
The benefit of an overlay can be illustrated with a simple example. Consider a corporation that wants to stream corporate webcasts to their employees over their existing corporate intranet. In most cases, the existing intranet will not have the capacity to support a centralized streaming service for all their employees, where a single server would stream a separate unicast stream for each employee. In this case, a media overlay can provide distributed caching to replicate the webcast content at overlay nodes close to large employee sites. The employee requests would then be served by streaming the replica on the nearest overlay node rather than from the centralized server. In this example, the media overlay allowed the webcast application to be provided on an existing corporate intranet using the overlay capabilities of media caching and streaming, and it was performed in a manner that significantly reduced the network load on the corporate intranet.

19.1.1 Comparing Client–Server, Overlay, and Peer-to-Peer Models

To better understand media overlays, it is helpful to consider the traditional capabilities provided by current networks and the relative contributions of a media overlay to its alternatives. In this chapter we focus on infrastructure-based overlay systems. Some peer-to-peer systems can be considered as client-based overlays [11], but to simplify terminology we use the term overlay to refer to infrastructure-based overlays.

A challenge to all successful networks is the possibility of having usage demand outgrow the original capacity or design. In order to maintain or improve the performance of existing services or to provide new services, enhancement mechanisms need to be added in a fashion transparent to existing clients, without prolonged disruption to the services. One example of such evolution includes Intelligent Networks (IN) in telecommunications networks, which provide functions such as three-way calling and call waiting for a network originally designed for point-to-point voice communications. Another example is Content Distribution Networks (CDN) for the Internet, which provide reduced access latency for end users and improves scalability to a larger number of users by having multiple caches near the end users [20]. With an ever-increasing growth of multimedia access, especially for video, in both cellular networks and the Internet, it is becoming important to examine mechanisms to provide improved performance and added features for multimedia communications. In this chapter, similar to what INs have done for telephone networks and CDNs for the Internet, our focus will be on improvement in the existing delivery infrastructure itself, rather than developing a completely new infrastructure or assuming a peer-to-peer architecture.

The traditional approach to supporting streaming media and Web traffic alike is to use the Client–Server model of Figure 19.2a where a single server is in charge of serving all clients. As the client population grows, this model hits two limitations. First, a single server cannot scale to serve an ever-increasing number of clients. This problem is sometimes solved by implementing a logical server by multiple physical servers. Second, since a server cannot be simultaneously close to all clients, access latency for some clients is bound to be long. The use of an overlay infrastructure, as shown in Figure 19.2b, is one possible solution to the limitations of the client–server approach. An overlay network is a logical network that relies on a physical network for connectivity service. Specifically, in Figure 19.2b, having nodes A and B inside the infrastructure allows content to be served from multiple possible servers, and specifically for each requesting client we can choose the "closest" server to serve the content. A key point is that an overlay infrastructure is designed based on the underlying network, as well as expected client demands and desired services. An important example of commercially deployed overlay infrastructure are CDNs for Web traffic. So far, we have only discussed the use of an overlay infrastructure for caching purposes.



(a) Client–Server Model

(b) Overlay Model

(c) Peer-to-Peer Model

FIGURE 19.2: (a) The classical Client–Server Model assumes only basic connectivity service from the network. (b) The Overlay Model uses the same connectivity service, but also has overlay nodes inside the infrastructure to provide computation and storage inside the infrastructure. (c) The Peer-to-Peer Model, however, relies on peer nodes outside the infrastructure to provide storage and computation.

Other benefits include the possibility of performing additional services and the provisioning of multiple paths. For example, the minimum-hop path between the client and the server in Figure 19.2 traverses links 1, 2, 3, 4, 5, and 6 and involves a bottleneck link 3. Using an overlay infrastructure for relaying, it is possible to effectively use another path 1, 7, 8, 9, 10, 6 even when the physical network does not offer routing choices.

Another approach used to overcome the limitations of the client–server model is the peer-to-peer model (P2P) of Figure 19.2c. Common for file-swapping applications in which a large number of clients want to receive the same piece of content, the P2P architecture allows every client to access the already-downloaded portions of other clients. In return, every client would make available the portion of content it has downloaded. A P2P network is therefore self-scaling in the sense that the number of requesting nodes and potential serving nodes is equal. Therefore, P2P architectures have the advantage of supporting potentially very large active client populations. In addition, for popular content it is possible for a client

Table 19.1: Comparison among client–server, overlay infrastructure, and peer-to-peer systems for three different classes of metrics.

	Metric	Client–server	Overlay	Peer-to-peer
User-centric performance	Availability	Bad	Good	Good–poor
	Latency	Bad	Good	Good–poor
	Streaming quality	Bad	Good	Average
	File transfer quality	Bad	Good	Best
System-wide efficiency	Low control overhead	Best	Good	Poor
	Bandwidth usage	Bad	Good	Good
	Control/manageability	Good	Good	Bad
Service extensibility	Ease to add services and capabilities	Good	Good	Poor

to download from a closer location under P2P than is possible under infrastructure overlay. In P2P systems the peers view network and infrastructure as a black box providing connectivity, and the P2P intelligence resides at the end hosts. One major disadvantage of P2P is the lack of control of *peers* in terms of population size, availability, and actions, making it difficult to provide predictable quality of service. For example, P2P networks are typically afflicted by sizable churn where the peers may come and go. Another disadvantage is the large amount of control traffic necessary in a typical implementation to maintain the distribution structure. Furthermore, fairness is a difficult problem in P2P systems. An overview of when P2P systems or infrastructure solutions are preferred based on desired application attributes is given in [31].

In this chapter, we focus on infrastructure-based overlay networks that are designed and deployed with the underlying network resources in mind [18]. A simplistic overlay network can be constructed, for example, by using the system shown in Figure 19.2c, where content distribution is performed in a way that the server streams to P, which in turns relays to P'. In this case, P is a new physical resource that enables an overlay relaying service to P', but is impractical due to the lack of resource planning.

In Table 19.1, we compare the relative merits of the client–server model, the overlay model, and the P2P model in the following three classes of metrics. The first class is user centric and involves access latency and availability for a single user. The second class relates to system-wide performance. The third class focuses on the ability to introduce new services. We see that the overlay infrastructure provides a useful balance among the three classes of objectives.

19.1.2 Overview of a Streaming Media Overlay

A streaming media overlay is designed to provide a number of basic capabilities, such as media streaming, caching, content distribution, resource monitor-

ing, resource management, and signaling. These capabilities are discussed in Section 19.2. An overlay may also have advanced capabilities that can perform session management of streaming media sessions, request redirection, load balancing, caching, and relaying media streams. Furthermore, an overlay can perform media processing operations such as transcoding to adapt media streams for different display sizes and network capacities, perform logo insertion to personalize media streams for individuals or locations, or voice/video activity detection to enable applications such as multiuser conferencing, which is discussed in Sections 19.2 and 19.5. These capabilities can be upgraded over time to provide new media services.

Existing network infrastructure can be viewed as a number of layers, as shown in Figure 19.3. At the base level, there is a base-wired/wireless IP network that, for example, could be a telecom network, an enterprise, the open Internet, a hot spot network, or a home network. Any of these networks may have a series of elementary network services in them. For example, a telecom network may have a home location registry that tracks client movements. The next layer may contain a number of overlay network subsystems. The media overlay and media service network reside at this layer. Finally, a system and network management layer may exist to monitor and manage the entire network and systems.

Since the overlay becomes an integral part of the infrastructure, it is important that it respects the existing network applications and perhaps even improves them. Thus, manageability is critical to an overlay's success. Thus, the overlay nodes perform a valuable distributed network monitoring function to make network and system measurements based on their observations of network and system load

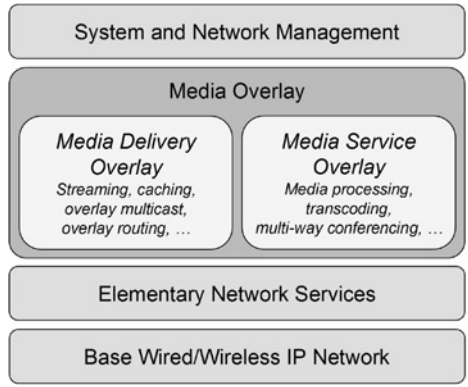


FIGURE 19.3: A media overlay is an integrated part of an extended system architecture. It exploits the base network and elementary network services for data delivery, but may itself be managed by other entities.

and client requests. These measurements can be aggregated and analyzed to make decisions on how to handle media streams in the overlay. For example, it may decide to cache popular content at overlay nodes closer to end users, reroute streams to avoid congestion points, or transcode media streams to adapt them for lower bandwidth links.

The overlay can be architected in a manner that provides the benefits of incremental deployment and upgradeability. Thus, a small overlay deployment can be used to provide an initial media service or application. Then, more resources can be deployed over time based on the measured usage of the application. Furthermore, the capabilities of the overlay can be upgraded to provide new services or new capabilities. This can be done by adding additional overlay resources or upgrading the existing resources with new capabilities.

19.1.3 Chapter Outline

This chapter continues in Section 19.2 by examining the basic capabilities that may be provided by a streaming media infrastructure, including an example to illustrate the use of these capabilities. The architectural and design considerations that arise when designing and operating this infrastructure are discussed in Section 19.3. More advanced functions that may be provided by a streaming media infrastructure are then discussed in Section 19.4. The benefits of such streaming media overlays are discussed in Section 19.5. The chapter concludes by providing a summary and pointers for further reading in Section 19.6.

19.2 CAPABILITIES OF A STREAMING MEDIA OVERLAY

A streaming media overlay infrastructure can enable large-scale media delivery on an existing network infrastructure. A media overlay can improve the performance of a media delivery system in a number of ways. For example, if a requested media stream is cached on the overlay, the request can be served from the overlay, thereby reducing the latency of the streaming session. Also, since the overlay server can be colocated at the network edge, it can quickly adapt the streaming session to the rapid variations of the last link; for example, it can be colocated at a wireless base station to adapt to a rapidly varying wireless channel. In addition, since overlay servers are located at intermediate locations in the network, they can act as monitoring points in the network and provide system and network load information that can be used to improve the performance of the overall system.

19.2.1 Media Overlay Server Capabilities

A media overlay adds resources on top of an existing network to enhance its media capabilities. These added resources can include overlay servers and managers

that can be placed in the middle and at the edge of the network, as shown in Figure 19.4. Overlay servers are the basic building block of a media overlay, and they can have capabilities of streaming, caching and content distribution, resource monitoring and management, signaling, and possibly media processing [26]. They can be used to store or cache media streams in the network and to relay media streams across the network. Furthermore, overlay servers can monitor and log the conditions of their surroundings and gather valuable network health information.

The overlay server's basic *streaming* capabilities allow it to send and receive media streams to and from streaming servers and clients. The overlay server is capable of handling many simultaneous input and output streams; it uses a scheduler to coordinate the streaming of these sessions. Furthermore, the overlay server is able to start, stop, and pause its outgoing streaming media sessions and record its incoming streaming media sessions. Handoff capabilities during streaming sessions can also be an important capability for streaming media overlay servers, in particular because of the long-lived nature of streaming media sessions.

The overlay server has *caching* capabilities that allow it to store requested media content for future requests. An overlay server can obtain media data to be cached via data transfer or streaming modes. The resulting cached media streams can then be transmitted as data transfers or streaming sessions as well.

Media content can also be distributed across the overlay infrastructure using the overlay's *content distribution* capabilities. This allows the overlay servers to transfer media streams to other overlay servers, web servers, and media servers. Cached media content can be possibly locked for a specified period of time to prevent premature eviction.

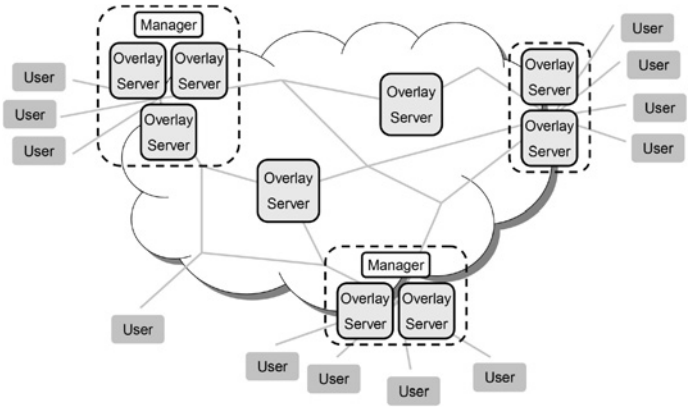


FIGURE 19.4: Overlay resources are placed at strategic locations in a network. Each location may contain one or more overlay servers and/or managers.

The overlay server has *resource monitoring and management* capabilities that allow it to monitor and log its observations over time and to share these logs with other overlay servers and managers through the control/management interface. Overlay servers can track user requests as well as server load and observed network conditions over time. These logs can be gathered and analyzed to improve the performance of the media overlay.

Overlay servers have *coordination* capabilities that allow them to query or track the contents of other overlay servers for requests that are not in its cache. It can have parent/child or sibling relationships with other overlay servers, which can be set or changed through the management and control interface.

These capabilities are discussed in more detail in the following sections.

19.2.2 Media Transport and Streaming

Media overlay servers must be able to transport media to and from sources, other overlay servers, and clients. Media transport can be performed in different ways depending on the needs of the application and the capabilities of the origin servers and clients.

Sending media to clients: A media overlay must be able to deliver media to clients using protocols that the clients understand. Download media clients receive media streams in a file or data transfer mode using protocols such as TCP/IP or HTTP. These clients download the entire media file and store them on a local disk and then allow a user to play the locally stored media content at any time after the download. Streaming media clients receive packetized media content in a streaming mode using protocols such as UDP or RTP streaming or HTTP streaming. These clients store the received media packets in a receiver buffer and then decode and play back these media packets after a short delay. Note that when protocols such as UDP or RTP are used, it is possible that some packets will be lost before reaching the client. In this case, it is important that the media client has error-resilience capabilities to be able to deal with lost packets effectively [40]. Further discussion on error-resilience can be found in Chapters 2 and 3.

Receiving media from media sources: A media overlay must be able to receive media from various media sources that are not part of the media overlay using protocols that the media sources understand. Media sources such as web servers and content servers and media upload clients often use file and data transfer protocols such as TCP/IP, HTTP, or even FTP for reliable media transport. Media sources also include streaming servers and live media recorders that use streaming protocols such as UDP or RTP streaming and HTTP streaming. Note that if UDP or RTP streaming is used and an overlay server wants to cache or store the stream, then it is possible that some media packets will be lost and the stored media content will have some errors. In this case, the overlay server needs to know how to handle these streams by only transferring these streams to clients

with error-resilience capabilities or by repairing the stream into a recognizable media stream format before sending the media content to clients that do not have error-resilience capabilities.

Media transport between overlay servers: In addition to delivering media to and from media clients and sources, media overlay servers can also transport streams with each other. This can be done in data or file transport modes to perform operations such as content distribution, prefetching, and caching. This can also be done using streaming connections to perform operations such as stream redirection, stream relay, and stream splitting.

Stream relay: Overlay servers that can send and receive media streams through streaming connections allow the overlay to perform stream relay tasks. Stream relay can be used to explicitly route or redirect streams in the network, for example, to avoid network bottlenecks that are detected by the overlay. Routing streams through overlay servers can improve operational efficiency of the network by exploiting routes with underutilized resources, while allowing health monitoring of the network or media stream by detecting packet loss or performing bandwidth and latency measurements using the media packets.

Note that the overlay can be designed so that it is transparent as to whether the media is going to or coming from a media source, another overlay server, or a media client. This design allows media streams to be relayed through any number of overlay servers, allowing for various degrees of precision in stream routing and network health monitoring.

Stream splitting: A key problem in networked multimedia is supporting popular events with many users and supporting one-to-many communication [17]. While IP Multicast is a possible solution to this problem, difficulties arise when media streams must traverse networks that do not have IP Multicast support. In these cases, stream splitting, also referred to as overlay or application-level multicast, can be used to increase the scalability of the system and improve network efficiency.

Stream splitting is also useful if a sending device is capable of serving only one stream but many clients are interested in receiving that stream. In that case, the original stream can be sent to an overlay server, which can relay it to multiple downstream devices, as shown in Figure 19.5. This is called stream splitting, overlay multicast, or application-layer multicast. Stream splitting has the advantage that it can be used to provide multicast-like capabilities on networks that do not have native IP multicast support.

19.2.3 Media Distribution and Caching

Media distribution across overlay infrastructure: Placing media content on an overlay server close to a requesting client can lead to the media being streamed over a shorter network path, thus reducing the start-up latency of a streaming ses-

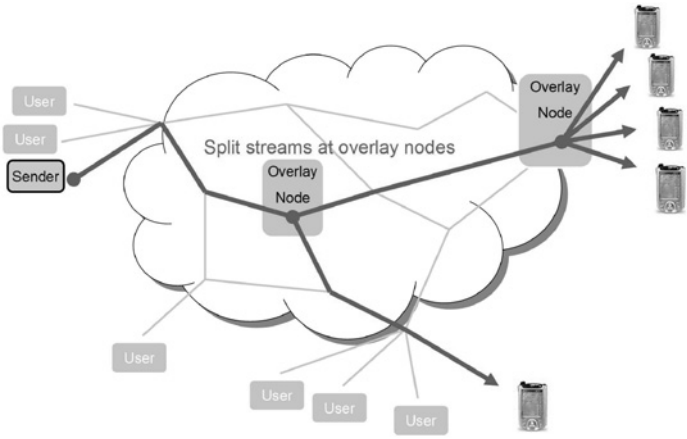


FIGURE 19.5: Stream relay and stream multicast capabilities allow overlay nodes to form flexible media distribution paths, as illustrated here in the form of a multicast tree. Other benefits include the ability to route or redirect media streams around failed links, perform network health monitoring operations by analyzing the media packets, and efficiently deliver media streams to many users.

sion, the probability of packet loss, and the total network usage. This motivates the need for media distribution algorithms that optimize system performance based on the predicted demand. These optimizations can be performed by aggregating measured statistics and developing predictive prefetching algorithms based on statistical analysis. Specifically, the prediction can be based on the content request patterns monitored, logged, and reported by the overlay servers through control and management interfaces. Therefore, important issues relate to the number of caching nodes, their placement in the network based on traffic demands, the predistribution of media content across the caching infrastructure, and the dynamic cache allocation within a single node as described next [27].

Media caching: Closely related is the problem of media caching on the overlay servers. The goal of improving the cache hit rate makes it desirable to store large numbers of media streams on the overlay servers. However, since media streams can require large amounts of storage, storing entire media streams in a cache is clearly inefficient. Thus, the media caching problem involves determining which media streams [23] to cache. These decisions can be based on a number of factors, such as media popularity, size, cacheability, and other factors such as premium content versus free content. Media distribution and caching are critical

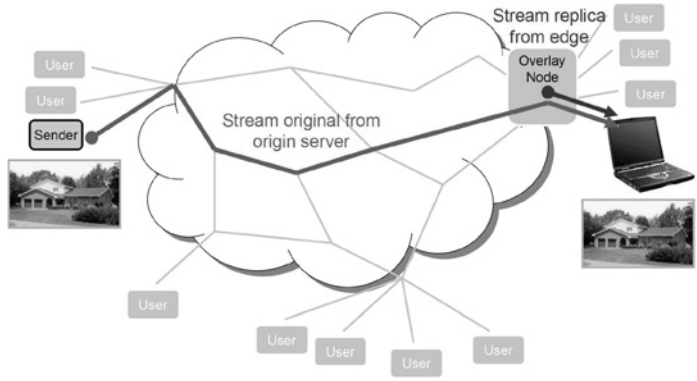


FIGURE 19.6: A media overlay improves network efficiency by allowing a cached replica of the media to be streamed from the edge rather than from the origin.

components of a streaming media infrastructure as they can lead to considerable improvements in resource utilization and system reliability (Figure 19.6).

Media segmentation: Another interesting overlay capability is media segmentation, where a long media stream is divided into media segments that can then be distributed across the network and cached in various overlay servers [8,34]. These segments can be cached in a manner that is aligned with user-viewing statistics. For example, an overlay may stream a live sporting event to many users. At the same time, it may record and store the stream for users who wish to view the event at a later time. However, it may turn out that the most viewed segments of a sporting event are where the goals and highlights occur. The overlay can track this behavior and then cache the various segments according to the user-viewing statistics. For example, the most watched highlight segments can be replicated more frequently than the less viewed segments.

19.2.4 Client Request Handling

There are a number of ways a client can gain access to the services in a media overlay. In the simplest model, a client directly accesses a service portal that is part of the media overlay. Examples discussed in further details later include recording and retrieval of multimedia messages. Alternatively, if a client is attempting to access portals outside the overlay, additional mechanisms are necessary in order to invoke overlay services. If a request always traverses the overlay network, explicit detection and redirection are possible. Otherwise, common techniques include explicit proxy configuration at the clients or DNS-based redirection [5].

After a request reaches a portal in an overlay node, the client needs to be redirected to a server with appropriate capability to satisfy the request. This requires determining which is the "best" overlay server for serving the client, where best is determined by a number of metrics, including which server(s) has the desired content, as well as the server and network loads and capabilities such as transcoding. To evaluate the suitability of each server requires a system monitoring and management component for gathering and processing this information in a timely manner. An example of an architecture designed for monitoring the server and network load of overlay servers and assigning requests to the least-loaded, available edge server is given in [32]. Once the best server is determined the client redirection to that server can be achieved through a number of mechanisms, such as techniques based on DNS [5]. Alternatively, redirection to an appropriate server can happen through mechanisms such as dynamic SMIL rewriting [46].

19.2.5 System Monitoring and Management

System monitoring: Monitoring is an important capability in an overlay network. As mentioned before, statistics such as server and network load allow selection of appropriate servers for satisfying a particular request. Generally, monitoring enables other adaptation on a finer or coarser timescale. For example, real-time network and server statistics can guide short timescale adaptations such as transcoding or other media adaptation, and server handoffs. Longer timescale statistics, however, facilitate resource planning, such as reservation of resources for "flash crowds."

The application awareness of a media overlay allows collection of semantically meaningful statistics, such as the response time of a request to retrieve a stored multimedia message. Furthermore, the location of overlay nodes inside the infrastructure and its application awareness allow collection of better statistics than is possible between end points only. For example, in a traditional client–server streaming setting, a server has limited visibility for the state of the network since it can only observe aggregate conditions along the entire path, and only from traffic originating from itself. An overlay node, however, can observe statistics for a segment shorter than the end-to-end path, and it can observe flows from all servers traversing through it. By observing the network statistics on shorter network path segments, the overlay can achieve improved streaming performance by providing functions such as network-adaptive streaming, as discussed in Section 19.4.1. Furthermore, an overlay can collect certain useful statistics that are not possible without application awareness. For example, an overlay node can differentiate adaptive UDP flows from nonadaptive UDP flows, which are generally difficult to differentiate [14].

System management: Management is another important feature of a media overlay. While monitoring allows the overlay to gather system statistics, man-

agement allows the overlay to analyze the gathered data and assert control on various parts of the overlay and system as a whole. Management capabilities allow managers and overlay servers to query other media overlay components for their monitored information, such as content usage statistics, server load, and network congestion, and to give commands to other media overlay components. This allows the overlay servers to cooperate and act as a system to collect and analyze statistics, predict behaviors from these statistics, and perform tasks to serve predicted user requests in a resource-efficient manner. Furthermore, this allows the overlay servers to work cooperatively to handle the media delivery load in response to changing user patterns and time-varying network and system loads.

An overlay's control and management capabilities also allow the media overlay to receive information from other layers of the network, as shown in Figure 19.3. For example, the control interface can allow a home location registry to pass client mobility information to the overlay server to trigger a streaming server handoff. Furthermore, it allows the overlay to perform operational roles, such as adding or reconfiguring overlay servers and shutting them down for maintenance.

19.2.6 Media Processing Services

Up to this point, this chapter has discussed how an overlay can be used for delivering media streams across a network. In this section, we discuss another important capability that the overlay can provide, namely performing media processing operations on the media streams that are transported by the overlay. We refer to networked media processing operations performed by overlay servers as *media processing services* or, more simply, *media services* provided by the overlay.

Media processing: An overlay server may be able to perform a media service such as media transcoding to adapt media streams for diverse client capabilities and changing network conditions [6,19,36,43]. For example, Figure 19.7 shows a high-resolution, high-bandwidth media stream being simultaneously delivered to two clients with different display capabilities and network connections. The first client may be able to receive and decode the entire high-resolution, high-bandwidth media stream. However, a second client with a lower display resolution and slower network connection may not be able to receive and decode the entire stream. In this case, the overlay can transcode the media stream to a lower resolution and bit rate and then relay the transcoded stream to the low-resolution client.

Other media processing services that an overlay can provide include video and audio processing operations such as VCR functionalities, speed-up/slow-down, logo insertion, background removal, and noise reduction. Overlays can also provide conferencing services such as video tiling, speaker detection, and speaker focus.

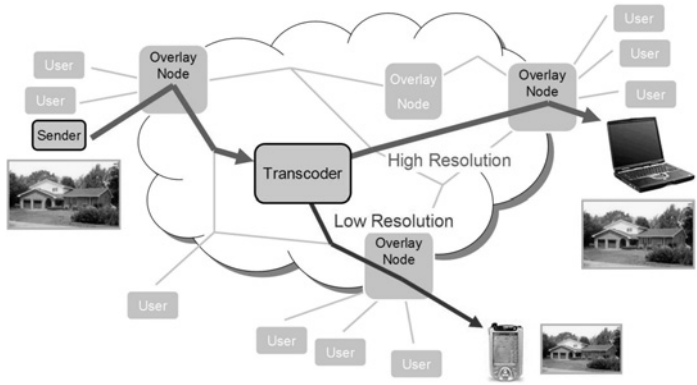


FIGURE 19.7: Performing live transcoding inside an overlay simplifies requirements on both clients and senders. To support the multicast application shown, the clients and senders need only support one stream and do not need the resource or algorithm to perform transcoding.

For mobile multimedia applications, it may be useful for overlay servers to be able to perform midstream handoffs of streaming sessions to adapt to user movements across cell sites. If a transcoding service is provided in a mobile overlay network, it may also be useful for overlay servers to be able to perform midstream handoffs of live transcoding sessions [33].

Notice that when an overlay performs media processing services in the middle of the network, it often operates on input compressed media streams and produces output compressed media streams. Some of these processing operations can be compute-intensive. Since an overlay server may need to process many media streams at once, it is useful to develop computationally efficient algorithms for processing compressed streams [44]. This is a research area that has been examined for many years, and technology advances are now allowing these media processing services to be performed in real time.

Media services architecture: In addition to being able to perform media processing operations, the overlay must be able to manage the media services. Thus, it is important to have a *media services architecture* that allows the media services to be deployed, operated, and managed in the overlay [16] in a manner that works smoothly with the media delivery capabilities of the overlay. This requires the overlay to be able to deploy new media service capabilities on existing overlay servers, track which servers have which media service capabilities, and redirect media streams to the appropriate overlay servers for a streaming session. In addition, the overlay should be able to track the load of the various overlay

servers to ensure that a chosen server is capable of handling the additional streaming task.

19.2.7 Example Walk-Through of Media Overlay Usage

Figure 19.8 shows an illustration of one possible way in which different functions discussed in Sections 19.2.3 to 19.2.5 can act together. Other choices are possible and are discussed further in Section 19.3.

In Figure 19.8, two events happen before a client can access a piece of content from a media overlay. First, information such as availability of contents and server load are continuously collected, as shown by step 0 in Figure 19.8. In Figure 19.8, the information is collected centrally by a *Location Manager*. Such exchanges can be conveniently implemented using standardized protocols such as the Simple Object Access Protocol, and possibly *digests* [30] of server contents are exchanged to reduce communication cost. Second is the media distribution described in Section 19.2.3, which is indicated by step 1 in Figure 19.8. In our example, media is predelivered to one or more servers before the first request arrives; however, it is also possible to perform delivery following the initial request. The preferred mode of operation depends on the prediction or measurement of the content's demand, as well as the relative demand for other content, which may be

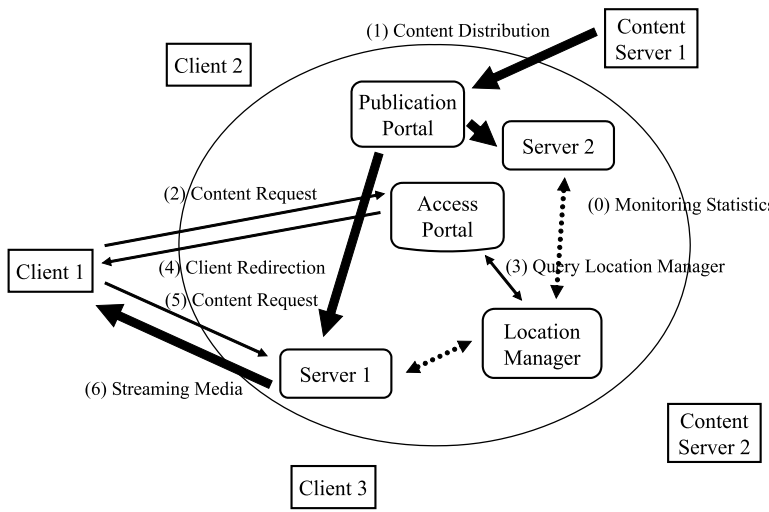


FIGURE 19.8: Various parts of a media overlay, shown in rounded boxes and enclosed, function together to allow access of media content from *Content Server 1* to *Client 1*.

cached in the same servers, etc. One possible implementation of media transport for step 1 is HTTP, which is widely used for data transport.

When a client wishes to retrieve a content, it would contact an *Access Portal* using common mechanisms such as the Real-Time Streaming Protocol (RTSP) or Synchronized Multimedia Integration Language (SMIL). Further details on some common streaming protocols can be found in Chapter 15. In the case of RTSP, step 2 of Figure 19.8 may be an RTSP SETUP message, upon receiving which the *Access Portal* would consult the *Location Manager* in step 3 for the name of an appropriate server. The requesting client can be redirected to the appropriate server through the use of an RTSP REDIRECT message, although it can be more convenient to simply return an error of "moved temporarily" to *Server 1* in response to the SETUP message. This would instruct the client to contact the intended server in step 5 using another RTSP SETUP message, which eventually leads to streaming of media in step 6, possibly using RTP transport.

When using the SMIL-based approach, described in further detail in [46], the request in step 2 may assume the form of an HTTP GET message to retrieve an SMIL file. After consulting with the *Location Manager* in step 3, the *Access Portal* would dynamically rewrite the content of the SMIL to use the intended server. Step 4 then becomes the HTTP transport of the (rewritten) SMIL file, which contains instructions on how to access and compose the media content. In particular, it may contain instruction to access the desired content using RTSP to *Server 1*. An RTSP SETUP message would then be generated in step 5, which results in media streaming in step 6. In both the RTSP and SMIL examples, the client is not aware of the complex operations between the overlay entities and assumes the simple client–server model.

Figure 19.9 shows a possible implementation of messages in steps 2 and 4 of Figure 19.8 when RTSP is used by the client. Note that the *Location Manager* determines that the desired server is *Server 1* and specifies the redirected location for the content in the response to the RTSP SETUP message. Example content of an SMIL file for client redirection is given in Figure 19.10. Note that the address of the desired server, *Server 1*, is specified in the body of the file. As a reminder,

```
SETUP rtsp://Access-Portal:554/content.3gp RTSP/1.0
CSeq: 2
Transport: RTP/AVP;unicast;client_port=8322-8323

RTSP/1.0 302 Moved Temporarily
CSeq: 2
Location: rtsp://Server_1:554/content.3gp
```

FIGURE 19.9: Example content of messages in steps 2 (top) and 4 (bottom) of Figure 19.8 when client makes an RTSP request.

```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
    ...
    <body>
        <audio src="rtsp://Server_1:554/content.3gp/Track1"/ >
        <video src="rtsp://Server_1:554/content.3gp/Track2"/ >
    </body>
</smil>
```

FIGURE 19.10: Example content of an SMIL file for client redirection.

the SMIL file may be rewritten for each client's request based on network and server load.

So far, our media overlay example illustrates how a client can be redirected to a desired server to obtain the *same* piece of content. A media overlay can perform media processing functions as well. For instance, through various client capability exchanges, a streaming server may determine that it is necessary to transcode a content to a reduced frame size for proper display on a client. Clearly, other content transformations are possible, and further discussion about incorporation of media processing functions is given in Section 19.2.6.

19.3 ARCHITECTURE AND DESIGN PRINCIPLES

Media overlays must have a very modular and robust design and architecture to meet the demands of large-scale, streaming media delivery. This section describes the architecture and design principles of an infrastructure-based media overlay network.

19.3.1 Modular Media Overlay Design

A modular design allows a media overlay to be scaled over time in a manner that adapts to user demand and network and system load. For example, as seen in Figure 19.11, an initial deployment of a media overlay network can include a number of overlay servers at a couple of network nodes in different locations in the network. As the number of users increases, additional overlay servers can be added in those locations to satisfy the user demand. As the number of users further increases, the network resource usage may become prohibitive. At this stage, a new overlay server location can be added to improve the network efficiency. A well-architected media overlay with a modular design can allow for this type of scalability over time.

The philosophy of the overlay is that if even just one overlay server is deployed, it should work *independently* to improve the media delivery performance of the system, and if more and more overlay servers are deployed they should work

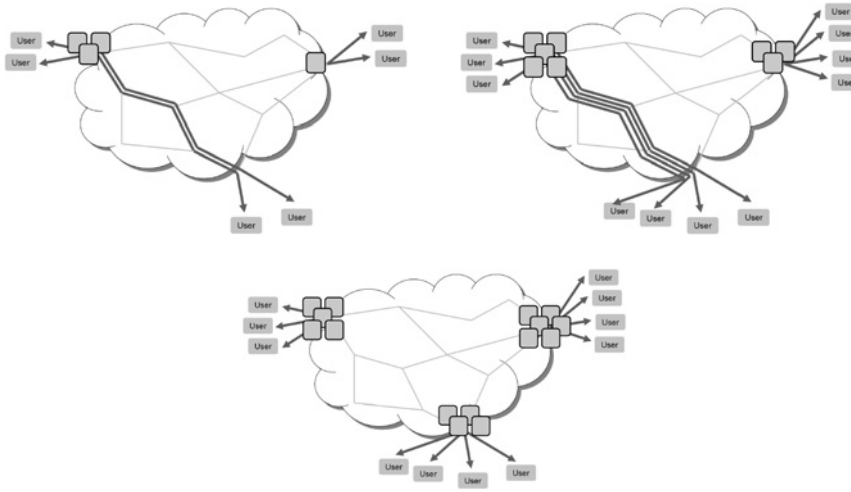


FIGURE 19.11: Media overlays should be designed and architected in a manner that allows them to be scaled to adapt to user demand and system and network load over time. An initial deployment (upper left) can add basic media capabilities to the network. As usage increases, additional resources can be deployed to satisfy the increased user demand (upper right) and improve resource efficiency (bottom).

cooperatively so that their combined performance is greater than the sum performance of the individual components. Also, overlay servers should have *peering* capabilities so that they can be grouped with other overlay servers.

The media overlay architecture should allow for the *incremental deployment* of overlay servers. Since the peering relationships can be controlled, the process of adding a new overlay server can be as simple as putting the new overlay server in place and then setting a peering relationship with an existing overlay server. Likewise, user requests can now be directed to the new overlay server, which can act in isolation as a cache or can be peered with other overlay servers as well.

Furthermore, the system should be *scalable and adaptable* to load in a number of ways. For example, if an overlay server receives many requests, additional overlay servers can be added to increase the cluster size of the existing overlay server. Also, if new areas of the network start experiencing high loads, new overlay servers can be added to those areas. If neighboring areas start seeing lots of correlation between their requests, they can be peered to share content usage statistics and content data between them. If the correlation patterns change, the peering relationships between servers can also be changed. This adaptability makes

the media overlay well suited for campus and enterprise environments and cellular and 802.11 wireless environments.

19.3.2 Media Overlay System Management

Manageability was a key design goal of the media overlay. Media overlay management can be divided into two functions: (1) system monitoring, measurement, and analysis (through queries) and (2) system control (through commands). Both these functions can be performed between components through a management/control interface. The control interface allows the system to accept and give requests and commands. Since each overlay server tracks its own statistics, it can respond to queries for content usage, server load, and network conditions. Also, overlay servers can respond to commands for moving content, beginning and ending streaming sessions, and processing streams. These commands and requests can be from other overlay servers or managers. It is this modular design that allows the media overlay to be configured in many different modes of operation.

A number of specific notes can be made about media overlay system management. First, since overlay servers are constantly monitoring and logging statistics, they can be configured to periodically report their statistics to a specified entity or to reply to queries for these statistics received through the management/control interface. Next, it should be noted that overlay servers can be turned on or off for administrative needs such as system maintenance. Also, overlay servers can be added to the media overlay to facilitate incremental deployment, and they can be moved between nodes to adapt to evolving request patterns and system and network load patterns. In addition, as overlay servers get loaded with streaming sessions or as overlay network links become congested, it may become necessary to change servers during midsession. This can be done with the overlay server's streaming handoff capabilities. Thus, management can influence active streaming sessions. Finally, managers can change the peering relationships of overlay servers through the management interface. Peered overlay servers can be in the same overlay node or in different overlay nodes, and different overlay servers within a single overlay node do not necessarily have to be peered with each other.

19.3.3 Media Overlay Design Choices

The flexibility and modularity of the media overlay architecture allow it to be used in a number of modes of operation and customized for a number of deployment scenarios. We discuss a few of these modes and scenarios.

The modular components and interfaces of the media overlay allow it to work in a *centralized or distributed* mode of operation (Figure 19.12). In centralized mode, a central manager can collect statistics from all the overlay servers, analyze these statistics, determine the best strategy for delivering media streams, and

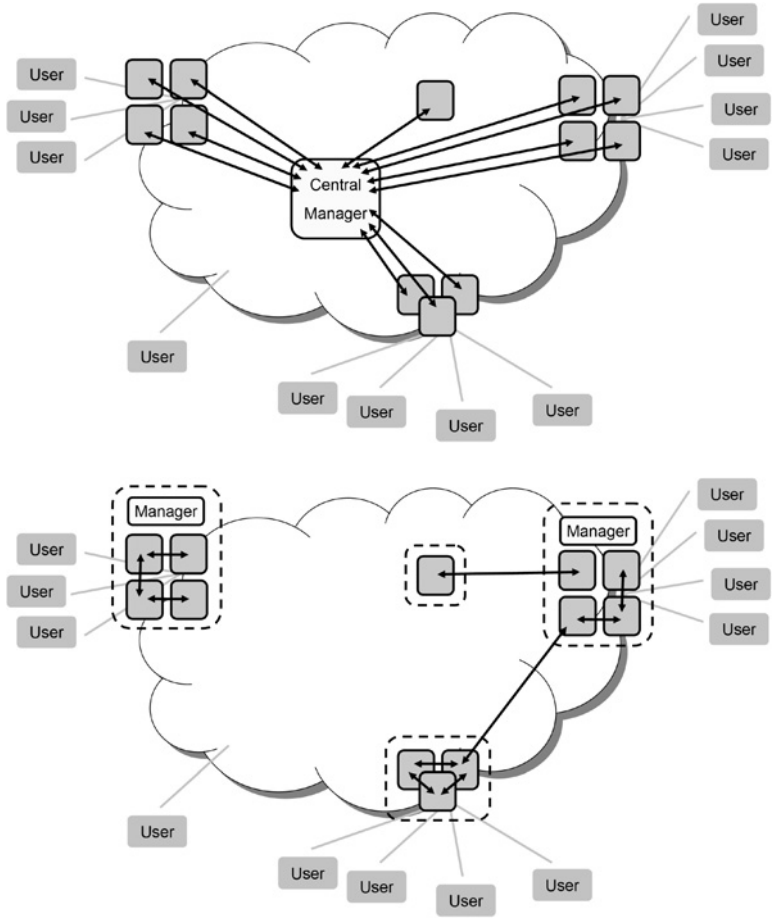


FIGURE 19.12: Centralized vs. distributed management: a media overlay can be managed centrally (top) where all overlay servers report to and are controlled by a central manager. A media overlay can also be operated in a more distributed fashion (bottom) where overlay servers are peered to share statistics and make more rapid, local decisions.

send commands to the overlay servers to carry out this strategy. In distributed mode, each overlay server can analyze its own statistics and perhaps collect and analyze statistics from neighboring overlay servers, and then make decisions on how best to serve mobile streaming requests. The centralized operation has the advantage of having a global view of the statistics, but may be better suited to longer

timescale adjustments, whereas the distributed operation may allow quicker data collection and analysis and allow quicker reactions to rapidly changing user patterns and system and network load patterns. Of course, hybrid combinations may be advantageous for a number of scenarios.

A media overlay network can have a *single owner* responsible for its operation. In this case, system management becomes much simpler because the owner can track the deployment of each overlay server and monitor the server and network load. Alternatively, a media overlay can have *multiple owners* who must agree to how their overlay resources should cooperate to deliver the media streams between end points. Finally, a media overlay can have *no defined owner*, in which case each of the overlay servers must be able to operate independently and cooperatively using standard compliant protocols for its interfaces.

The system can act in *push* or *pull* mode. In pull mode, content distribution can be triggered by user requests, for example, by caching the content with the highest number of requests. In push mode, the content can be distributed based on an analysis of user requests and network and system load performed by an overlay's management capabilities. In other words, content can be prefetched by the various overlay servers. Also, the push to overlay servers can be explicitly configured, for example, by an operator or content owner to ensure high-quality access to specific content.

19.4 ADVANCED TOPICS

A variety of more advanced capabilities can also be incorporated in a media overlay to improve the media delivery performance for end users or to improve the network utilization. This section briefly highlights some of the advanced capabilities that have been developed in recent years.

19.4.1 Network-Adaptive Media Streaming

Once media distribution and client request redirection are performed, the streaming session itself can begin. Streaming involves the delivery of long, continuous media streams and desires highly predictable bandwidths, low delay, and preferably no losses. In particular, midstream disruption of a streaming session can be highly distracting. There are a variety of important opportunities in overlay nodes for performing adaptive streaming for improving system performance, for example, see [37].

Stream scheduling: A number of opportunities lie in the general area of stream scheduling, where the basic idea is scheduling the packet transmissions for a media stream over a channel that may exhibit time-varying available bandwidth, loss rate, and delay. These scheduling problems have a number of flavors, including scheduling delivery of a single stream over a rate-constrained, time-varying

channel, shared scheduling of many streams across a shared channel, and shared scheduling of many streams of a single server itself. While basic streaming systems simply transmit media packets in consecutive order without regard to the importance of individual packets, significant benefits can be obtained by exploiting the natural priority of each packet, for example, I, P, or B frames or scalable layer [1]. Further benefits result from rate-distortion optimized packet scheduling, which decides which packet should be transmitted at each transmission opportunity, as a function of each packet's importance and coding dependencies, estimated channel conditions, and feedback from the client on prior received/lost packets (see [10,15] and Chapters 4, 10, and 14). In addition, low-complexity stream scheduling algorithms can exploit periodic coding structures in the encoded video [45].

Wireless streaming: Wireless channels are a shared, highly dynamic medium, leading to unpredictable, time-varying available bandwidth, delay, and loss rates [13]. A key opportunity lies in optimizing wireless streaming algorithms from the overlay nodes to the mobile client. The streaming algorithm must adapt to time-varying network conditions and must be resilient to packet loss over error-prone wireless channels, as described in Chapters 11–13.

When an overlay server is colocated with a wireless base station and has accurate and timely information about the channel conditions, it can more readily adapt the streaming to the wireless channel variations. If an overlay server is not serving a stream but merely relaying content to a wireless client, there are still several beneficial functions that it can perform. First, it provides an additional observation point for packet reception statistics. When combined with reception statistics observed at the client, this allows the determination of whether packet losses are due to congestion or wireless link corruption. Such information is important for congestion control purposes. Second, in wireless wide-area networks, wireless link delay is typically very high, making adaptation inefficient. The overlay node could provide feedback on a much shorter timescale to make media adaptation more effective [9].

Adaptive streaming for multiple clients: Streaming applications may need to stream media to clients with different network bandwidths. These clients require different bit rate versions of the same content. This can be supported by using scalable coding and sending different layers on different multicast trees—each receiver joins the appropriate multicast tree(s) based on the desired content [22]. Similarly, multiple multicast trees can provide different amounts of forward error correction (FEC) for error control, where each client selects the desired amount of FEC [38].

19.4.2 Real-Time Media Adaptation and Transcoding

A streaming system must be able to deliver media streams to a diverse range of clients over heterogeneous, time-varying networks. In many scenarios, the down-

stream network conditions and client capabilities are not known in advance, and the network conditions may be time varying based on cross traffic. To overcome these obstacles requires dynamically matching the streaming media to the available bandwidth and capabilities of the specific client device. A number of approaches can be used to solve this problem. Multiple file switching switches between media files coded at different data rates [12]. Scalable coding stores base and enhancement streams that can be sent in a prioritized fashion [28]. Transcoding adapts precompressed streams into formats better suited for downstream conditions. These methods provide different trade-offs in terms of flexibility, compression efficiency, and complexity [6,19,36,39,43,44].

Mid-network stream adaptation: Media adaptation or transcoding may be performed at the sender or at a mid-network node. For example, for pre-encoded content it is customary to adapt the pre-encoded content at the sender for the current delivery situation. However, it is often valuable to transcode at a mid-network node—a node in the middle of the network between the sender and the receiver. A practical reason is that it is unlikely that every content source will have transcoding capability, and a transcoding-capable overlay is arguably the simplest solution. There are technical advantages in other situations as well. Another example is multicast video, where a single input stream is adapted to create multiple streams of different bit rates. Note that mid-network node transcoding is important for both pre-encoded and live content. Therefore, mid-network transcoding is a generally useful capability for overlay nodes to transcode streams according to downstream network conditions, such as lower bandwidth channels, congested network nodes, and time varying wireless channels. Since an overlay node may have to transcode many streams at once, both quality and computational efficiency are of importance.

19.4.3 Media Security

There are a number of important issues that relate to media security in the context of streaming media infrastructure. These include security issues related to the media itself, such as providing confidentiality of the media content or limiting access to the content to only those with appropriate access rights. Digital rights management becomes more involved since centralized solutions at the origin server must be extended to the distributed infrastructure where caching and streaming may occur at overlay nodes. For example, the policies for certain content may specify that it should not be cached. There is a need to control malicious attacks, such as Denial of Service attacks. Many of the aforementioned security issues are not specific to media and are not discussed further here due to limited space. One security problem that does directly relate to media processing is the question of how to transcode encrypted content.

End-to-end security and mid-network secure transcoding: A practically important problem involves how to provide end-to-end security for a streaming session, while also supporting mid-network transcoding. End-to-end security corresponds to encrypting the content at the sender, decrypting at the receiver, and having the content in encrypted form everywhere in between. Mid-network transcoding is therefore challenging because the content is encrypted. The conventional approach is to give the transcoder the key so that it can decrypt/transcode/re-encrypt the content, but this breaks the end-to-end security. To be more specific, giving the transcoder the key raises the key distribution problem, and the transcoder may also be untrustworthy. While end-to-end security and mid-network transcoding appear to be mutually exclusive, a careful codesign of the compression, encryption, and packetization can enable mid-network transcoding while preserving end-to-end security—referred to as *secure transcoding* to emphasize that the transcoding is performed without compromising end-to-end security [3,42].

19.4.4 Path and Server Diversities

Routing around failures: The ability of an overlay node to relay traffic provides flexibility in selecting the network path. Path selection, whereby a path of best or good quality is chosen among a set of candidate paths, is an obvious way to improve streaming quality. In particular, selection allows routing around failed links, which may otherwise render parts of a network inaccessible.

Robust streaming using distributed infrastructure and diversity: The distributed infrastructure of the overlay network also provides an opportunity to explicitly achieve *path diversity* and *server diversity* between each client and multiple nearby overlay servers. For example, multiple servers can send different streams over different paths (partially shared and partially not) to each client, thereby providing various forms of diversity that can overcome congestion or outage along a single path and improved fault tolerance. This may be achieved using multiple description (MD) coding as an MD-CDN [4] (using various MD codecs, e.g. [2, 29,41]) or single description or scalable coding with FEC [7,21,25]. It can also be achieved by using multiple wireless base stations or 802.11 access points [24]. A more detailed discussion of the benefits and use of path diversity is given in Chapter 17.

19.4.5 Mid-Session Streaming Handoff

Handoff of streaming sessions: Streaming media delivery differs from webpage delivery in that streaming sessions are often long lived. The long-lived nature of streaming sessions, combined with user mobility, raises the possible need of midstream handoffs of streaming sessions between overlay servers. This handoff should be transparent to the receiving client. Handoff of video sessions between

overlay servers is challenging since the receiving client is highly sensitive to any interrupts. Furthermore, when the streaming session involves transcoding, mid-session handoff of the transcoding session may also be required between overlay nodes [33].

Dynamic load balancing of long-lived streaming sessions: The midstream handoff capability is also useful for enabling improved dynamic load balancing and fault tolerance. As more streaming or transcoding sessions are started, it may be useful to rebalance the streaming or transcoding sessions. For example, the overlay nodes can be used to explicitly route streams by using application-level forwarding where the overlay servers act as relays. By combining this with midstream handoff capability, streams can be dynamically rerouted to alleviate network congestion and improve load balancing.

19.5 MEDIA OVERLAY USES AND BENEFITS

In this section, we discuss some example applications enabled by a media overlay infrastructure. The advantages of media overlay can be examined from several perspectives: (1) improvement for the end user, (2) improvement for system scalability and performance, and (3) new capabilities. For each of the examples discussed in the remainder of this section we highlight important advantages along these three perspectives.

19.5.1 Media Delivery

The most basic function of a network is the delivery of data. In this section, we discuss how media delivery can be improved with a media overlay.

Consider an end user who is employing streaming media for entertainment and communication. The user's primary concerns (cost not withstanding) are ease of content access, smooth media playback, and acceptable latency for interactive communication. We next examine how a media overlay can help achieve these objectives.

Similar to how CDNs reduce access latency for web access, a media overlay improves latency when accessing media by physically locating the content closer to the user. By using multiple distributed servers, and selecting the "closest" server for each requesting client, improved responsiveness can be achieved as compared to using a single, likely distant server. The shorter distance from server to user also reduces the likelihood of encountering a network bottleneck that causes packet losses and throughput degradation, thereby improving user streaming experience.

Media overlays provide unique features for streaming media that have no correspondence in CDNs for web access. These are needed to accommodate the con-

tinuous requirement to have streaming media delivered on time to end users. One important feature that provides many advantages is media multicast or splitting; this is discussed further in Section 19.5.2.

In addition, the overlay can provide the feature of adapting media content on a real-time basis. One example of such adaptation is prioritized dropping, where packets that are more disposable are preferentially discarded when needed. Such dropping of data is best performed inside the infrastructure for a number of reasons. First, while a transmitting source can best prioritize its own traffic, it cannot optimize across the different media flows that are sharing the resource bottleneck. Second, real-time information for the resource bottleneck may be difficult to obtain due to large physical separation or impossible to obtain due to the presence of administrative boundaries.

Another example of media adaptation by a media overlay is transcoding a media stream to a lower bit rate in response to, for example, throughput degradation. Traditional approaches to address throughput variation in networks include adaptive live encoding or rate transcoding at the transmitting source for live content and switching between multiple copies of the same content at different bit rates for stored content. A media overlay is superior for the former by being able to perform adaptation closer to the bottleneck. For stored content, it is often economical to produce multiple copies only for content that is popular, and even then, only very limited bit rate options are available. Live transcoding inside a media overlay provides the option of a customized stream whenever necessary.

One key benefit of a media overlay architecture is the possibility of having path diversity, which is the subject of Chapter 17. Path diversity can be exploited in several ways, including selection and aggregation. Under selection, a good or best path is selected for a media stream, while multiple paths are used for a single stream under aggregation. The advantages afforded by path diversity include improved reliability. This is achieved by the ability to bypass network failures via alternative paths and by the potential to exploit underutilized paths to improve throughput stability.

From a system perspective, a media overlay enables a number of advantages, which we shall discuss shortly. Similar to CDN for web traffic, by locating content close to the user, a media overlay is efficient in that the network resources needed to achieve a task are typically reduced compared to using a single distant server. This is achieved by the caching functions of a media overlay. In addition, the possibility of using multiple paths and servers allows a more efficient use of resources via balancing of network and server loads. One benchmark for an efficient delivery infrastructure is that every packet traverses every link at most once. The combination of caching and routing functions of a media overlay provides the key components for realizing this ideal.

19.5.2 Multicast or "Splitting" Service

A well-known overlay application is point-to-multipoint communication or multicast, under which overlay architectures are used to form distribution trees for relaying a single content to multiple recipients [11,17]. These overlays effectively emulate a multicast service in an otherwise point-to-point network. Since an overlay node may receive one copy of a packet and relay it to multiple recipients, the operation is sometimes called "splitting." This is achieved by the flexible delivery path afforded by a media overlay discussed in Section 19.4.4.

Traditionally in a point-to-point network, the only way to achieve multicast communications is to employ multiple independent transmissions of the same data. Clearly, such repetition can be wasteful for large group sizes and poses heavy loads for servers and networks alike. Nevertheless, repetition allows the preservation of the simple client–server model of Figure 19.2a, which is of practical importance for compatibility with capability-constrained devices or older devices. In the other extreme, it is possible to establish relaying infrastructure using end hosts only, as is done in many peer-to-peer systems. For example, in Figure 19.2c, it is possible for "Server" to stream to client P, which is then responsible for relaying data to P'. When each client relays data to only one other client, we have a distribution chain, but more general distribution structures are clearly possible. The use of end-host systems can surely relieve the server load and, to a certain extent, relieve network load as well, depending on the particular distribution structure employed. Nevertheless, these all come at the expense of higher complexity and resource requirements for the clients. For example, the simple client–server model is no longer valid, and a client has to maintain communications with multiple end hosts and change these connections based on end-host churning. Furthermore, end hosts now need to perform data transmission in addition to reception.

By strategically placing the "splitting" function in the overlay nodes, the overlay multicast solution allows clients to assume the simple client–server model, while relieving the high server and network loads associated with using multiple independent transmissions. End-user video quality is improved with reduced server and network load, and the possibility of having "local" retransmissions between overlay nodes. For the operator, the reduction of network traffic associated with multicast greatly reduces network cost, and the simple client–server model guarantees support for a wide range of clients, as well as simpler problem diagnosis.

Many multicast communications with large client population are scheduled in advance. Examples include major sports events for entertainment and CEO announcements for corporate communications in a large enterprise. In Figure 19.4, overlay resources are represented as overlay nodes inside the infrastructure and edge servers near the clients. One possible approach to affect multicast is then to use the *push* mode of operation discussed in Section 19.3.3. Under push mode, the

overlay nodes and edge nodes form a distribution tree, with edge nodes assuming the additional duty of streaming to clients. A client can be redirected to an appropriate edge node via one of several mechanisms discussed in Section 19.4.

Application-level multicast or "splitting" architectures are particularly relevant for video content due to the high volume of data involved. However, many technical issues remain. For example, in the dense client situation (when the expected number of clients per nearby edge server is large) it is sensible to push the content to an edge server. This design provides two advantages. First, the edge server serves as a rendezvous point for clients to access the content. Second, a relatively static distribution structure can be used to connect the edge servers, each of which individually handles joining and leaving of clients. When clients are sparsely distributed, two corresponding problems arise. First, it may no longer be efficient to involve an edge server, and an entry point into the overlay system is required, for example, via a portal. Second, the distribution structure may need to evolve as clients join and leave, which is typically challenging due to the relatively unpredictable behavior of individual clients.

19.5.3 Multi-Way Conferencing

A media overlay allows new capabilities to be introduced incrementally to an existing network without modifying existing clients or requiring global infrastructure upgrades. For example, consider a network that supports point-to-point video communication. In this example, client devices have a video conferencing application that can receive, decode, and play back one audio/video stream and that can capture, encode, and send one audio/video stream. Normally, clients can communicate directly with one another to establish a two-way conference, as shown in the upper left of Figure 19.13. Three-way or multi-way video conferencing can be achieved, as shown in the upper right of Figure 19.13, where all the streams are sent to all the clients, but this approach requires upgrading all the clients and requires each client to have the ability to receive and process multiple streams. However, the question that we now consider is how to support three-way or, more generally, multi-way video conferencing using existing applications designed for two-way conferencing? We show how the media overlay can enable this improved functionality without requiring any changes in the client, as shown in the lower right of Figure 19.13.

Since the client is able to send and receive a single audio/video stream, the key capability that the media overlay must provide is the ability to combine the audio/video streams from all the other clients in the video conferencing session into a single audio/video stream that can be decoded by the client. This single audio/video stream should contain the appropriate view from one or more of the various session participants. This can be achieved in a number of ways. First, the media overlay can turn the remote participants' streams into a single stream by

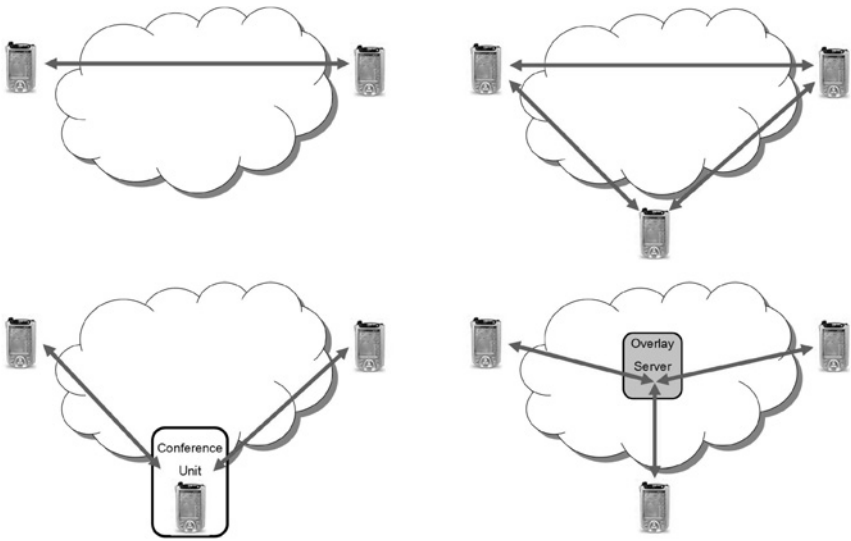


FIGURE 19.13: Conferencing. (Upper left) Two-way conferencing. (Upper right) Three-way conferencing can be accomplished by sending all streams to all other users, but this requires each participant to be able to receive all the clients to be upgraded and capable of receiving and playing multiple streams. (Lower left) Multi-way conferencing can also be achieved by upgrading only one client with conferencing capabilities and having all the clients send and receive to this client. (Lower right) Multi-way conferencing can be achieved transparently to all clients with the overlay approach. All the clients interact with an overlay server as if it were a single client. The overlay then combines the streams into the appropriate single stream for each client. The overlay can be upgraded over time to provide improved conferencing capabilities in a manner that is transparent to clients.

performing a video transcoding operation that downscales all the video streams and stitches or tiles them together into a single video stream and by performing an audio processing operation that combines all the audio streams into a single audio stream. Another option for combining multiple downscaled video streams is selecting the video of the active speaker. The media overlay can achieve this by applying an activity detection algorithm to the audio and/or video streams and selecting the video of the active participant. The media overlay can perform other options as well, such as combining the first two approaches where all the video

streams are transcoded, but done in a manner so that the active remote participant is displayed with a larger display size than the nonactive participants. In all these examples, since the media overlay performs the multistream transcoding in a manner that produces a single audio/video stream that is accepted by the two-way video conferencing client application, multiuser conferencing is a new capability that is seamlessly provided by the media overlay.

To further understand the advantages provided by an overlay approach as compared to end-host approaches, the next few paragraphs further examine the tradeoffs that arise with different end-host approaches to multi-way conferencing. If the video conferencing client application could be upgraded, then one way to achieve multi-party conferencing is for each client to send to and receive from all other participants as shown in the upper left of Figure 19.13. Analogous client-side operations to achieve the features described earlier, such as video downscaling and audio mixing, are then performed locally. This is most flexible, but also requires the largest amount of network and system resources both in the infrastructure and at the clients. Therefore, this approach is not scalable to large conference sizes. In addition, this solution presumes appropriate network support: it may not be possible to get a new multi-party conferencing phone to work with the existing telephone network.

For tiling display of multiple participants, the scheme presented earlier can be improved by allowing participants to downscale their video streams to an appropriate size before sending them to the other participants. This requires coordination and signaling between the various clients to determine the size that each participant's video must be reduced to. For display of active speaker only, the scheme presented earlier can be improved by locally performing activity detection. Coordination and signaling for selecting an active sender are also required.

To allow Alice, Bob, and Carol to participate in three-way conferencing, it is possible for Alice alone to have an updated application, while Bob and Carol use their legacy application to call Alice. This is shown in the lower left of Figure 19.13. Alice then plays the role of the central aggregator and performs all media processing for the entire conference. Compared to the overlay solution, this end-host only solution has four limitations. First, it is not applicable to networks such as telephone networks that only allow a single point-to-point communication. Second, at least one participant must have an upgraded application that is known to all participants. Third, the limit on maximum number of participants is likely to be much lower for an end host than an overlay infrastructure. Finally, Alice, the central aggregator, cannot leave the conference without ending the conference.

For end users, overlay conferencing allows existing applications to be used, and simple logic for joining and leaving a conference that is likely to translate into fewer disturbances for other participants. It can also support greater customization, where each client decides, for example, on the desired viewing size for each

stream. From a systems perspective, the overlay solution is attractive due to the lower amount of traffic corresponding to a central aggregation point.

19.5.4 Additional Overlay-Based Media Features

Recording and retrieval of multimedia messages: In addition to data delivery, the recording and retrieval of voice and video messages have evolved to become common features in many networks. An overlay infrastructure of Figure 19.4 is well equipped for this task because of its storage and streaming capabilities. The recording and retrieval of multimedia messages can be implemented in the client if it is permanently attached to a network, as is the case in land-line telephones. Nevertheless, the overlay implementation offers several advantages. For the user, it provides higher service availability, as the client no longer needs to be always turned on and attached to a network. The user also benefits from easier access to the content from other devices. From a systems perspective, the economy of scale allows such recording and retrieval functions to be more economically implemented in the infrastructure than in the end clients.

Enhanced media access: With new media formats being created, and the capability of network and client devices being improved continuously, the range of available media content becomes highly diversified in terms of codec types and options, bit rates, and picture sizes. This clearly poses a nightmare for interoperability. While updating applications on clients to support new codecs is arguably possible, although administratively difficult, a high bit rate or high complexity content may always be incompatible with a constrained client. A media overlay is in an ideal position to bridge the gap. Since a media overlay is involved in the delivery of the media content, it can exploit its available computation resources to transcode the media content to a format that a client is capable of handling, where format subsumes relevant parameters such as codec types and options, bit rate, and frame size. For the user, the benefit is access to content that would otherwise be unaccessible. While there may be slight quality degradations associated with transcoding, for a large class of content whose purpose is communication rather than visual entertainment, such a trade-off is well justified. From a systems perspective, the overlay solution provides a single point of upgrading for new capabilities, such as a new codec. In addition, it is impossible to predict and impractical to encode, at content creation time, all possible configurations required by different clients. The overlay solution effectively provides a way of dynamically creating a custom stream only when necessary. The possibility of caching transcoded content makes the solution more competitive in terms of computation requirements [35].

Content sharing: In an example given earlier, the overlay infrastructure stores the multimedia message boxes for users. Generally, the infrastructure can store other content for the user and allow playback not only to a device, but to a con-

ference of participants as well. This allows an enhanced version of the multiway conferencing discussed earlier, where the display options are expanded from scaled and tiled video of participants, and the active speaker, to include a document under discussion or vacation video being shared. Again, the overlay solution does not require any clients to be modified. An equivalent system implemented in the end host would require all clients wishing to share content to be upgraded, along with other drawbacks outlined in Section 19.5.2. While the necessary media processing may be held in an overlay server, the content to be shared may reside in another server inside or outside the overlay infrastructure. Therefore, this application cannot be realized without the media processing and content distribution functions of the overlay infrastructure.

Content adaptation for improved usability: Similar to transcoding of media discussed earlier, there are a number of other media processing that could be incorporated in a media overlay infrastructure without requiring client changes. We will discuss here two examples for improving usability of media content. The first example is "camera stabilization." Many mobile devices are equipped with cameras, but shooting usable video requires a trained and stable hand. When incorporation of image stabilization in the end devices is not computationally practical, the media overlay provides a natural place for such algorithms, especially when the content is stored in the overlay. The second example relates to the rendering of speech as text overlayed onto the video. This is important for viewing content in public places, such as libraries, or in noisy places, such as trains and restaurants. In noisy environments, visual text may be more comprehensible than listening to the original audio through an earphone. While the overlaying of text onto video is not a difficult task and may be performed at the client, speech recognition may prove too complex for most client devices. Again, the media overlay infrastructure is conveniently involved in the delivery of content and can perform the aforementioned processing to render an appropriate stream to be delivered to the client.

Additional possible services: There are many other potentially important media services, including video or audio processing operations, such as speed up/slow down of playback, VCR functionalities, logo insertion, background removal, enhancing resolution, and deblurring and noise reduction. In addition, and very importantly, an overlay infrastructure provides a convenient and highly flexible platform for introducing new services and gradually expanding the services based on the number of users or improved functionalities, and it provides this capability while generally enabling backward compatibility with older client devices.

19.6 SUMMARY AND FURTHER READING

This chapter introduces the basic concepts, capabilities, and operations of a media overlay. The media overlay is an extension of an existing network infrastruc-

ture and improves upon the basic network by having strategically located overlay nodes with processing, storage, and relaying capabilities. This is in contrast to the traditional client–server approach, where all intelligence and processing reside in the two end points, and the peer-to-peer model, where the intelligence and processing are distributed across multiple end points without being integrated with the basic network.

The benefits of the infrastructure approach assumed by a media overlay can be examined in many different respects, including (1) improved availability, latency, and quality of service for end users, (2) improved operational efficiency and manageability for network operators, and (3) ease of adding new services and capabilities. All of these benefits are achieved with minimal changes to existing clients and network infrastructure and allow incremental introduction of new features and scaling of existing features to a larger audience.

Many capabilities are central to the operations of a media overlay. For media delivery, these include media distribution, caching, multicast, media serving, and security. For maintenance and management, these include resource monitoring, management, and handoffs functions. For media services, these include operations that adapt the content of the media being delivered. For example, transcoding can be used to adapt a media stream's display resolution for a particular client device or bit rate for a congested network.

Since the overlay is a part of a larger infrastructure, design and architectural considerations are important. Some important design considerations for media overlay design discussed in this chapter include modular design with well-defined interfaces for data and control, push and pull delivery of media, and centralized and distributed modes of operation for the media overlay. Some architectural considerations discussed in this chapter include the system and network monitoring capabilities of the overlay and the manageability of the overlay, which includes allowing incremental deployment and upgradeability over time.

By properly designing a media overlay, an existing network infrastructure can be evolved to handle multimedia applications. This chapter discusses a number of overlay-enabled applications, including media delivery, overlay multicast, multiway conferencing, and other media services, including multimedia messaging, enhanced media access, content sharing, and content adaptation.

Evolution of networked media services: Many networked media services such as multimedia messaging, streaming media, video conferencing, mobile television, multiplayer gaming, and video blogging and podcasting are beginning to emerge. We foresee widespread adoption in the coming decade, and we believe that two key ingredients are giving users compelling media services and high-quality media experiences. This requires a flexible infrastructure that allows new services to be incrementally deployed, since it is difficult to predict user response to new services, and allows its media delivery capabilities to be upgraded in an evolutionary manner, since there are costs associated with infrastructure improve-

ments and network owners are hesitant to upgrade their infrastructure until they see signs of high adoption and associated revenue.

Media overlays are a likely path to the widespread adoption of networked multimedia services because they provide a highly flexible platform for introducing and trying new services and are a means for gradually expanding the service deployment as the number of users increases. It also allows the infrastructure's media capabilities to be upgraded in an evolutionary manner. Furthermore, over time we believe that many of these media overlay capabilities will eventually be built into the base network itself, but on the path to widespread deployment, media overlays will provide the vehicle through which these capabilities and new services will first be developed and deployed.

Comparisons of infrastructure-based overlays to other approaches are given in [11,17,20,31]. In [20], the design issues and choices of an infrastructure-based overlay are discussed. In [17], the trade-offs among several approaches for video-on-demand applications are evaluated. A readable account of when to adopt peer-to-peer systems is given in [31], while [11] presents an approach in which an overlay is formed by clients rather than by infrastructure nodes. Readers interested in problems relating to server placement or content placement may consult [27], while [4] discusses the use of media overlays for streaming of video coded in a multiple-description fashion. A detailed example of SMIL-based redirection, as well as the use of segmented video, is discussed in [46]. Further discussion on how media processing services, such as transcoding, may be implemented in a media overlay is given in [16]. An example of possible media adaptation inside an overlay infrastructure to improve transport over wireless networks is given in [9]. Further readings on several topics can be found in other chapters in this book. For example, bandwidth adaptation techniques are discussed in Chapters 4 and 10, streaming media on demand in Chapter 14, and media streaming protocols in Chapter 15.

REFERENCES

[1] A. Albanese, J. Blomer, J. Edmonds, M. Luby, and M. Sudan. "Priority Encoding Transmission," in *Proceedings of IEEE Symposium on Foundations of Computer Science*, pages 604–612, November 1994.

[2] J. Apostolopoulos. "Reliable Video Communication over Lossy Packet Networks Using Multiple State Encoding and Path Diversity," *Proceedings of Visual Communications and Image Processing (VCIP)*, January 2001.

[3] J. Apostolopoulos. "Secure Media Streaming and Secure Adaptation for Non-scalable Video," *Proceedings of IEEE International Conference on Image Processing*, October 2004.

[4] J. Apostolopoulos, T. Wong, W. Tan, and S. Wee. "On Multiple Description Streaming with Content Delivery Networks," *Proceedings of IEEE INFOCOM*, June 2002.

[5] A. Barbir, B. Cain, F. Douglis, M. Green, M. Hofmann, R. Nair, D. Potter, and O. Spatscheck. "Known Content Network (CN) Request-Routing Mechanisms," *Internet Engineering Task Force*, RFC 3568, July 2003.

[6] N. Bjork and C. Christopoulos. "Transcoder Architectures for Video Coding," in *Proceeedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, Seattle, WA, May 1998.

[7] J. Chakareski and B. Girod. "Rate-Distortion Optimized Packet Scheduling and Routing for Media Streaming with Path Diversity," *Proceedings of IEEE Data Compression Conference*, April 2003.

[8] S. Chen, B. Shen, S. Wee, and X. Zhang. "Segment-Based Streaming Media Proxy: Modeling and Optimization," *IEEE Transactions on Multimedia*, 8(2):243–256, April 2006.

[9] G. Cheung, W. Tan, and T. Yoshimura. "Double Feedback Streaming Agent for Real-Time Delivery of Media over 3G Wireless Networks," *IEEE Transactions on Multimedia*, 6(2):304–314, April 2004.

[10] P. Chou and Z. Miao. "Rate-Distortion Optimized Streaming of Packetized Media," *IEEE Transactions on Multimedia*, 8(2):390–404, April 2006.

[11] Y. Chu, S. Rao, S. Seshan, and H. Zhang. "A Case for End System Multicast," *IEEE Journal on Selected Areas Communications*, 20(8):1456–1471, October 2002.

[12] G. Conklin, G. Greenbaum, K. Lillevold, A. Lippman, and Y. Reznik. "Video Coding for Streaming Media Delivery on the Internet," *IEEE Transactions on Circuits and Systems for Video Technology*, March 2001.

[13] S. Dawkins, M. Kojo, V. Magret, and N. Vaidya. "Long Thin Networks," *Internet Engineering Task Force*, RFC 2757, January 2000.

[14] S. Floyd and K. Fall. "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Transactions on Networking*, 7(4):458–472, August 1999.

[15] B. Girod, J. Chakareski, M. Kalman, Y. Liang, E. Setton, and R. Zhang. "Advances in Network-Adaptive Video Streaming," *Tyrrhenian International Workshop on Digital Communications*, September 2002.

[16] M. Harville, M. Covell, and S. Wee. "An Architecture for Componentized, Network-Based Media Services," in *Proceedings of IEEE International Conference on Multimedia and Expo*, Baltimore, Maryland, July 2003.

[17] K. Hua, M. Tantaoui, and W. Tavanapong. "Video Delivery Technologies for Large-Scale Deployment of Multimedia Applications," *Proceedings of IEEE*, September 2004.

[18] R. Katz and E. Brewer. "The Case for Wireless Overlay Networks," in Tomasz Imielinski and Henry F. Korth, editors, *Mobile Computing*, pages 621–650. Kluwer Academic Publishers, 1996.

[19] G. Keesman, R. Hellinghuizen, F. Hoeksema, and G. Heideman. "Transcoding MPEG Bitstreams," *Signal Processing: Image Communication*, 8(6), September 1996.

[20] L. Kontothanassis, R. Sitaraman, J. Wein, D. Hong, R. Kleinberg, B. Mancuso, D. Shaw, and D. Stodolsky. "A Transport Layer for Live Streaming in a Content Delivery Network," *Proceedings of IEEE*, September 2004.

[21] A. Majumdar, R. Puri, and K. Ramchandran. "Distributed Multimedia Transmission from Multiple Servers," *Proceedings of IEEE International Conference on Image Processing*, September 2002.

[22] S. McCanne, V. Jacobsen, and M. Vetterli. "Receiver-Driven Layered Multicast," *ACM SIGCOMM*, August 1996.

[23] M. Miao and A. Ortega. "Scalable Proxy Caching of Video under Storage Constraints," *IEEE Journal on Selected Areas Communications*, September 2002.

[24] A. Miu, J. Apostolopoulos, W. Tan, and M. Trott. "Low-Latency Wireless Video over 802.11 Networks Using Path Diversity," *Proceedings of IEEE International Conference on Multimedia and Expo*, July 2003.

[25] T. Nguyen and A. Zakhor. "Distributed Video Streaming over Internet," *SPIE Multimedia Computing and Networking 2002*, January 2002.

[26] C. Patrikakis, Y. Despotopoulos, A. Rompotis, N. Minogiannis, A. Lambiris, and A. Salis. "An Implementation of an Overlay Network Architecture Scheme for Streaming Media Distribution," in *29th Euromicro Conference (EUROMICRO'03)*, September 2003.

[27] L. Qiu, V. Padmanabhan, and G. Voelker. "On the Placement of Web Server Replicas," *Proceedings of IEEE INFOCOM*, April 2001.

[28] H. Radha, M. van der Schaar, and Y. Chen. "The MPEG-4 Fine-Grained Scalable Video Coding Method for Multimedia Streaming over IP," *IEEE Trans. on Multimedia*, March 2001.

[29] A. Reibman, H. Jafarkhani, Y. Wang, M. Orchard, and R. Puri. "Multiple Description Video Coding Using Motion-Compensated Temporal Prediction," *IEEE Trans. Circuits and Systems for Video Technology*, March 2002.

[30] A. Rousskow and D. Wessels. "Cache Digests," *Computer Networks and ISDN Systems*, 30(22–23):2155–2168, November 1998.

[31] M. Roussopoulos, M. Baker, D. Rosenthal, T. Giuli, P. Maniatis, and J. Mogul. "2 P2P or Not 2 P2P," *Proceedings of 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2004.

[32] S. Roy, M. Covell, J. Ankcorn, and S. Wee. "A System Architecture for Managing Mobile Streaming Media Services," in *Proceedings of IEEE International Workshop on Mobile Distributed Computing*, May 2003.

[33] S. Roy, B. Shen, V. Sundaram, and R. Kumar. "Application Level Handoff Support for Mobile Media Transcoding Sessions," *ACM NOSSDAV*, May 2002.

[34] S. Sen, J. Rexford, and D. Towsley. "Proxy Prefix Caching for Multimedia Streams," in *Proceedings of IEEE INFOCOM*, March 1999.

[35] B. Shen, S. J. Lee, and S. Basu. "Caching Strategies in Transcoding-Enabled Proxy Systems for Streaming Media Distribution Networks," *IEEE Transactions on Multimedia*, 6(2):375–386, April 2004.

[36] H. Sun, W. Kwok, and J. Zdepski. "Architectures for MPEG Compressed Bitstream Scaling," *IEEE Transactions on Circuits and Systems for Video Technology*, 6(2), April 1996.

[37] M. Sun and A. Reibman, editors. *Compressed Video over Networks*. Marcel Dekker, 2001.

[38] W. Tan and A. Zakhor. "Video Multicast Using Layered FEC and Scalable Compression," *IEEE Transactions on Circuits and Systems for Video Technology*, March 2001.

[39] A. Vetro, C. Christopoulos, and H. Sun. "Video Transcoding Architectures and Techniques: An Overview," *IEEE Signal Processing Magazine*, March 2003.

[40] Y. Wang, M. Hannuksela, V. Varsa, A. Hourunranta, and M. Gabbouj. "The Error Concealment Feature in the H.26L Test Model," in *Proceedings International Conference on Image Processing*, pages II-729–II-732, September 2002.

[41] Y. Wang and S. Lin. "Error Resilient Video Coding Using Multiple Description Motion Compensation," *IEEE Transactions on Circuits and Systems for Video Technology*, June 2002.

[42] S. Wee and J. Apostolopoulos. "Secure Scalable Streaming Enabling Transcoding without Decryption," *Proceedings of IEEE International Conference on Image Processing*, October 2001.

[43] S. Wee, J. Apostolopoulos, and N. Feamster. "Field-to-Frame Transcoding with Spatial and Temporal Downsampling," in *Proceedings of IEEE International Conference on Image Processing*, Kobe, Japan, October 1999.

[44] S. Wee, B. Shen, and J. Apostolopoulos. "Compressed-Domain Video Processing," *HP Labs Tech Report (HPL-2002-282)*, October 2002.

[45] S. Wee, W. Tan, J. Apostolopoulos, and M. Etoh. "Optimized Video Streaming for Networks with Varying Delay," in *Proceedings of IEEE International Conference on Multimedia and Expo*, Lausanne, Switzerland, August 2002.

[46] T. Yoshimura, Y. Yonemoto, T. Ohya, M. Etoh, and S. Wee. "Mobile Streaming Media CDN Enabled by Dynamic SMIL," in *Proceedings of the International World Wide Web Conference*, Honolulu, Hawaii, May 2002.

INDEX