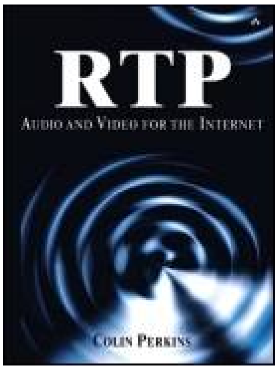


RTP: Audio and Video for the Internet

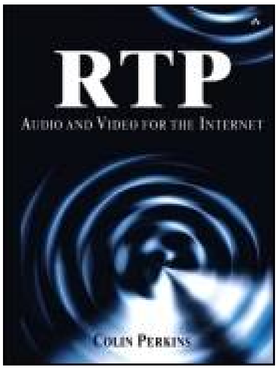
By Colin Perkins

Publisher: Addison Wesley
Pub Date: June 12, 2003
ISBN: 0-672-32249-8
Pages: 432

The Real-time Transport Protocol (RTP) provides a framework for delivery of audio and video across IP networks and unprecedented quality and reliability. In RTP: Audio and Video for the Internet, Colin Perkins, a leader of the RTP standardization process in the IETF, offers readers detailed technical guidance for designing, implementing, and managing any RTP-based system.

By bringing together crucial information that was previously scattered or difficult to find, Perkins has created an incredible resource that enables professionals to leverage RTP's benefits in a wide range of Voice-over IP(VoIP) and streaming media applications. He demonstrates how RTP supports audio/video transmission in IP networks, and shares strategies for maximizing performance, robustness, security, and privacy.

Comprehensive, exceptionally clear, and replete with examples, this book is the definitive RTP reference for every audio/video application designer, developer,researcher, and administrator.



RTP: Audio and Video for the Internet

By Colin Perkins

START READING

Publisher: Addison Wesley
Pub Date: June 12, 2003
ISBN: 0-672-32249-8
Pages: 432

Copyright

The publisher offers discounts on this book when ordered in quantity for bulk purchases and special sales. For more information, please contact:

U.S. Corporate and Government Sales

(800) 382-3419

corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact:

International Sales

(317) 581-3793

international@pearsontechgroup.com

Visit Addison-Wesley on the Web: www.awprofessional.com

Library of Congress Cataloging-in-Publication Data

LCCN: 2001089234

Preface

Introduction

This book describes the protocols, standards, and architecture of systems that deliver real-time voice, music, and video over IP networks, such as the Internet. These systems include voice-over-IP, telephony, teleconferencing, streaming video, and webcasting applications. The book focuses on media transport: how to deliver audio and video reliably across an IP network, how to ensure high quality in the face of network problems, and how to ensure that the system is secure.

The book adopts a standards-based approach, based around the Real-time Transport Protocol (RTP) and its associated profiles and payload formats. It describes the RTP framework, how to build a system that uses that framework, and extensions to RTP for security and reliability.

Many media codecs are suitable for use with RTP—for example, MPEG audio and video; ITU H.261 and H.263 video; G.711, G.722, G.726, G.728, and G.729 audio; and industry standards such as GSM, QCELP, and AMR audio. RTP implementations typically integrate existing media codecs, rather than developing them specifically. Accordingly, this book describes how media codecs are integrated into an RTP system, but not how media codecs are designed.

Call setup, session initiation, and control protocols, such as SIP, RTSP, and H.323, are also outside the scope of this book. Most RTP implementations are used as part of a complete system, driven by one of these control protocols. However, the interactions between the various parts of the system are limited, and it is possible to understand media transport without understanding the signaling. Similarly, session description using SDP is not covered, because it is part of the signaling.

Resource reservation is useful in some situations, but it is not required for the correct operation of RTP. This book touches on the use of resource reservation through both the Integrated Services and the Differentiated Services frameworks, but it does not go into details.

That these areas are not covered in this book does not mean that they are unimportant. A system using RTP will use a range of media codecs and will employ some form of call setup, session initiation, or control. The way this is done depends on the application, though: The needs of a telephony system are very different from those of a webcasting application. This book describes only the media transport layer that is common to all those systems.

Organization of the Book

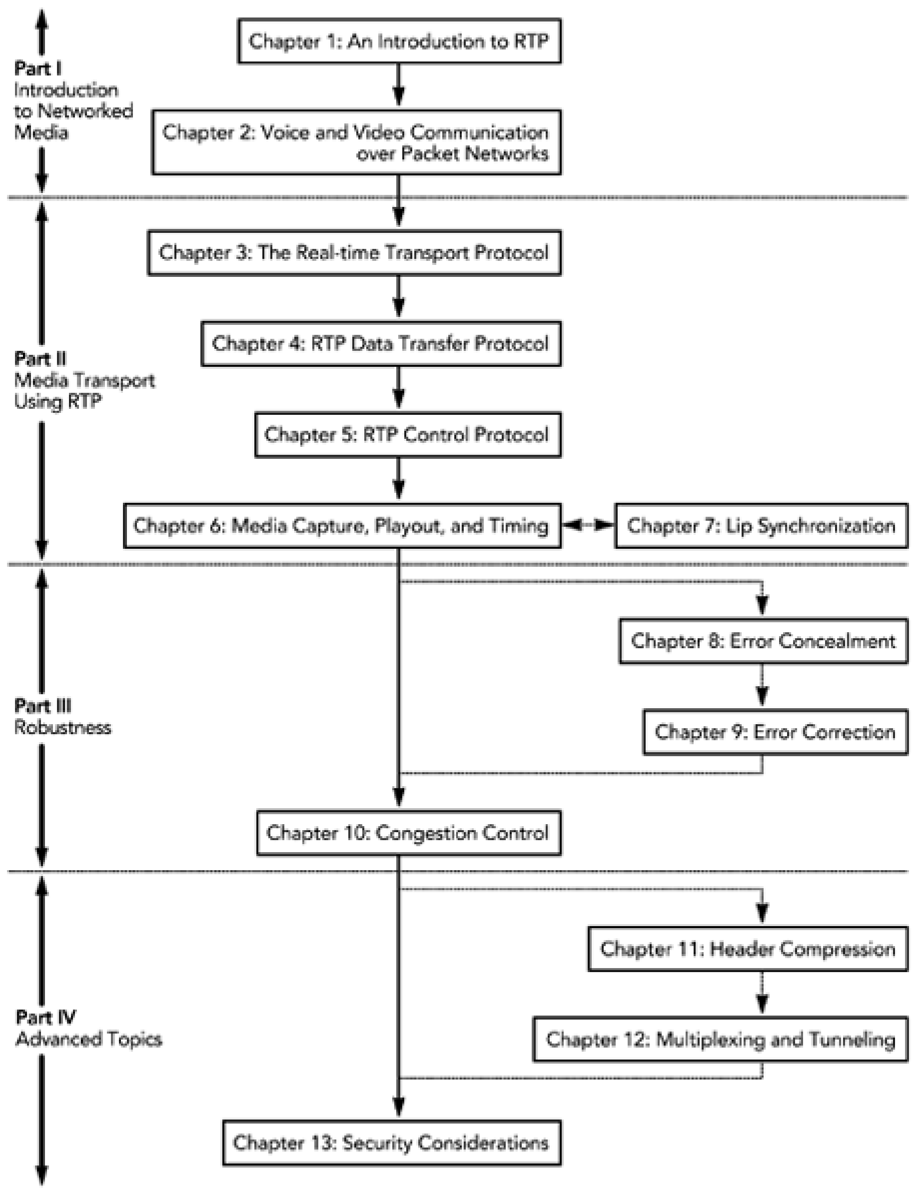
The book is logically divided into four parts: Part I, Introduction to Networked Multimedia, introduces the problem space, provides background, and outlines the properties of the Internet that affect audio/video transport:

- Chapter 1, An Introduction to RTP, gives a brief introduction to the Real-time Transport Protocol, outlines the relationship between RTP and other standards, and describes the scope of the book.

- Chapter 2, Voice and Video Communication over Packet Networks, describes the unique environment provided by IP networks, and how this environment affects packet audio/video applications.

The next five chapters, which constitute Part II, Media Transport Using RTP, discuss the basics of the Real-time Transport Protocol.

Road Map for This Book



You will need this information to design and build a tool for voice-over-IP, streaming music or video, and so on.

-

Intended Audience

This book describes audio/video transport over IP networks in considerable detail. It assumes some basic familiarity with IP network programming and the operation of network protocol stacks, and it builds on this knowledge to describe the features unique to audio/video transport. An extensive list of references is included, pointing readers to additional information on specific topics and to background reading material.

Several classes of readers might be expected to find this book useful:

- Engineers. The primary audience is those building voice-over-IP applications, teleconferencing systems, and streaming media and webcasting applications. This book is a guide to the design and implementation of the media engine of such systems. It should be read in conjunction with the relevant technical standards, and it builds on those standards to show how a system is built. This book does not discuss signaling (for example, SIP, RTSP, or H.323), which is a separate subject worthy of a book in its own right. Instead it talks in detail about media transport, and how to achieve good-quality audio and smooth-motion video over IP networks.
- Students. The book can be read as an accompaniment to a course in network protocol design or telecommunications, at either a graduate or an advanced undergraduate level. Familiarity with IP networks and layered protocol architectures is assumed. The unique aspects of protocols for real-time audio/video transport are highlighted, as are the differences from a typical layered system model. The cross-disciplinary nature of the subject is highlighted, in particular the relation between the psychology of human perception and the demands of robust media delivery.
- Researchers. Academics and industrial researchers can use this book as a source of information about the standards and algorithms that constitute the current state of the art in real-time audio/video transport over IP networks. Pointers to the literature are included in the References section, and they will be useful starting points for those seeking further depth and areas where more research is needed.
- Network administrators. An understanding of the technical protocols underpinning the common streaming audio/video applications is useful for those administering computer networks—to show how those applications can affect the behavior of the network, and how the network can be engineered to suit those applications better. This book includes extensive discussion of the most common network behavior (and how applications can adapt to it), the needs of congestion control, and the security implications of real-time audio/video traffic.

In summary, this book can be used as a reference, in conjunction with the technical standards, as a study guide, or as part of an advanced course on network protocol design or communication technology.

Acknowledgments

A book such as this is not written in isolation; rather it is shaped by the author's experiences and interactions with other researchers and practitioners. In large part, I gained my experience while working at University College London. I am grateful to Vicky Hardman, Peter Kirstein, and Angela Sasse for the opportunity to work on their projects, and to Anna Bouch, Ian Brown, Anna Conniff, Jon Crowcroft, Panos Gevros, Atanu Ghosh, Mark Handley, Tristan Henderson, Orion Hodson, Nadia Kausar, Isidor Kouvelas, Piers O'Hanlon, Louise Sheeran, Lorenzo Vicisano, and everyone else associated with G11, for providing a stimulating working environment, and a distracting social scene.

I wish to thank my colleagues at the USC Information Sciences Institute for their support, in particular Alec Aakesson, Helen Ellis, Jarda Flidr, Ladan Gharai, Tom Lehman, Dan Massey, and Nikhil Mittal. Allison Mankin provided the opportunity to work at USC/ISI, for which I am grateful.

On a personal note, Peter Phillips, Stewart Cambridge, Sonja Krugmann, and Alison Gardiner each helped me make the big move, in their own special way. I thank you.

The staff at Addison-Wesley did an excellent job in the production of this book. In particular, Dayna Isley and Jessica Goldstein provided encouragement to a new author and showed great patience during endless revisions. Thanks are also due to Amy Fleischer, Elizabeth Finney, Laurie McGuire, Cheri Clark, Rebecca Martin, and Stephanie Hiebert. The technical editors—Steve Casner and Orion Hodson—did sterling work, significantly improving the quality of the book, correcting many mistakes and contributing significantly to the text. Any errors that remain are mine alone.

Part I: Introduction to Networked Multimedia

Chapter 1. An Introduction to RTP

The Internet is changing: Static content is giving way to streaming video, text is being replaced by music and the spoken word, and interactive audio and video is becoming commonplace. These changes require new applications, and they pose new and unique challenges for application designers.

This book describes how to build these new applications: voice-over-IP, telephony, teleconferencing, streaming video, and webcasting. It looks at the challenges inherent in reliable delivery of audio and video across an IP network, and it explains how to ensure high quality in the face of network problems, as well as how to ensure that the system is secure. The emphasis is on open standards, in particular those devised by the Internet Engineering Task Force (IETF) and the International Telecommunications Union (ITU), rather than on proprietary solutions.

This chapter begins our examination of the Real-time Transport Protocol (RTP) with a brief look at the history of audio/video networking and an overview of RTP and its relation to other standards.

Throughout this text, extensive references are provided, as indicated by superscript numbers that map to entries in the References section at the end of the book. Because the RTP standard is still evolving, and because it intersects with so many other technologies, these references are provided to help readers gain additional background information and pursue further research interests.

A Brief History of Audio/Video Networking

The idea of using packet networks—such as the Internet—to transport voice and video is not new. Experiments with voice over packet networks stretch back to the early 1970s. The first RFC on this subject—the Network Voice Protocol (NVP)[1]—dates from 1977. Video came later, but still there is over ten years of experience with audio/video conferencing and streaming on the Internet.

Early Packet Voice and Video Experiments

The initial developers of NVP were researchers transmitting packet voice over the ARPANET, the predecessor to the Internet. The ARPANET provided a reliable-stream service (analogous to TCP/IP), but this introduced too much delay, so an "uncontrolled packet" service was developed, akin to the modern UDP/IP datagrams used with RTP. The NVP was layered directly over this uncontrolled packet service. Later the experiments were extended beyond the ARPANET to interoperate with the Packet Radio Network and the Atlantic Satellite Network (SATNET), running NVP over those networks.

All of these early experiments were limited to one or two voice channels at a time by the low bandwidth of the early networks. In the 1980s, the creation of the 3-Mbps Wideband Satellite Network enabled not only a larger number of voice channels but also the development of packet video. To access the one-hop, reserved-bandwidth, multicast service of the satellite network, a connection-oriented inter-network protocol called the Stream Protocol (ST) was developed. Both a second version of NVP, called NVP-II, and a companion Packet Video Protocol were transported over ST to provide a prototype packet-switched video teleconferencing service.

In 1989–1990, the satellite network was replaced with the Terrestrial Wideband Network and a research network called DARTnet while ST evolved into ST-II. The packet video conferencing system was put into scheduled production to support geographically distributed meetings of network researchers and others at up to five sites simultaneously.

ST and ST-II were operated in parallel with IP at the inter-network layer but achieved only limited deployment on government and research networks. As an alternative, initial deployment of conferencing using IP began on DARTnet, enabling multiparty conferences with NVP-II transported over multicast UDP/IP. At the March 1992 meeting of the IETF, audio was transmitted across the Internet to 20 sites on three continents over multicast "tunnels"—the Mbone (which stands for "multicast backbone")—extended from DARTnet. At that same meeting, development of RTP was begun.

Audio and Video on the Internet

Following from these early experiments, interest in video conferencing within the Internet community took hold in the early 1990s. At about this time, the processing power and multimedia capabilities of workstations and PCs became sufficient to enable the simultaneous capture, compression, and playback of audio and video streams. In parallel, development of IP multicast allowed the transmission of real-time data to any number of recipients connected to the Internet.

Video conferencing and multimedia streaming were obvious and well-executed multicast applications. Research groups took to developing tools such as vic and vat from the Lawrence Berkeley Laboratory,[87] nevot from the University of Massachusetts, the INRIA video conferencing system, nv from Xerox PARC, and rat from University College London.[77] These tools followed a new approach to conferencing, based on connectionless protocols, the end-to-end argument, and application-level framing.[65,70,76] Conferences were minimally managed, with no admission or floor control, and the transport layer was thin and adaptive. Multicast was used both for wide-area data transmission and as an interprocess communication mechanism between applications on the same machine (to exchange synchronization information between audio and video tools). The resulting collaborative environment consisted of lightly coupled applications and highly distributed participants.

The multicast conferencing (Mbone) tools had a significant impact: They led to widespread understanding of the problems inherent in delivering real-time media over IP networks, the need for scalable solutions, and error and congestion control. They also directly influenced the development of several key protocols and standards.

A Snapshot of RTP

The key standard for audio/video transport in IP networks is the Real-time Transport Protocol (RTP), along with its associated profiles and payload formats. RTP aims to provide services useful for the transport of real-time media, such as audio and video, over IP networks. These services include timing recovery, loss detection and correction, payload and source identification, reception quality feedback, media synchronization, and membership management. RTP was originally designed for use in multicast conferences, using the lightweight sessions model. Since that time, it has proven useful for a range of other applications: in H.323 video conferencing, webcasting, and TV distribution; and in both wired and cellular telephony. The protocol has been demonstrated to scale from point-to-point use to multicast sessions with thousands of users, and from low-bandwidth cellular telephony applications to the delivery of uncompressed High-Definition Television (HDTV) signals at gigabit rates.

RTP was developed by the Audio/Video Transport working group of the IETF and has since been adopted by the ITU as part of its H.323 series of recommendations, and by various other standards organizations. The first version of RTP was completed in January 1996.[6] RTP needs to be profiled for particular uses before it is complete; an initial profile was defined along with the RTP specification,[7] and several more profiles are under development. Profiles are accompanied by several payload format specifications, describing the transport of a particular media format. Development of RTP is ongoing, and a revision is nearing completion at the time of this writing.[49,50]

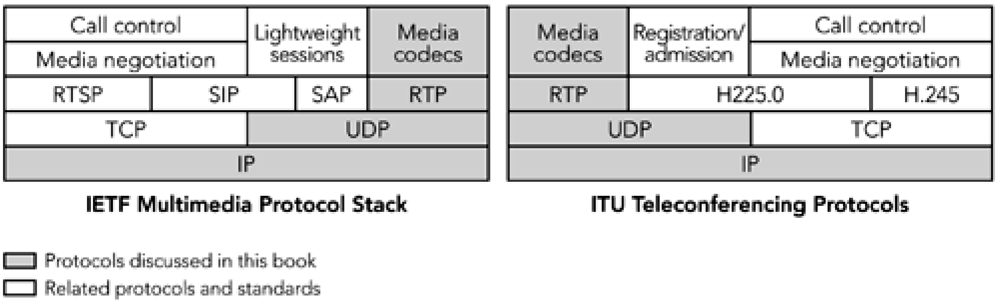
A detailed introduction to RTP is provided in Chapter 3, The Real-time Transport Protocol, and most of this book discusses the design of systems that use RTP and its various extensions.

Related Standards

In addition to RTP, a complete system typically requires the use of various other protocols and standards for session announcement, initiation, and control; media compression; and network transport.

Figure 1.1 shows how the negotiation and call control protocols, the media transport layer (provided by RTP), the compression-decompression algorithms (codecs), and the underlying network are related, according to both the IETF and ITU conferencing frameworks. The two parallel sets of call control and media negotiation standards use the same media transport framework. Like-wise, the media codecs are common no matter how the session is negotiated and irrespective of the underlying network transport.

Figure 1.1. IETF and ITU Protocols for Audio/Video Transport on the Internet



The relation between these standards and RTP is outlined further in Chapter 3, The Real-time Transport Protocol. However, the main focus of this book is media transport, rather than signaling and control.

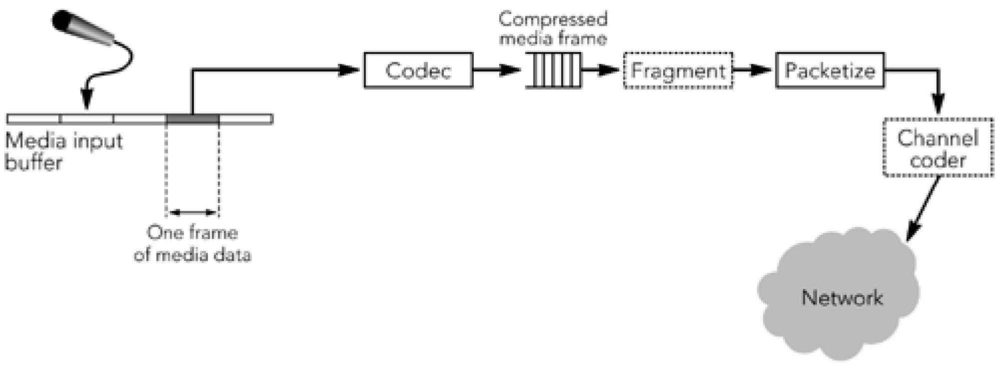
Overview of an RTP Implementation

As Figure 1.1 shows, the core of any system for delivery of real-time audio/video over IP is RTP: It provides the common media transport layer, independent of the signaling protocol and application. Before we look in more detail at RTP and the design of systems using RTP, it will be useful to have an overview of the responsibilities of RTP senders and receivers in a system.

Behavior of an RTP Sender

A sender is responsible for capturing and transforming audiovisual data for transmission, as well as for generating RTP packets. It may also participate in error correction and congestion control by adapting the transmitted media stream in response to receiver feedback. A diagram of the sending process is shown in Figure 1.2.

Figure 1.2. Block Diagram of an RTP Sender



Uncompressed media data—audio or video—is captured into a buffer, from which compressed frames are produced. Frames may be encoded in several ways depending on the compression algorithm used, and encoded frames may depend on both earlier and later data.

Compressed frames are loaded into RTP packets, ready for sending. If frames are large, they may be fragmented into several RTP packets; if they are small, several frames may be bundled into a single RTP packet. Depending on the error correction scheme in use, a channel coder may be used to generate error correction packets or to reorder packets before transmission.

After the RTP packets have been sent, the buffered media data corresponding to those packets is eventually freed. The sender must not discard data that might be needed for error correction or for the encoding process. This requirement may mean that the sender must buffer data for some time after the corresponding packets have been sent, depending on the codec and error correction scheme used.

The sender is responsible for generating periodic status reports for the media streams it is generating, including those required for lip synchronization. It also receives reception quality feedback from other participants and may use that information to adapt its transmission.

Behavior of an RTP Receiver

A receiver is responsible for collecting RTP packets from the network, correcting any losses, recovering the timing, decompressing the media, and presenting the result to the user. It also sends reception quality feedback, allowing the sender to adapt the transmission to the receiver, and it maintains a database of participants in the session. A possible block diagram for the receiving process is shown in Figure 1.3; implementations sometimes perform the operations in a different order depending on their needs.

Figure 1.3. Block Diagram of an RTP Receiver

Summary

This chapter has introduced the protocols and standards for real-time delivery of multimedia over IP networks, in particular the Real-time Transport Protocol (RTP). The remainder of this book discusses the features and use of RTP in detail. The aim is to expand on the standards documents, explaining both the rationale behind the standards and possible implementation choices and their trade-offs.

Chapter 2. Voice and Video Communication Over Packet Networks

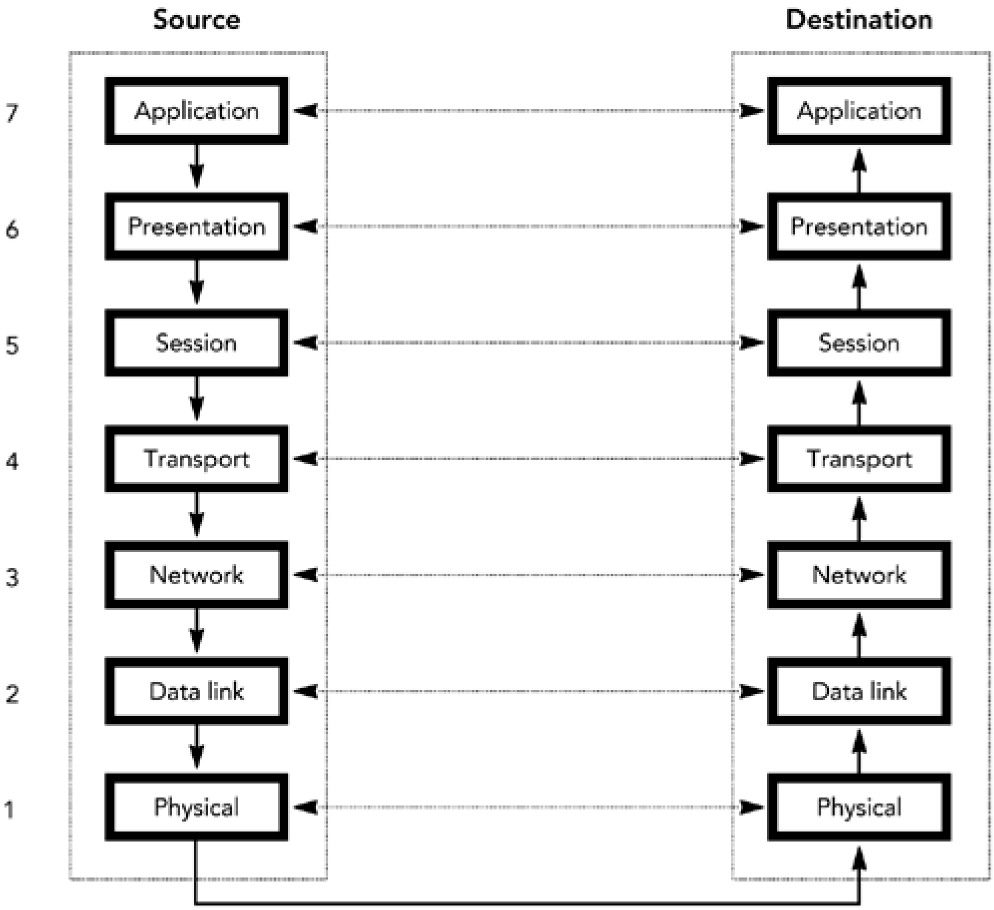
Before delving into details of RTP, you should understand the properties of IP networks such as the Internet, and how they affect voice and video communication. This chapter reviews the basics of the Internet architecture and outlines typical behavior of a network connection. This review is followed by a discussion of the transport requirements for audio and video, and how well these requirements are met by the network.

IP networks have unique characteristics that influence the design of applications and protocols for audio/video transport. Understanding these characteristics is vital if you are to appreciate the trade-offs involved in the design of RTP, and how they influence applications that use RTP.

TCP/IP and the OSI Reference Model

When you're thinking about computer networks, it is important to understand the concepts and implications of protocol layering. The OSI reference model,93 illustrated in Figure 2.1, provides a useful basis for discussion and comparison of layered systems.

Figure 2.1. The OSI Reference Model



The model comprises a set of seven layers, each building on the services provided by the lower layer and, in turn, providing a more abstract service to the layer above. The functions of the layers are as listed here:

1. Physical layer. The lowest layer—the physical layer—includes the physical network connection devices and protocols, such as cables, plugs, switches, and electrical standards.
2. Data link layer. The data link layer builds on the physical connection; for example, it turns a twisted-pair cable into Ethernet. This layer provides framing for data transport units, defines how the link is shared among multiple connected devices, and supplies addressing for devices on each link.
3. Network layer. The network layer connects links, unifying them into a single network. It provides addressing and routing of messages through the network. It may also provide control of congestion in the switches, prioritization of certain messages, billing, and so on. A network layer device processes messages received from one link and dispatches them to another, using routing information exchanged with its peers at the far ends of those links.
4. Transport layer. The transport layer is the first end-to-end layer. It takes responsibility for delivery of messages from one system to another, using the services provided by the network layer. This responsibility includes providing reliability and flow control if they are needed by the session layer and not provided by the

Performance Characteristics of an IP Network

As is apparent from the hourglass model of the Internet architecture, an application is hidden from the details of the lower layers by the abstraction of IP. This means it's not possible to determine directly the types of networks across which an IP packet will have traveled—it could be anything from a 14.4-kilobit cellular radio connection to a multi-gigabit optical fiber—or the level of congestion of that network. The only means of discovering the performance of the network are observation and measurement.

So what do we need to measure, and how do we measure it? Luckily, the design of the IP layer means that the number of parameters is limited, and that number often can be further constrained by the needs of the application. The most important questions we can ask are these:

- What is the probability that a packet will be lost in the network?
- What is the probability that a packet will be corrupted in the network?
- How long does a packet take to traverse the network? Is the transit time constant or variable?
- What size of packet can be accommodated?
- What is the maximum rate at which we can send packets?

The next section provides some sample measurements for the first four listed parameters. The maximum rate is closely tied to the probability that packets are lost in the network, as discussed in Chapter 10, Congestion Control.

What affects such measurements? The obvious factor is the location of the measurement stations. Measurements taken between two systems on a LAN will clearly show properties different from those of a transatlantic connection! But geography is not the only factor; the number of links traversed (often referred to as the number of hops), the number of providers crossed, and the times at which the measurements are taken all are factors. The Internet is a large, complex, and dynamic system, so care must be taken to ensure that any measurements are representative of the part of the network where an application is to be used.

We also have to consider what sort of network is being used, what other traffic is present, and how much other traffic is present. To date, the vast majority of network paths are fixed, wired (either copper or optical fiber) connections, and the vast majority of traffic (96% of bytes, 62% of flows, according to a recent estimate123) is TCP based. The implications of these traffic patterns are as follows:

- Because the infrastructure is primarily wired and fixed, the links are very reliable, and loss is caused mostly by congestion in the routers.
- TCP transport makes the assumption that packet loss is a signal that the bottleneck bandwidth has been reached, congestion is occurring, and it should reduce its sending rate. A TCP flow will increase its sending rate until loss is observed, and then back off, as a way of determining the maximum rate a particular connection can support. Of course, the result is a temporary overloading of the bottleneck link, which may affect other traffic.

If the composition of the network infrastructure or the traffic changes, other sources of loss may become important. For example, a large increase in the number of wireless users would likely increase the proportion of loss due to packet corruption and interference on the wireless links. In another example, if the proportion of multimedia traffic

Measuring IP Network Performance

This section outlines some of the available data on IP network performance, including published results on average packet loss, patterns of loss, packet corruption and duplication, transit time, and the effects of multicast.

Several studies have measured network behavior over a wide range of conditions on the public Internet. For example, Paxson reports on the behavior of 20,000 transfers among 35 sites in 9 countries;124,95 Handley122 and Bolot67,66 report on the behavior of multicast sessions; and Yajnik, Moon, Kurose, and Towsley report on the temporal dependence in packet loss statistics.89,108,109 Other sources of data include the traffic archives maintained by CAIDA (the Cooperative Association for Internet Data Analysis),117 the NLANR (National Laboratory for Applied Network Research),119 and the ACM (Association for Computing Machinery).116

Average Packet Loss

Various packet loss metrics can be studied. For example, the average loss rate gives a general measure of network congestion, while loss patterns and correlation give insights into the dynamics of the network.

The reported measurements of average packet loss rate show a range of conditions. For example, measurements of TCP/IP traffic taken by Paxson in 1994 and 1995 show that 30% to 70% of flows, depending on path taken and date, showed no packet loss, but of those flows that did show loss, the average loss ranged from 3% to 17% (these results are summarized in Table 2.1). Data from Bolot, using 64-kilobit PCM-encoded audio, shows similar patterns, with loss rates between 4% and 16% depending on time of day, although this data also dates from 1995. More recent results from Yajnik et al., taken using simulated audio traffic in 1997–1998, show lower loss rates of 1.38% to 11.03%. Handley's results—two sets of approximately 3.5 million packets of data and reception report statistics for multicast video sessions in May and September 1996—show loss averaged over five-second intervals varying between 0% and 100%, depending on receiver location and time of day. A sample for one particular receiver during a ten-hour period on May 29, 1996, plotted in Figure 2.5, shows the average loss rate, sampled over five-second intervals, varying between 0% and 20%.

Figure 2.5. Loss Rate Distribution versus Time122

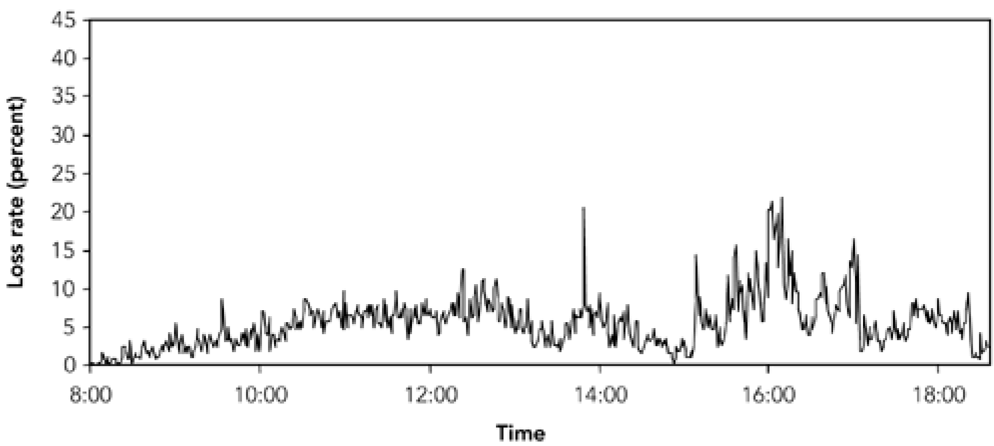


Table 2.1. Packet Loss Rates for Various Regions95

	Fraction of Flows Showing No Loss		Average Loss Rate for Flows with Loss	
Region	Dec. 1994	Dec. 1995	Dec. 1994	Dec. 1995
Within Europe	48%	58%	5.3%	5.9%
Within U.S.	66%	69%	3.6%	4.4%

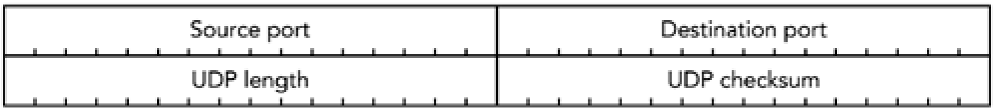
Effects of Transport Protocols

Thus far, our consideration of network characteristics has focused on IP. Of course, programmers almost never use the raw IP service. Instead, they build their applications on top of one of the higher-layer transport protocols, typically either UDP or TCP. These protocols provide additional features beyond those provided by IP. How do these added features affect the behavior of the network as seen by the application?

UDP/IP

The User Datagram Protocol (UDP) provides a minimal set of extensions to IP. The UDP header is shown in Figure 2.14. It comprises 64 bits of additional header representing source and destination port identifiers, a length field, and a checksum.

Figure 2.14. Format of a UDP Header



The source and destination ports identify the endpoints within the communicating hosts, allowing for multiplexing of different services onto different ports. Some services run on well-known ports; others use a port that is dynamically negotiated during call setup. The length field is redundant with that in the IP header. The checksum is used to detect corruption of the payload and is optional (it is set to zero for applications that have no use for a checksum).

Apart from the addition of ports and a checksum, UDP provides the raw IP service. It does not provide any enhanced reliability to the transport (although the checksum does allow for detection of payload errors that IP does not detect), nor does it affect the timing of packet delivery. An application using UDP provides data packets to the transport layer, which delivers them to a port on the destination machine (or to a group of machines if multicast is used). Those packets may be lost, delayed, or misordered in transit, exactly as observed for the raw IP service.

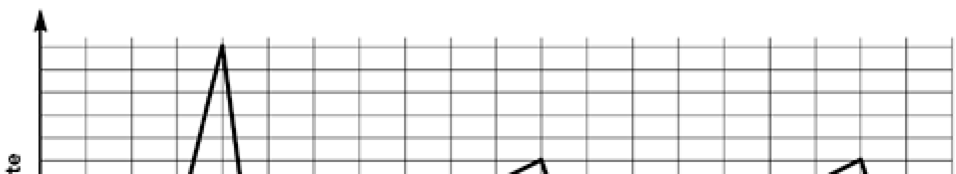
TCP/IP

The most common transport protocol on the Internet is TCP. Although UDP provides only a small set of additions to the IP service, TCP adds a significant amount of additional functionality: It abstracts the unreliable packet delivery service of IP to provide reliable, sequential delivery of a byte stream between ports on the source and a single destination host.

An application using TCP provides a stream of data to the transport layer, which fragments it for transmission in appropriately sized packets, and at a rate suitable for the network. Packets are acknowledged by the receiver, and those that are lost in transit are retransmitted by the source. When data arrives, it is buffered at the receiver so that it can be delivered in order. This process is transparent to the application, which simply sees a "pipe" of data flowing across the network.

As long as the application provides sufficient data, the TCP transport layer will increase its sending rate until the network exhibits packet loss. Packet loss is treated as a signal that the bandwidth of the bottleneck link has been exceeded and the connection should reduce its sending rate to match. Accordingly, TCP reduces its sending rate when loss occurs. This process continues, with TCP continually probing the sustainable transmission rate across the network; the result is a sending rate such as that illustrated in Figure 2.15.

Figure 2.15. Sample TCP Sending Rate



Requirements for Audio/Video Transport in Packet Networks

So far, this chapter has explored the characteristics of IP networks in some detail, and has looked briefly at the behavior of the transport protocols layered above them. We can now relate this discussion to real-time audio and video transport, consider the requirements for delivery of media streams over an IP network, and determine how well the network meets those requirements.

When we describe media as real-time, we mean simply that the receiver is playing out the media stream as it is received, rather than simply storing the complete stream in a file for later play-back. In the ideal case, playout at the receiver is immediate and synchronous, although in practice some unavoidable transmission delay is imposed by the network.

The primary requirement that real-time media places on the transport protocol is for predictable variation in network transit time. Consider, for example, an IP telephony system transporting encoded voice in 20-millisecond frames: The source will transmit one packet every 20 milliseconds, and ideally we would like those to arrive with the same spacing so that the speech they contain can be played out immediately. Some variation in transit time can be accommodated by the insertion of additional buffering delay at the receiver, but this is possible only if that variation can be characterized and the receiver can adapt to match the variation (this process is described in detail in Chapter 6, Media Capture, Playout, and Timing).

A lesser requirement is reliable delivery of all packets by the network. Clearly, reliable delivery is desirable, but many audio and video applications can tolerate some loss: In our IP telephony example, loss of a single packet will result in a dropout of one-fiftieth of a second, which, with suitable error concealment, is barely noticeable. Because of the time-varying nature of media streams, some loss is usually acceptable because its effects are quickly corrected by the arrival of new data. The amount of loss that is acceptable depends on the application, the encoding method used, and the pattern of loss. Chapter 8, Error Concealment, and Chapter 9, Error Correction, discuss loss tolerance.

These requirements drive the choice of transport protocol. It should be clear that TCP/IP is not appropriate because it favors reliability over timeliness, and our applications require timely delivery. A UDP/IP-based transport should be suitable, provided that the variation in transit time of the network can be characterized and loss rates are acceptable.

The standard Real-time Transport Protocol (RTP) builds on UDP/IP, and provides timing recovery and loss detection, to enable the development of robust systems. RTP and associated standards will be discussed in extensive detail in the remainder of this book.

Despite TCP's limitations for real-time applications, some audio/video applications use it for their transport. Such applications attempt to estimate the average throughput of the TCP connection and adapt their send rate to match. This approach can be made to work when tight end-to-end delay bounds are not required and an application has several seconds worth of buffering to cope with the variation in delivery time caused by TCP retransmission and congestion control. It does not work reliably for interactive applications, which need short end-to-end delay, because the variation in transit time caused by TCP is too great.

The primary rationale for the use of TCP/IP transport is that many firewalls pass TCP connections but block UDP. This situation is changing rapidly, as RTP-based systems become more prevalent and firewalls smarter. I strongly recommend that new applications be based on RTP-over-UDP/IP. RTP provides for higher quality by enabling applications to adapt in a way that is appropriate for real-time media, and by promoting interoperability (because it is an open standard).

Benefits of Packet-Based Audio/Video

At this stage you may be wondering why anyone would consider a packet-based audio or video application over an IP network. Such a network clearly poses challenges to the reliable delivery of real-time media streams. Although these challenges are real, an IP network has some distinct advantages that lead to the potential for significant gains in efficiency and flexibility, which can outweigh the disadvantages.

The primary advantage of using IP as the bearer service for realtime audio and video is that it can provide a unified

Summary

The properties of an IP network are significantly different from those of traditional telephony, audio, or video distribution networks. When designing applications that work over IP, you need to be aware of these unique characteristics, and make your system robust to their effects.

The remainder of this book will describe an architecture for such systems, explaining RTP and its model for timing recovery and lip synchronization, error correction and concealment, congestion control, header compression, multiplexing and tunneling, and security.

Part II: Media Transport Using RTP

Chapter 3. The Real-Time Transport Protocol

This chapter describes the design of the RTP framework starting with the philosophy and background of the design, gives an overview of the applicable standards, and explains how those standards interrelate. It concludes with a discussion of possible future directions for the development of those standards.

Fundamental Design Philosophies of RTP

The challenge facing the designers of RTP was to build a mechanism for robust, real-time media delivery above an unreliable transport layer. They achieved this goal with a design that follows the twin philosophies of application-level framing and the end-to-end principle.

Application-Level Framing

The concepts behind application-level framing were first elucidated by Clark and Tennenhouse65 in 1990. Their central thesis is that only the application has sufficient knowledge of its data to make an informed decision about how that data should be transported. The implication is that a transport protocol should accept data in application-meaningful units (application data units, ADUs) and expose the details of their delivery as much as possible so that the application can make an appropriate response if an error occurs. The application partners with the transport, cooperating to achieve reliable delivery.

Application-level framing comes from the recognition that there are many ways in which an application can recover from network problems, and that the correct approach depends on both the application and the scenario in which it is being used. In some cases it is necessary to retransmit an exact copy of the lost data. In others, a lower-fidelity copy may be used, or the data may have been superseded, so the replacement is different from the original. Alternatively, the loss can be ignored if the data was of only transient interest. These choices are possible only if the application interacts closely with the transport.

The goal of application-level framing is somewhat at odds with the design of TCP, which hides the lossy nature of the underlying IP network to achieve reliable delivery at the expense of timeliness. It does, however, fit well with UDP-based transport and with the characteristics of real-time media. As noted in Chapter 2, Voice and Video Communication over Packet Networks, real-time audio and visual media is often loss tolerant but has strict timing bounds. By using application-level framing with UDP-based transport, we are able to accept losses where necessary, but we also have the flexibility to use the full spectrum of recovery techniques, such as retransmission and forward error correction, where appropriate.

These techniques give an application great flexibility to react to network problems in a suitable manner, rather than being constrained by the dictates of a single transport layer.

A network that is designed according to the principles of application-level framing should not be specific to a particular application. Rather it should expose the limitations of a generic transport layer so that the application can cooperate with the network in achieving the best possible delivery. Application-level framing implies a weakening of the strict layers defined by the OSI reference model. It is a pragmatic approach, acknowledging the importance of layering, but accepting the need to expose some details of the lower layers.

The philosophy of application-level framing implies smart, network-aware applications that are capable of reacting to problems.

The End-to-End Principle

The other design philosophy adopted by RTP is the end-to-end principle.70 It is one of two approaches to designing a system that must communicate reliably across a network. In one approach, the system can pass responsibility for the correct delivery of data along with that data, thus ensuring reliability hop by hop. In the other approach, the responsibility for data can remain with the endpoints, ensuring reliability end-to-end even if the individual hops are unreliable. It is this second end-to-end approach that permeates the design of the Internet, with both TCP and RTP following the end-to-end principle.

The main consequence of the end-to-end principle is that intelligence tends to bubble up toward the top of the protocol stack. If the systems that make up the network path never take responsibility for the data, they can be simple and do not need to be robust. They may discard data that they cannot deliver, because the endpoints will recover without their help. The end-to-end principle implies that intelligence is at the endpoints, not within the network.

Standard Elements of RTP

The primary standard for audio/video transport in IP networks is the Real-time Transport Protocol (RTP), along with associated profiles and payload formats. RTP was developed by the Audio/Video Transport working group of the Internet Engineering Task Force (IETF), and it has since been adopted by the International Telecommunications Union (ITU) as part of its H.323 series of recommendations, and by several other standards organizations.

RTP provides a framework for the transport of real-time media and needs to be profiled for particular uses before it is complete. The RTP profile for audio and video conferences with minimal control was standardized along with RTP, and several more profiles are under development. Each profile is accompanied by several payload format specifications, each of which describes the transport of a particular media format.

The RTP Specification

RTP was published as an IETF proposed standard (RFC 1889) in January 1996,[6] and its revision for draft standard status is almost complete.[50] The first revision of ITU recommendation H.323 included a verbatim copy of the RTP specification; later revisions reference the current IETF standard.

> In the IETF standards process,[8] a specification undergoes a development cycle in which multiple Internet drafts are produced as the details of the design are worked out. When the design is complete, it is published as a proposed standard RFC. A proposed standard is generally considered stable, with all known design issues worked out, and suitable for implementation. If that proposed standard proves useful, and if there are independent and interoperable implementations of each feature of that standard, it can then be advanced to draft standard status (possibly involving changes to correct any problems found in the proposed standard). Finally, after extensive experience, it may be published as a full standard RFC. Advancement beyond proposed standard status is a significant hurdle that many protocols never achieve.

RTP typically sits on top of UDP/IP transport, enhancing that transport with loss detection and reception quality reporting, provision for timing recovery and synchronization, payload and source identification, and marking of significant events within the media stream. Most implementations of RTP are part of an application or library that is layered above the UDP/IP sockets interface provided by the operating system. This is not the only possible design, though, and nothing in the RTP protocol requires UDP or IP. For example, some implementations layer RTP above TCP/IP, and others use RTP on non-IP networks, such as Asynchronous Transfer Mode (ATM) networks.

There are two parts to RTP: the data transfer protocol and an associated control protocol. The RTP data transfer protocol manages delivery of real-time data, such as audio and video, between end systems. It defines an additional level of framing for the media payload, incorporating a sequence number for loss detection, timestamp to enable timing recovery, payload type and source identifiers, and a marker for significant events within the media stream. Also specified are rules for timestamp and sequence number usage, although these rules are somewhat dependent on the profile and payload format in use, and for multiplexing multiple streams within a session. The RTP data transfer protocol is discussed further in Chapter 4.

The RTP control protocol (RTCP) provides reception quality feedback, participant identification, and synchronization between media streams. RTCP runs alongside RTP and provides periodic reporting of this information. Although data packets are typically sent every few milliseconds, the control protocol operates on the scale of seconds. The information sent in RTCP is necessary for synchronization between media streams—for example, for lip synchronization between audio and video—and can be useful for adapting the transmission according to reception quality feedback, and for identifying the participants. The RTP control protocol is discussed further in Chapter 5.

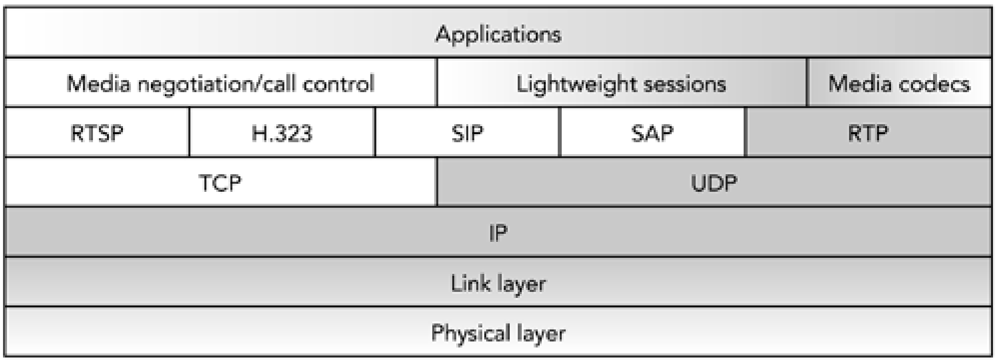
RTP supports the notion of mixers and translators, middle boxes that can operate on the media as it flows between endpoints. These may be used to translate an RTP session between different lower-layer protocols—for example, bridging between participants on IPv4 and IPv6 networks, or bringing a unicast-only participant into a multicast

Related Standards

In addition to the RTP framework, a complete system will typically require the use of various other protocols and standards for call setup and control, session description, multiparty communication, and signaling quality-of-service requirements. Although this book does not cover the use of such protocols in detail, in this section it provides pointers to their specification and further reading.

The complete multimedia protocol stack is illustrated in Figure 3.1, showing the relationship between the RTP framework and the supporting setup and control protocols. The protocols and functions that are discussed in this book are highlighted.

Figure 3.1. The Multimedia Protocol Stack



Call Setup and Control

Various call setup, control, and advertisement protocols can be used to start an RTP session, depending on the application scenario:

- For the purpose of starting an interactive session, be it a voice telephony call or a video conference, there are two standards. The original standard in this area was ITU recommendation H.323,62 and more recently the IETF has defined the Session Initiation Protocol (SIP).28,111
- For the purpose of starting a noninteractive session—for example, video-on-demand—the main standard is the Real-Time Streaming Protocol (RTSP).14
- The original use of RTP was with IP multicast and the lightweight sessions model of conferencing. This design used the Session Announcement Protocol (SAP)35 and IP multicast to announce ongoing sessions, such as seminars and TV broad-casts, that were open to the public.

The requirements for these protocols are quite distinct, in terms of the number of participants in the session and the coupling between those participants. Some sessions are very loosely coupled, with only limited membership control and knowledge of the participants. Others are closely managed, with explicit permission required to join, talk, listen, and watch.

These different requirements have led to very different protocols being designed for each scenario, with tremendous ongoing work in this area. RTP deliberately does not include session initiation and control functions, making it suitable for a wide range of applications.

As an application designer, you will have to implement some form of session initiation, call setup, or call control in addition to the media transport provided by RTP.

Session Description

Future Standards Development

With the revision of RTP for draft standard status, there are no known unresolved issues with the protocol specification, and RTP itself is not expected to change in the foreseeable future. This does not mean that the standards work is finished, though. New payload formats are always under development, and work on new profiles will extend RTP to encompass new functionality (for example, the profiles for secure RTP and enhanced feedback).

In the long term, we expect the RTP framework to evolve along with the network itself. Future changes in the network may also affect RTP, and we expect new profiles to be developed to take advantage of any changes. We also expect a continual series of new payload format specifications, to keep up with changes in codec technology and to provide new error resilience schemes.

Finally, we can expect considerable changes in the related protocols for call setup and control, resource reservation, and quality of service. These protocols are newer than RTP, and they are currently undergoing rapid development, implying that changes here will likely be more substantial than changes to RTP, its profile, and payload formats.

Summary

RTP provides a flexible framework for delivery of real-time media, such as audio and video, over IP networks. Its core philosophies—application-level framing and the end-to-end principle—make it well suited to the unique environment of IP networks.

This chapter has provided an overview of the RTP specification, profiles, and payload formats. Related standards cover call setup, control and advertisement, and resource reservation.

The two parts of RTP introduced in this chapter—the data transfer protocol and the control protocol—are covered in detail in the next two chapters.

Chapter 4. RTP Data Transfer Protocol

This chapter explains the RTP data transfer protocol, the means by which real-time media is exchanged. The discussion focuses on the "on-the-wire" aspects of RTP—that is, the packet formats and requirements for interoperability; the design of a system using RTP is explained in later chapters.

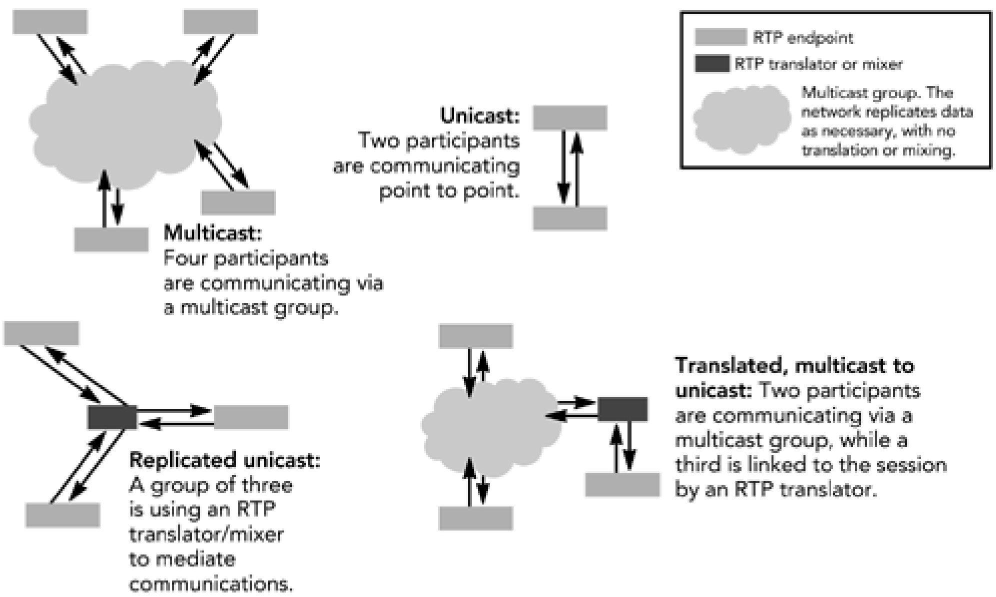
RTP Sessions

A session consists of a group of participants who are communicating using RTP. A participant may be active in multiple RTP sessions—for instance, one session for exchanging audio data and another session for exchanging video data. For each participant, the session is identified by a network address and port pair to which data should be sent, and a port pair on which data is received. The send and receive ports may be the same. Each port pair comprises two adjacent ports: an even-numbered port for RTP data packets, and the next higher (odd-numbered) port for RTCP control packets. The default port pair is 5004 and 5005 for UDP/IP, but many applications dynamically allocate ports during session setup and ignore the default. RTP sessions are designed to transport a single type of media; in a multimedia communication, each media type should be carried in a separate RTP session.

> The latest revision to the RTP specification relaxes the requirement that the RTP data port be even-numbered, and allows non-adjacent RTP and RTCP ports. This change makes it possible to use RTP in environments where certain types of Network Address Translation (NAT) devices are present. If possible, for compatibility with older implementations, it is wise to use adjacent ports, even though this is not strictly required.

A session can be unicast, either directly between two participants (a point-to-point session) or to a central server that redistributes the data. Or it can be multicast to a group of participants. A session also need not be restricted to a single transport address space. For example, RTP translators can be used to bridge a session between unicast and multicast, or between IP and another transport, such as IPv6 or ATM. Translators are discussed in more detail later in this chapter, in the section titled Translators and Mixers. Some examples of session topologies are shown in Figure 4.1.

Figure 4.1. Types of RTP Sessions



The range of possible sessions means that an RTP end system should be written to be essentially agnostic about the underlying transport. It is good design to restrict knowledge of the transport address and ports to your low-level networking code only, and to use RTP-level mechanisms for participant identification. RTP provides a "synchronization source" for this purpose, described in more detail later in this chapter.

In particular, note these tips:

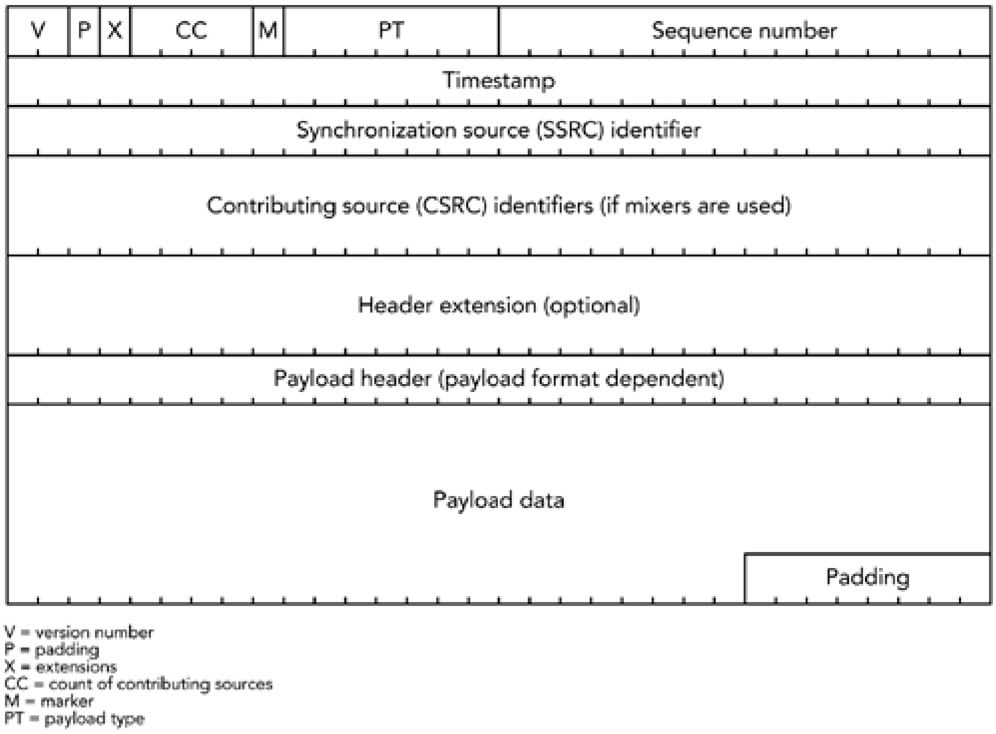
- You should not use a transport address as a participant identifier because the data may have passed through a translator or mixer that may hide the original source address. Instead, use the synchronization source identifiers.
-

The RTP Data Transfer Packet

The format of an RTP data transfer packet is illustrated in Figure 4.2. There are four parts to the packet:

1. The mandatory RTP header
2. An optional header extension
3. An optional payload header (depending on the payload format used)
4. The payload data itself

Figure 4.2. An RTP Data Transfer Packet

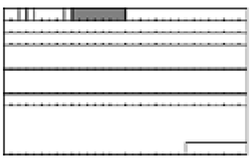


The entire RTP packet is contained within a lower-layer payload, typically UDP/IP.

Header Elements

The mandatory RTP data packet header is typically 12 octets in length, although it may contain a contributing source list, which can expand the length by 4 to 60 additional octets. The fields in the mandatory header are the payload type, sequence number, time-stamp, and synchronization source identifier. In addition, there is a count of contributing sources, a marker for interesting events, support for padding and a header extension, and a version number.

PAYLOAD TYPE



The payload type, or PT, field of the RTP header identifies the media transported by an RTP packet. The receiving application examines the payload type to determine how to treat the data—for example, passing it to a particular

Packet Validation

Because RTP sessions typically use a dynamically negotiated port pair, it is especially important to validate that packets received really are RTP, and not misdirected other data. At first glance, confirming this fact is nontrivial because RTP packets do not contain an explicit protocol identifier; however, by observing the progression of header fields over several packets, we can quickly obtain strong confidence in the validity of an RTP stream.

Possible validity checks that can be performed on a stream of RTP packets are outlined in Appendix A of the RTP specification. There are two types of tests:

1. Per-packet checking, based on fixed known values of the header fields. For example, packets in which the version number is not equal to 2 are invalid, as are those with an unexpected payload type.
2. Per-flow checking, based on patterns in the header fields. For example, if the SSRC is constant, and the sequence number increments by one with each packet received, and the timestamp intervals are appropriate for the payload type, this is almost certainly an RTP flow and not a misdirected stream.

The per-flow checks are more likely to detect invalid packets, but they require additional state to be kept in the receiver. This state is required for a valid source, but care must be taken because holding too much state to detect invalid sources can lead to a denial-of-service attack, in which a malicious source floods a receiver with a stream of bogus packets designed to use up resources.

A robust implementation will employ strong per-packet validity checks to weed out as many invalid packets as possible before committing resources to the per-flow checks to catch the others. It should also be prepared to aggressively discard state for sources that appear to be bogus, to mitigate the effects of denial-of-service attacks.

It is also possible to validate the contents of an RTP data stream against the corresponding RTCP control packets. To do this, the application discards RTP packets until an RTCP source description packet with the same SSRC is received. This is a very strong validity check, but it can result in significant validation delay, particularly in large sessions (because the RTCP reporting interval can be many seconds). For this reason we recommend that applications validate the RTP data stream directly, using RTCP as confirmation rather than the primary means of validation.

Translators and Mixers

In addition to normal end systems, RTP supports middle boxes that can operate on a media stream within a session. Two classes of middle boxes are defined: translators and mixers.

Translators

A translator is an intermediate system that operates on RTP data while maintaining the synchronization source and timeline of a stream. Examples include systems that convert between media-encoding formats without mixing, that bridge between different transport protocols, that add or remove encryption, or that filter media streams. A translator is invisible to the RTP end systems unless those systems have prior knowledge of the untranslated media. There are a few classes of translators:

- Bridges. Bridges are one-to-one translators that don't change the media encoding—for example, gateways between different transport protocols, like RTP/UDP/IP and RTP/ATM, or RTP/UDP/IPv4 and RTP/UDP/IPv6. Bridges make up the simplest class of translator, and typically they cause no changes to the RTP or RTCP data.
- Transcoders. Transcoders are one-to-one translators that change the media encoding—for example, decoding the compressed data and reencoding it with a different payload format—to better suit the characteristics of the output network. The payload type usually changes, as may the padding, but other RTP header fields generally remain unchanged. These translations require state to be maintained so that the RTCP sender reports can be adjusted to match, because they contain counts of source bit rate.
- Exploders. Exploders are one-to-many translators, which take in a single packet and produce multiple packets. For example, they receive a stream in which multiple frames of codec output are included within each RTP packet, and they produce output with a single frame per packet. The generated packets have the same SSRC, but the other RTP header fields may have to be changed, depending on the translation. These translations require maintenance of bidirectional state: The translator must adjust both outgoing RTCP sender reports and returning receiver reports to match.
- Mergers. Mergers are many-to-one translators, combining multiple packets into one. This is the inverse of the previous category, and the same issues apply.

The defining characteristic of a translator is that each input stream produces a single output stream, with the same SSRC. The translator itself is not a participant in the RTP session—it does not have an SSRC and does not generate RTCP itself—and is invisible to the other participants.

Mixers

A mixer is an intermediate system that receives RTP packets from a group of sources and combines them into a single output, possibly changing the encoding, before forwarding the result. Examples include the networked equivalent of an audio mixing deck, or a video picture-in-picture device.

Because the timing of the input streams generally will not be synchronized, the mixer will have to make its own adjustments to synchronize the media before combining them, and hence it becomes the synchronization source of the output media stream. A mixer may use playout buffers for each arriving media stream to help maintain the timing relationships between streams. A mixer has its own SSRC, which is inserted into the data packets it generates. The SSRC identifiers from the input data packets are copied into the CSRC list of the output packet.

A mixer has a unique view of the session: It sees all sources as synchronization sources, whereas the other

Summary

This chapter has described the on-the-wire aspects of the RTP data transfer protocol in some detail. We considered the format of the RTP header and its use, including the payload type for identification of the format of the data, sequence number to detect loss, timestamp to show when to play out data, and synchronization source as a participant identifier. We also discussed the minor header fields: marker, padding, and version number.

The concept of the payload format, and its mapping onto payload type identifier and payload header, should now be apparent, showing how RTP is tailored to different types of media. This is an important topic, to which we will return in later chapters.

Finally, we discussed RTP translators and mixers: intermediate systems that extend the reach of RTP in a controlled manner, allowing sessions to bridge heterogeneity of the network.

Associated with the RTP data transfer protocol is a control channel, RTCP, which has been mentioned several times in this chapter. The next chapter focuses on this control channel in some depth, completing our discussion of the network aspects of RTP.

Chapter 5. RTP Control Protocol

There are two parts to RTP: the data transfer protocol, which was described in Chapter 4, and an associated control protocol, which is described in this chapter. The control protocol, RTCP, provides for periodic reporting of reception quality, participant identification and other source description information, notification on changes in session membership, and the information needed to synchronize media streams.

This chapter describes the uses of RTCP, the format of RTCP packets, and the timing rules used to scale RTCP over the full range of session sizes. It also discusses the issues in building a participant database, using the information contained in RTCP packets.

Components of RTCP

An RTCP implementation has three parts: the packet formats, the timing rules, and the participant database.

There are several types of RTCP packets. The five standard packet types are described in the section titled RTCP Packet Formats later in this chapter, along with the rules by which they must be aggregated into compound packets for transmission. Algorithms by which implementations can check RTCP packets for correctness are described in the section titled Packet Validation.

The compound packets are sent periodically, according to the rules described in the section titled Timing Rules later in this chapter. The interval between packets is known as the reporting interval. All RTCP activity happens in multiples of the reporting interval. In addition to being the time between packets, it is the time over which reception quality statistics are calculated, and the time between updates of source description and lip synchronization information. The interval varies according to the media format in use and the size of the session; typically it is on the order of 5 seconds for small sessions, but it can increase to several minutes for very large groups. Senders are given special consideration in the calculation of the reporting interval, so their source description and lip synchronization information is sent frequently; receivers report less often.

Each implementation is expected to maintain a participant database, based on the information collected from the RTCP packets it receives. This database is used to fill out the reception report packets that have to be sent periodically, but also for lip synchronization between received audio and video streams and to maintain source description information. The privacy concerns inherent in the participant database are mentioned in the section titled Security and Privacy later in this chapter. The Participant Database section, also in this chapter, describes the maintenance of the participant database.

Transport of RTCP Packets

Each RTP session is identified by a network address and a pair of ports: one for RTP data and one for RTCP data. The RTP data port should be even, and the RTCP port should be one above the RTP port. For example, if media data is being sent on UDP port 5004, the control channel will be sent to the same address on UDP port 5005.

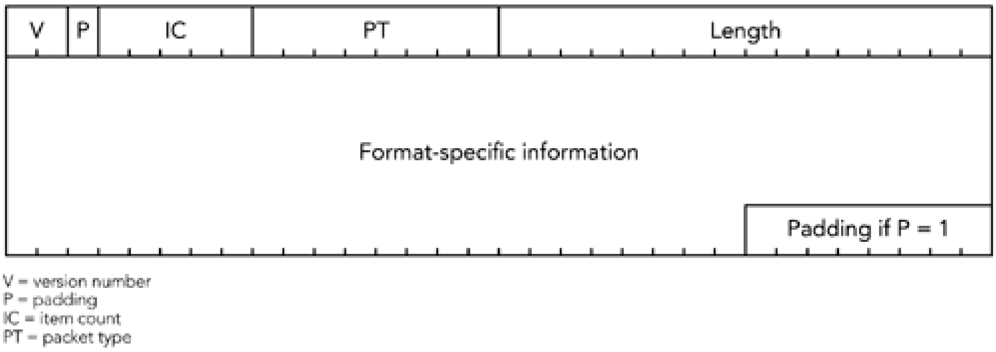
> As noted in Chapter 4, RTP Data Transfer Protocol, the latest revision of the RTP specification relaxes the requirement that the RTCP port is odd-numbered, and allows nonadjacent RTP and RTCP ports. For compatibility with older implementations, it is still wise to use adjacent ports where possible, with the RTCP data being sent on the higher, odd-numbered, port.

All participants in a session should send compound RTCP packets and, in turn, will receive the compound RTCP packets sent by all other participants. Note that feedback is sent to all participants in a multiparty session: either unicast to a translator, which then redistributes the data, or directly via multicast. The peer-to-peer nature of RTCP gives each participant in a session knowledge of all other participants: their presence, reception quality, and—optionally—personal details such as name, e-mail address, location, and phone number.

RTCP Packet Formats

Five types of RTCP packets are defined in the RTP specification: receiver report (RR), sender report (SR), source description (SDES), membership management (BYE), and application-defined (APP). They all follow a common structure—illustrated in Figure 5.1—although the format-specific information changes depending on the type of packet.

Figure 5.1. The Basic RTCP Packet Format



The header that all five packet types have in common is four octets in length, comprising five fields:

1. Version number (V). The version number is always 2 for the current version of RTP. There are no plans to introduce new versions, and previous versions are not in widespread use.
2. Padding (P). The padding bit indicates that the packet has been padded out beyond its natural size. If this bit is set, one or more octets of padding have been added to the end of this packet, and the last octet contains a count of the number of padding octets added. Its use is much the same as the padding bit in RTP data packets, which was discussed in Chapter 4, RTP Data Transfer Protocol, in the section titled Padding. Incorrect use of the padding bit has been a common problem with RTCP implementations; the correct usage is described in the sections titled Packing Issues and Packet Validation later in this chapter.
3. Item count (IC). Some packet types contain a list of items, perhaps in addition to some fixed, type-specific information. The item count field is used by these packet types to indicate the number of items included in the packet (the field has different names in different packet types depending on its use). Up to 31 items may be included in each RTCP packet, limited also by the maximum transmission unit of the network. If more than 31 items are needed, the application must generate multiple RTCP packets. An item count of zero indicates that the list of items is empty (this does not necessarily mean that the packet is empty). Packet types that don't need an item count may use this field for other purposes.
4. Packet type (PT). The packet type identifies the type of information carried in the packet. Five standard packet types are defined in the RTP specification; other types may be defined in the future (for example, to report additional statistics or to convey other source-specific information).
5. Length. The length field denotes the length of the packet contents following the common header. It is measured in units of 32-bit words because all RTCP packets are multiples of 32 bits in length, so counting octets would only allow the possibility of inconsistency. Zero is a valid length, indicating that the packet consists of only the four-octet header (the IC header field will also be zero in this case).

Following the RTCP header is the packet data (the format of which depends on the packet type) and optional padding. The combination of header and data is an RTCP packet. The five standard types of RTCP packets are described in the sections that follow.

Security and Privacy

Various privacy issues are inherent in the use of RTCP—in particular, source description packets. Although these packets are optional, their use can expose significant personal details, so applications should not send SDES information without first informing the user that the information is being made available.

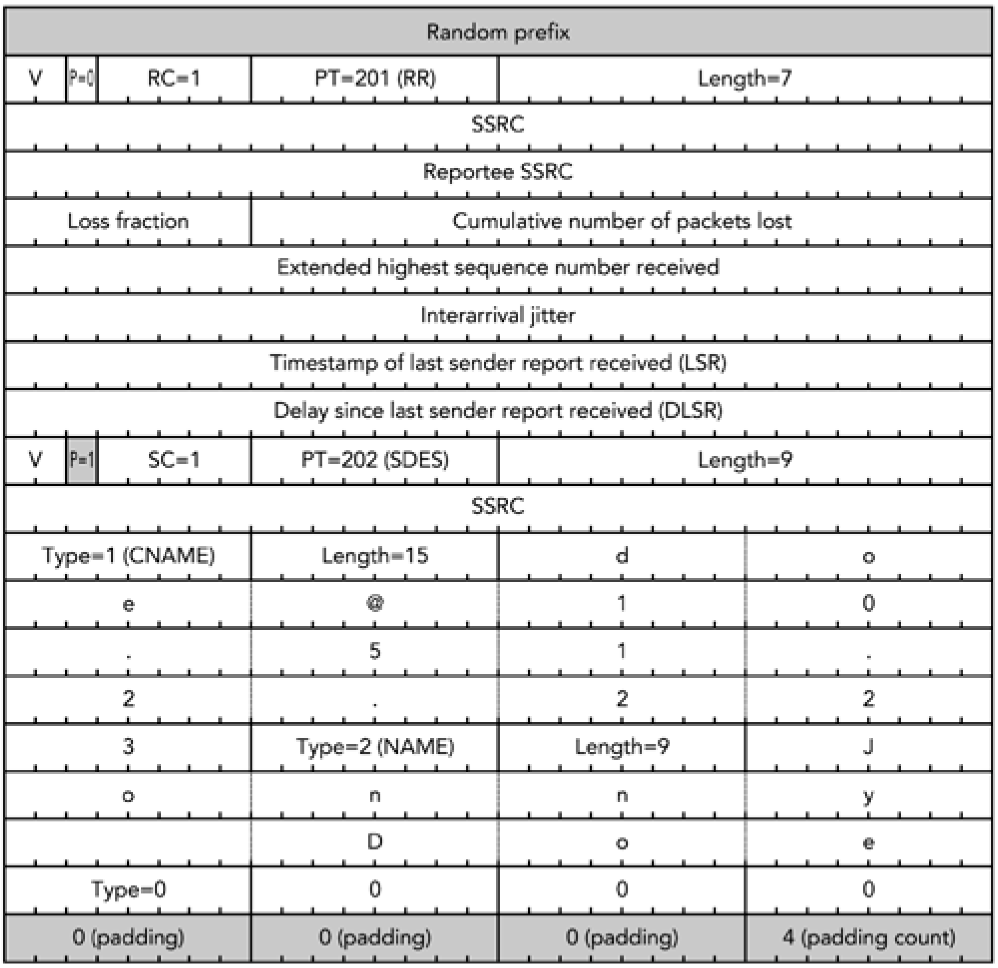
The use of SDES CNAME packets is an exception because these packets are mandatory. The inclusion of an IP address within CNAME packets is a potential issue. However, the same information is available from the IP header of the packet. If the RTP packets pass through Network Address Translation (NAT), the translation of the address in the IP header that is performed should also be performed on the address in the CNAME. In practice, many NAT implementations are unaware of RTP, so there is a potential for leakage of the internal IP address.

The exposure of user names may be a greater concern—in which case applications may omit or rewrite the user name, provided that this is done consistently among the set of applications using CNAME for association.

Some receivers may not want their presence to be visible. It is acceptable if those receivers do not send RTCP at all, although doing so prevents senders from using the reception quality information to adapt their transmission to match the receivers.

To achieve confidentiality of the media stream, RTCP packets may be encrypted. When encrypted, each compound packet contains an additional 32-bit random prefix, as illustrated in Figure 5.12, to help avoid plain-text attacks.

Figure 5.12. Example of an Encrypted RTCP Packet, Showing the Correct Use of Padding



Security and privacy are discussed in more detail in Chapter 13, Security Considerations.

Packet Validation

It is important to validate whether received packets really are RTP or RTCP. The packing rules, mentioned earlier, allow RTCP packets to be rigorously validated. Successful validation of an RTCP stream gives high assurance that the corresponding RTP stream is also valid, although it does not negate the need for validation of the RTP packets.

Listing 5.1 shows the pseudocode for the validation process. These are the key points:

- All packets must be compound RTCP packets.
- The version field of all packets must equal 2.
- The packet type field of the first RTCP packet in a compound packet must be equal to SR or RR.
- If padding is needed, it is added to only the last packet in the compound. The padding bit should be zero for all other packets in the compound RTCP packet.
- The length fields of the individual RTCP packets must total the overall length of the compound RTCP packet as received.

Because new RTCP packet types may be defined by future profiles, the validation procedure should not require each packet type to be one of the five defined in the RTP specification.

Listing 5.1 Pseudocode for Packet Validation

```
validate_rtcp(rtcp_t *packet, int length)

    rtcp_t    *end   = (rtcp_t *) (((char *) packet) + length);
    rtcp_t    *r     = packet;
    int         l     = 0;
    int         p     = 0;
    // All RTCP packets must be compound packets
    if ((packet->length+ 1) * 4) == length) {
        ... error: not a compound packet
    }
    // Check the RTCP version, packet type, and padding of the first
    // in the compound RTCP packet...
    if (packet->version != 2) {
        ...error: version number != 2 in the first subpacket
    }
    if (packet-> p != 0) {
        ...error: padding bit is set on first packet in compound
    }
    if ((packet->pt != RTCP_SR) && (packet->pt != RTCP_RR)) {
        ...error: compound packet does not start with SR or RR
    }
    // Check all following parts of the compound RTCP packet. The RTP
    // version number must be 2, and the padding bit must be zero on
    // all except the last packet.
    do {
        if (p == 1) {
            ...error: padding before last packet in compound
        }
        if (r-> p) {
            p = 1;
        }
```

Participant Database

Each application in an RTP session will maintain a database of information about the participants and about the session itself. The session information, from which the RTCP timing is derived, can be stored as a set of variables:

- The RTP bandwidth—that is, the typical session bandwidth, configured when the application starts.
- The RTCP bandwidth fraction—that is, the percentage of the RTP bandwidth devoted to RTCP reports. This is usually 5%, but profiles may define a means of changing this (0% also may be used, meaning that RTCP is not sent).
- The average size of all RTCP packets sent and received by this participant.
- The number of members in the session, the number of members when this participant last sent an RTCP packet, and the fraction of those who have sent RTP data packets during the preceding reporting interval.
- The time at which the implementation last sent an RTCP packet, and the next scheduled transmission time.
- A flag indicating whether the implementation has sent any RTP data packets since sending the last two RTCP packets.
- A flag indicating whether the implementation has sent any RTCP packets at all.

In addition, the implementation needs to maintain variables to include in RTCP SR packets:

- The number of packets and octets of RTP data it has sent.
- The last sequence number it used.
- The correspondence between the RTP clock it is using and an NTP-format timestamp.

A session data structure containing these variables is also a good place to store the SSRC being used, the SDES information for the implementation, and the file descriptors for the RTP and RTCP sockets. Finally, the session data structure should contain a database for information held on each participant.

In terms of implementation, the session data can be stored simply: a single structure in a C-based implementation, a class in an object-oriented system. With the exception of the participant-specific data, each variable in the structure or class is a simple type: integer, text string, and so on. The format of the participant-specific data is described next.

To generate RTCP packets properly, each participant also needs to maintain state for the other members in the session. A good design makes the participant database an integral part of the operation of the system, holding not just RTCP-related information, but all state for each participant. The per-participant data structure may include the following:

- SSRC identifier.

Timing Rules

The rate at which each participant sends RTCP packets is not fixed but varies according to the size of the session and the format of the media stream. The aim is to restrict the total amount of RTCP traffic to a fixed fraction—usually 5%—of the session bandwidth. This goal is achieved by a reduction in the rate at which each participant sends RTCP packets as the size of the session increases. In a two-party telephone call using RTP, each participant will send an RTCP report every few seconds; in a session with thousands of participants—for example, an Internet radio station—the interval between RTCP reports from each listener may be many minutes.

Each participant decides when to send RTCP packets on the basis of the set of rules described later in this section. It is important to follow these rules closely, especially for implementations that may be used in large sessions. If implemented correctly, RTCP will scale to sessions with many thousands of members. If not, the amount of control traffic will grow linearly with the number of members and will cause significant network congestion.

Reporting Interval

Compound RTCP packets are sent periodically, according to a randomized timer. The average time each participant waits between sending RTCP packets is known as the reporting interval. It is calculated on the basis of several factors:

- The bandwidth allocated to RTCP. This is a fixed fraction—usually 5%—of the session bandwidth. The session bandwidth is the expected data rate for the session; typically this is the bit rate of a single stream of audio or video data, multiplied by the typical number of simultaneous senders. The session bandwidth is fixed for the duration of a session, and supplied as a configuration parameter to the RTP application when it starts.

  The fraction of the session bandwidth allocated to RTCP can be varied by the RTP profile in use. It is important that all members of a session use the same fraction; otherwise state for some members may be prematurely timed out.

- The average size of RTCP packets sent and received. The average size includes not just the RTCP data, but also the UDP and IP header sizes (that is, add 28 octets per packet for a typical IPv4 implementation).

- The total number of participants and the fraction of those participants who are senders. This requires an implementation to maintain a database of all participants, noting whether they are senders (that is, if RTP data packets or RTCP SR packets have been received from them) or receivers (if only RTCP RR, SDES, or APP packets have been received). The earlier section titled Participant Database explained this in detail.

  To guard against buggy implementations that might send SR packets when they have not sent data, a participant that does listen for data should consider another participant to be a sender only if data packets have been received. An implementation that only sends data and does not listen for others' data (such as a media server) may use RTCP SR packets as an indication of a sender, but it should verify that the packet and byte count fields are nonzero and changing from one SR to the next.

If the number of senders is greater than zero but less than one-quarter of the total number of participants, the reporting interval depends on whether we are sending. If we are sending, the reporting interval is set to the number of senders multiplied by the average size of RTCP packets, divided by 25% of the desired RTCP bandwidth. If we are not sending, the reporting interval is set to the number of receivers multiplied by the average size of RTCP packets, divided by 75% of the desired RTCP bandwidth:

```
If ((senders > 0) and (senders < (25% of total number of participants)) {
    If (we are sending) {
        Interval = average RTCP size * senders / (25% of RTCP bandwidth)
    } else {
        Interval = average RTCP size * receivers / (75% of RTCP bandwidth)
```

Summary

This chapter has described the RTP control protocol, RTCP, in some detail. There are three components:

1. The RTCP packet formats, and the means by which compound packets are generated
2. The participant database as the main data structure for an RTP-based application, and the information that needs to be stored for correct operation of RTCP
3. The rules governing timing of RTCP packets: periodic transmission, adaptation to the size of the session, and reconsideration

We have also briefly discussed security and privacy issues, which are discussed in depth in Chapter 13, Security Considerations, as well as the correct validation of RTCP packets.

The RTP control protocol is an integral part of RTP, used for reception quality reporting, source description, membership control, and lip synchronization. Correct implementation of RTCP can significantly enhance an RTP session: It permits the receiver to lipsync audio and video, identifies the other members of a session, and allows the sender to make an informed choice of error protection scheme to use to achieve optimum quality.

Chapter 6. Media Capture, Playout, and Timing

In this chapter we move on from our discussion of networks and protocols, and talk instead about the design of systems that use RTP. An RTP implementation has multiple aspects, some of which are required for all applications; others are optional depending on the needs of the application. This chapter discusses the most fundamental features necessary for media capture, playout, and timing recovery; later chapters describe ways in which reception quality can be improved or overheads reduced.

We start with a discussion of the behavior of a sender: media capture and compression, generation of RTP packets, and the under-lying media timing model. Then the discussion focuses on the receiver, and the problems of media playout and timing recovery in the face of uncertain delivery conditions. Key to receiver design is the operation of the playout buffer, and much of this chapter is spent on this subject.

It is important to remember that many designs are possible for senders and receivers, with the RTP specification permitting a wide range of implementation choices. The design outlined here is one possible implementation, with a particular set of trade-offs; implementers should use alternative techniques if they are more appropriate to particular scenarios. (Many implementations are described in the literature; for example, McCanne and Jacobson87 outline the structure of a video conferencing system influential in the design of RTP.)

Behavior of a Sender

As noted in Chapter 1, An Introduction to RTP, a sender is responsible for capturing audiovisual data, whether live or from a file, compressing it for transmission, and generating RTP packets. It may also participate in error correction and congestion control by adapting the transmitted media stream in response to receiver feedback. Figure 1.2 in Chapter 1 shows the process.

The sender starts by reading uncompressed media data—audio samples or video frames—into a buffer from which encoded frames are produced. Frames may be encoded in several ways depending on the compression algorithm used, and encoded frames may depend on both earlier and later data. The next section, Media Capture and Compression, describes this process.

Compressed frames are assigned a timestamp and a sequence number, and loaded into RTP packets ready for transmission. If a frame is too large to fit into a single packet, it may be fragmented into several packets for transmission. If a frame is small, several frames may be bundled into a single RTP packet. The section titled Generating RTP Packets later in this chapter describes these functions, both of which are possible only if supported by the payload format. Depending on the error correction scheme in use, a channel coder may be used to generate error correction packets or to reorder frames for interleaving before transmission (Chapters 8 and 9 discuss error concealment and error correction). The sender will generate periodic status reports, in the form of RTCP packets, for the media streams it is generating. It will also receive reception quality feedback from other participants, and it may use that information to adapt its transmission. RTCP was described in Chapter 5, RTP Control Protocol.

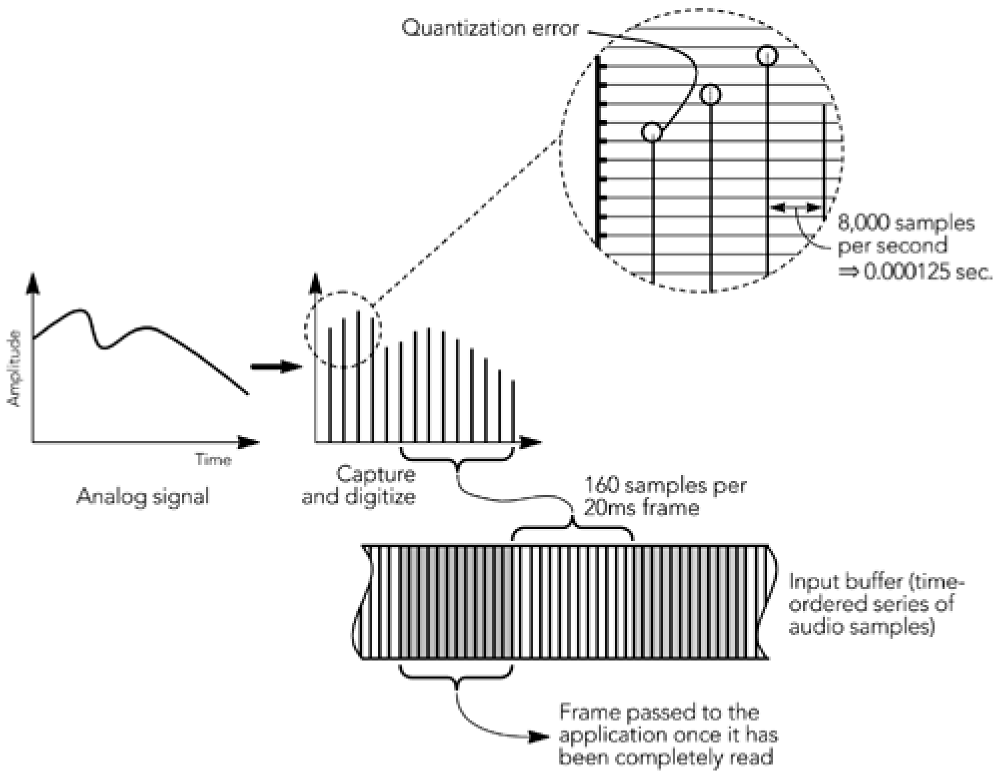
Media Capture and Compression

The media capture process is essentially the same whether audio or video is being transmitted: An uncompressed frame is captured, if necessary it is transformed into a suitable format for compression, and then the encoder is invoked to produce a compressed frame. The compressed frame is then passed to the packetization routine, and one or more RTP packets are generated. Factors specific to audio/video capture are discussed in the next two sections, followed by a description of issues raised by prerecorded content.

Audio Capture and Compression

Considering the specifics of audio capture, Figure 6.1 shows the sampling process on a general-purpose workstation, with sound being captured, digitized, and stored into an audio input buffer. This input buffer is commonly made available to the application after a fixed number of samples have been collected. Most audio capture APIs return data from the input buffer in fixed-duration frames, blocking until sufficient samples have been collected to form a complete frame. This imposes some delay because the first sample in a frame is not made available until the last sample has been collected. If given a choice, applications intended for interactive use should select the buffer size closest to that of the codec frame duration, commonly either 20 milliseconds or 30 milliseconds, to reduce the delay.

Figure 6.1. Audio Capture, Digitization, and Framing



Uncompressed audio frames can be returned from the capture device with a range of sample types and at one of several sampling rates. Common audio capture devices can return samples with 8-, 16-, or 24-bit resolution, using linear, μ-law or A-law quantization, at rates between 8,000 and 96,000 samples per second, and in mono or stereo. Depending on the capabilities of the capture device and on the media codec, it may be necessary to convert the media to an alternative format before the media can be used—for example, changing the sample rate or converting from linear to μ-law quantization. Algorithms for audio format conversion are outside the scope of this book, but standard signal-processing texts give a range of possibilities.

One of the most common audio format conversions is from one sampling rate to another, when the audio capture device samples at one rate, yet the codec requires another rate. (For example, the device may operate at a fixed rate to 44.1kHz to enable high-quality CD playback, yet the desire is to transmit using an 8kHz voice codec.) Sample rate conversion between arbitrary rates is possible but is considerably more efficient and accurate for conversion between rates that are integer multiples of each other. The

Generating RTP Packets

As compressed frames are generated, they are passed to the RTP packetization routine. Each frame has an associated timestamp, from which the RTP timestamp is derived. If the payload format supports fragmentation, large frames are fragmented to fit within the maximum transmission unit of the network (this is typically needed only for video). Finally, one or more RTP packets are generated for each frame, each including media data and any required payload header. The format of the media packet and payload header is defined according to the payload format specification for the codec used. The critical parts to the packet generation process are assigning timestamps to frames, fragmenting large frames, and generating the payload header. These issues are discussed in more detail in the sections that follow.

In addition to the RTP data packets that directly represent the media frames, the sender may generate error correction packets and may reorder frames before transmission. These processes are described in Chapters 8, Error Concealment, and 9, Error Correction. After the RTP packets have been sent, the buffered media data corresponding to those packets is eventually freed. The sender must not discard data that might be needed for error correction or in the encoding process. This requirement may mean that the sender must buffer data for some time after the corresponding packets have been sent, depending on the codec and error correction scheme used.

Timestamps and the RTP Timing Model

The RTP timestamp represents the sampling instant of the first octet of data in the frame. It starts from a random initial value and increments at a media-dependent rate.

During capture of a live media stream, the sampling instant is simply the time when the media is captured from the video frame grabber or audio sampling device. If the audio and video are to be synchronized, care must be taken to ensure that the processing delay in the different capture devices is accounted for, but otherwise the concept is straightforward. For most audio payload formats, the RTP timestamp increment for each frame is equal to the number of samples—not octets—read from the capture device. A common exception is MPEG audio, including MP3, which uses a 90kHz media clock, for compatibility with other MPEG content. For video, the RTP timestamp is incremented by a nominal per frame value for each frame captured, depending on the clock and frame rate. The majority of video formats use a 90kHz clock because that gives integer timestamp increments for common video formats and frame rates. For example, if sending at the NTSC standard rate of (approximately) 29.97 frames per second using a payload format with a 90kHz clock, the RTP timestamp is incremented by exactly 3,003 per packet.

For prerecorded content streamed from a file, the timestamp gives the time of the frame in the playout sequence, plus a constant random offset. As noted in Chapter 4, RTP Data Transfer Protocol, the clock from which the RTP timestamp is derived must increase in a continuous and monotonic fashion irrespective of seek operations or pauses in the presentation. This means that the timestamp does not always correspond to the time offset of the frame from the start of the file; rather it measures the timeline since the start of the playback.

Timestamps are assigned per frame. If a frame is fragmented into multiple RTP packets, each of the packets making up the frame will have the same timestamp.

The RTP specification makes no guarantee as to the resolution, accuracy, or stability of the media clock. The sender is responsible for choosing an appropriate clock, with sufficient accuracy and stability for the chosen application. The receiver knows the nominal clock rate but typically has no other knowledge regarding the precision of the clock. Applications should be robust to variability in the media clock, both at the sender and at the receiver, unless they have specific knowledge to the contrary.

The timestamps in RTP data packets and in RTCP sender reports represent the timing of the media at the sender: the timing of the sampling process, and the relation between the sampling process and a reference clock. A receiver is expected to reconstruct the timing of the media from this information. Note that the RTP timing model says nothing about when the media data is to be played out. The timestamps in data packets give the relative timing, and RTCP sender reports provide a reference for interstream synchronization, but RTP says nothing about the amount of buffering that may be needed at the receiver, or about the decoding time of the packets.

Behavior of a Receiver

As highlighted in Chapter 1, An Introduction to RTP, a receiver is responsible for collecting RTP packets from the network, repairing and correcting for any lost packets, recovering the timing, decompressing the media, and presenting the result to the user. In addition, the receiver is expected to send reception quality reports so that the sender can adapt the transmission to match the network characteristics. The receiver will also typically maintain a database of participants in a session to be able to provide the user with information on the other participants. Figure 1.3 in Chapter 1 shows a block diagram of a receiver.

The first step of the reception process is to collect packets from the network, validate them for correctness, and insert them into a per-sender input queue. This is a straightforward operation, independent of the media format. The next section—Packet Reception—describes this process.

The rest of the receiver processing operates in a sender-specific manner and may be media-specific. Packets are removed from their input queue and passed to an optional channel-coding routine to correct for loss (Chapter 9 describes error correction). Following any channel coder, packets are inserted into a source-specific playout buffer, where they remain until complete frames have been received and any variation in interpacket timing caused by the network has been smoothed. The calculation of the amount of delay to add is one of the most critical aspects in the design of an RTP implementation and is explained in the section titled The Playout Buffer later in this chapter. The section titled Adapting the Playout Point describes a related operation: how to adjust the timing without disrupting playout of the media.

Sometime before their playout time is reached, packets are grouped to form complete frames, damaged or missing frames are repaired (Chapter 8, Error Concealment, describes repair algorithms), and frames are decoded. Finally, the media data is rendered for the user. Depending on the media format and output device, it may be possible to play each stream individually—for example, presenting several video streams, each in its own window. Alternatively, it may be necessary to mix the media from all sources into a single stream for playout—for example, combining several audio sources for playout via a single set of speakers. The final section of this chapter—Decoding, Mixing, and Playout—describes these operations.

The operation of an RTP receiver is a complex process, and more involved than the operation of a sender. This increased complexity is largely due to the variability inherent in IP networks: Most of the complexity comes from the need to compensate for lost packets and to recover the timing of a stream.

Packet Reception

An RTP session comprises both data and control flows, running on distinct ports (usually the data packets flow on an even-numbered port, with control packets on the next higher—odd-numbered—port). This means that a receiving application will open two sockets for each session: one for data, one for control. Because RTP runs above UDP/IP, the sockets used are standard SOCK_DGRAM sockets, as provided by the Berkeley sockets API on UNIX-like systems, and by Winsock on Microsoft platforms.

Once the receiving sockets have been created, the application should prepare to receive packets from the network and store them for further processing. Many applications implement this as a loop, calling select() repeatedly to receive packets—for example:

```
fd_data = create_socket(...);
fd_ctrl = create_socket(...);
while (not_done) {
    FD_ZERO(&rfd);
    FD_SET(fd_data, &rfd);
    FD_SET(fd_ctrl, &rfd);
    timeout = ...;
    if (select(max_fd, &rfd, NULL, NULL, timeout) > 0) {
        if (FD_ISSET(fd_data, &rfd)) {
            ...validate data packet
            ...process data packet
        }
        if (FD_ISSET(fd_ctrl, &rfd)) {
            ...validate control packet
            ...process control packet
        }
    }
    ...do other processing
}
```

Data and control packets are validated for correctness as described in Chapters 4, RTP Data Transfer Protocol, and 5, RTP Control Protocol, and processed as described in the next two sections. The timeout of the select() operation is typically chosen according to the framing interval of the media. For example, a system receiving audio with 20-millisecond packet duration will implement a 20-millisecond timeout, allowing the other processing—such as decoding the received packets—to occur synchronously with arrival and playout, and resulting in an application that loops every 20 milliseconds.

Other implementations may be event driven rather than having an explicit loop, but the basic concept remains: Packets are continually validated and processed as they arrive from the network, and other application processing must be done in parallel to this (either explicitly time-sliced, as shown above, or as a separate thread), with the timing of the application driven by the media processing requirements. Real-time operation is essential to RTP receivers; packets must be processed at the rate they arrive, or reception quality will be impaired.

Receiving Data Packets

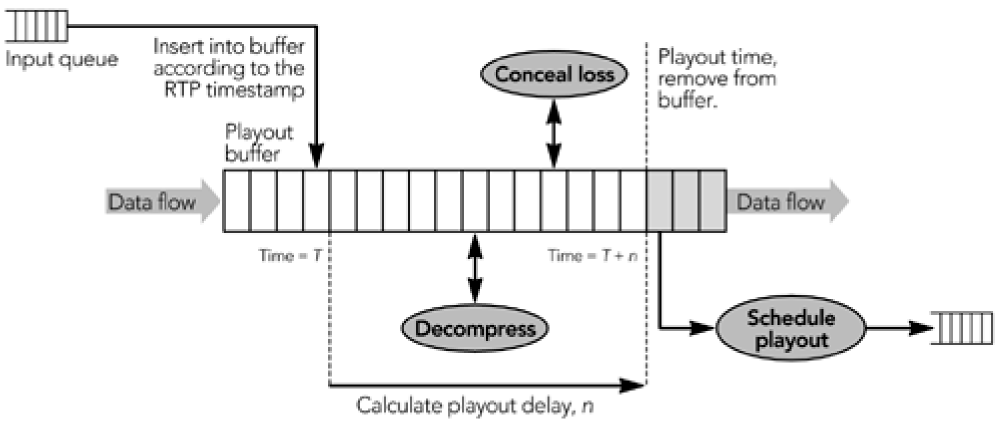
The first stage of the media playout process is to capture RTP data packets from the network, and to buffer those packets for further processing. Because the network is prone to disrupt the interpacket timing, as shown in Figure 6.5, there will be bursts when several packets arrive at once and/or gaps when no packets arrive, and packets may even arrive out of order. The receiver does not know when data packets are going to arrive, so it should be prepared to accept packets in bursts, and in any order.

Figure 6.5. Disruption of Interpacket Timing during Network Transit

The Playout Buffer

Data packets are extracted from their input queue and inserted into a source-specific playout buffer sorted by their RTP timestamps. Frames are held in the playout buffer for a period of time to smooth timing variations caused by the network. Holding the data in a playout buffer also allows the pieces of fragmented frames to be received and grouped, and it allows any error correction data to arrive. The frames are then decompressed, any remaining errors are concealed, and the media is rendered for the user. Figure 6.7 illustrates the process.

Figure 6.7. The Playout Buffer

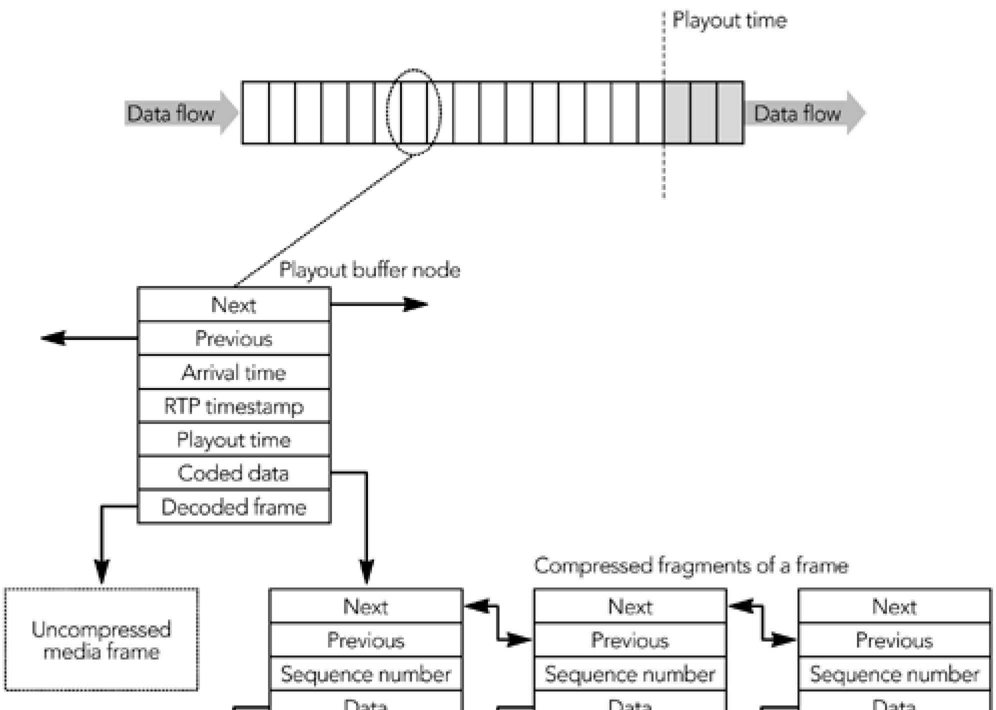


A single buffer may be used to compensate for network timing variability and as a decode buffer for the media codec. It is also possible to separate these functions: using separate buffers for jitter removal and decoding. However, there is no strict layering requirement in RTP: Efficient implementations often mingle related functions across layer boundaries, a concept termed integrated layer processing.65

Basic Operation

The playout buffer comprises a time-ordered linked list of nodes. Each node represents a frame of media data, with associated timing information. The data structure for each node contains pointers to the adjacent nodes, the arrival time, RTP timestamp, and desired playout time for the frame, and pointers to both the compressed fragments of the frame (the data received in RTP packets) and the uncompressed media data. Figure 6.8 illustrates the data structures involved.

Figure 6.8. The Playout Buffer Data Structures



Adapting the Playout Point

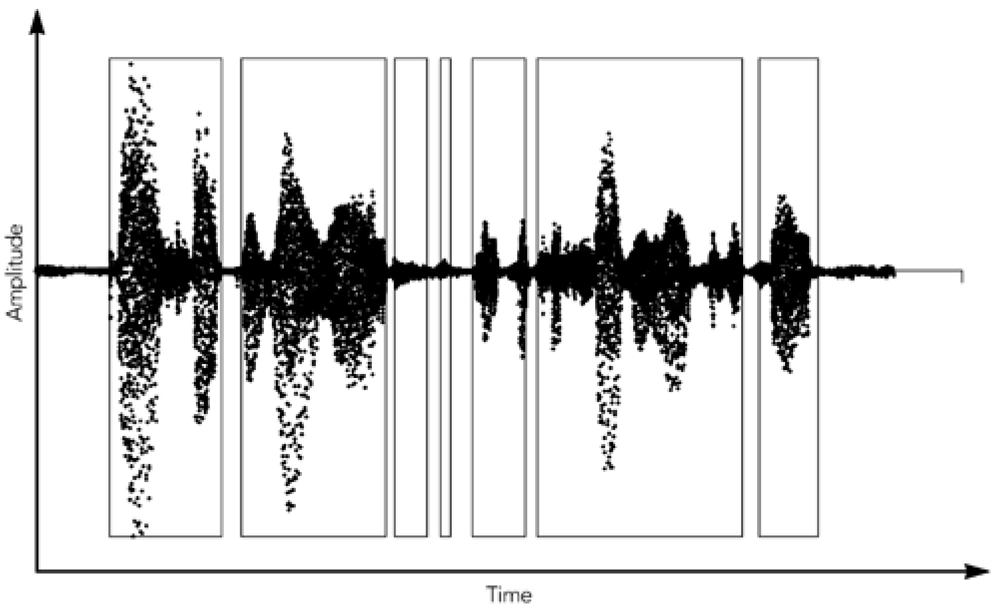
There are two basic approaches to adapting the playout point: receivers can either slightly adjust the playout time for each frame, making continual small adjustments to the playout point, or they can insert or remove complete frames from the media stream, making a smaller number of large adjustments as they become necessary. No matter how the adjustment is made, the media stream is disrupted to some extent. The aim of the adaptation must be to minimize this disruption, which requires knowledge of the media stream; accordingly, audio and video playout adaptation strategies are discussed separately.

Playout Adaptation for Audio with Silence Suppression

Audio is a continuous media format, meaning that each audio frame occupies a certain amount of time, and the next is scheduled to start immediately after it finishes. There are no gaps between frames unless silence suppression is used, and hence there is no convenient time to adapt the playout delay. For this reason the presence of silence suppression has a significant effect on the design of audio playout buffer algorithms.

For conversational speech signals, an active speaker will generate talk spurts several hundred milliseconds in duration, separated by silent periods. Figure 6.16 shows the presence of talk spurts in a speech signal, and the gaps left between them. The sender detects frames representing silent periods and suppresses the RTP packets that would otherwise be generated for those frames. The result is a sequence of packets with consecutive sequence numbers, but a jump in the RTP timestamp depending on the length of the silent period.

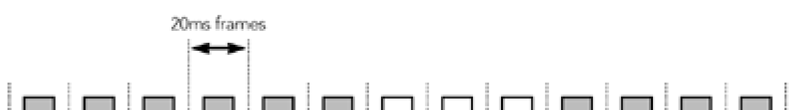
Figure 6.16. Talk Spurts in a Speech Signal



Adjusting the playout point during a talk spurt will cause an audible glitch in the output, but a small change in the length of the silent period between talk spurts will not be noticeable.92 This is the key point to remember in the design of a playout algorithm for an audio tool: If possible, adjust the playout point only during silent periods.

It is usually a simple matter for a receiver to detect the start of a talk spurt, because the sender is required to set the marker bit on the first packet after a silent period, providing an explicit indication of the start of a talk spurt. Sometimes, however, the first packet in a talk spurt is lost. It is usually still possible to detect that a new talk spurt has started, because the sequence number/timestamp relationship will change as shown in Figure 6.17, providing an implicit indication of the start of the talk spurt.

Figure 6.17. Implicit Indication of the Start of a Talk Spurt



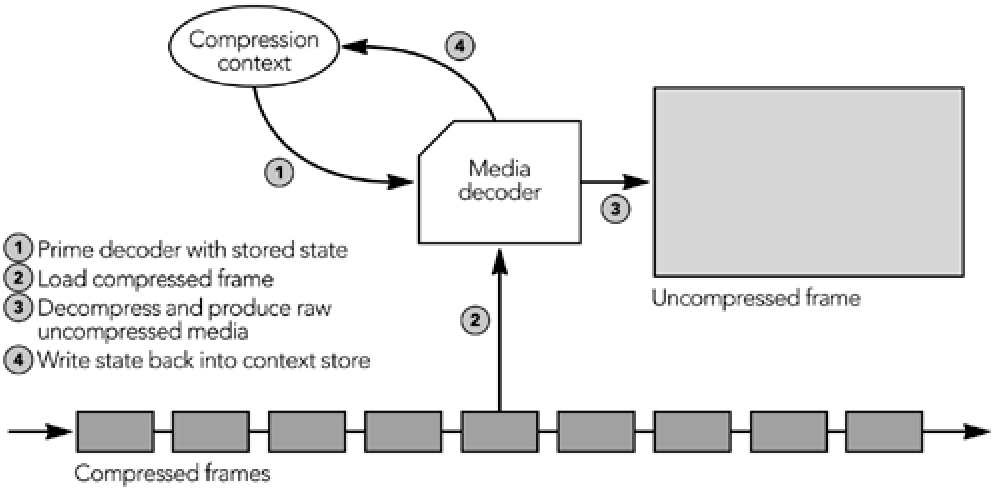
Decoding, Mixing, and Playout

The final stages of the playout process are to decode the compressed media, mix media streams together if there are fewer output channels than active sources, and finally play the media to the user. This section considers each stage in turn.

Decoding

For each active source the application must maintain an instantiation of the media decoder, comprising the decompression routines along with state known as the compression context. The decoder may be an actual hardware device or a software function, depending on the system. It converts each compressed frame into uncompressed media data, on the basis of the data in the frame and the compression context. As each frame is decoded, the compression context for the source is updated as shown in Figure 6.19.

Figure 6.19. Operation of a Media Decoder



The presence of accurate state in the decompression context is fundamental to correct operation of the decoder, and codecs will produce incorrect results if the context is missing or damaged. This is most often an issue if some data packets are lost because there will be a frame that cannot be decoded. The result will be a gap in the playout where that frame should have been, but the decompression context will also be invalidated and the following frames will be corrupted.

Depending on the codec, it may be possible to feed it an indication that a frame has been lost, allowing the decoder to better repair the context and reduce the damage to the media stream (for example, many speech codecs have the notion of erasure frames to signal losses). Otherwise the receiver should try to repair the context and conceal the effects of the loss, as discussed in Chapter 8, Error Concealment. Many loss concealment algorithms operate on the uncompressed media data, after decoding and before mixing and playout operation.

Audio Mixing

Mixing is the process of combining multiple media streams into one, for output. This is primarily an issue for audio applications because most systems have only a single set of speakers but multiple active sources—for example, in a multiparty teleconference. Once audio streams have been decoded, they must be mixed together before being written to the audio device. The final stages of an audio tool will typically be structured somewhat as shown in Figure 6.20. The decoder produces uncompressed audio data on a per-source basis, written into a per-source playout buffer, and the mixer combines the results into a single buffer for playout (these steps can, of course, be combined into one if the decoder understands the mixing process). Mixing can occur at any time after the media has been decoded, and before it is due for playout.

Figure 6.20. Audio Mixing

Summary

This chapter has described the fundamental behavior of RTP senders and receivers in some detail, focusing especially on the design of the receiver's playout buffer. The playout buffer design is relatively simple for streaming applications, which can accept several hundred milliseconds (or perhaps even seconds) of delay. For applications designed for interactive use, however, the play-out buffer is critical to achieving good performance. These applications need low latency, and hence playout buffer delays of a few tens of milliseconds, taxing the design of algorithms that must balance the needs of low delay with the need to avoid discarding late packets if possible.

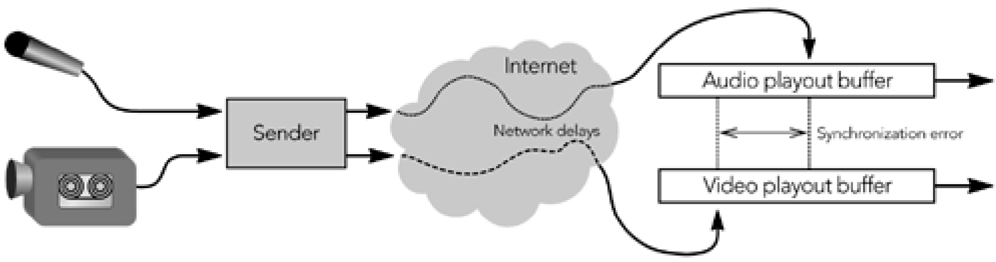
RTP systems place a lot of intelligence in the end systems, leaving them to compensate for the variability inherent in a best-effort packet network. Recognizing this is the key to good performance: A well-designed, robust implementation can perform significantly better than a naive design.

Chapter 7. Lip Synchronization

- Sender Behavior
- Receiver Behavior
- Synchronization Accuracy

A multimedia session comprises several media streams, and in RTP each is transported via a separate RTP session. Because the delays associated with different encoding formats vary greatly, and because the streams are transported separately across the network, the media will tend to have different playout times. To present multiple media in a synchronized fashion, receivers must realign the streams as shown in Figure 7.1. This chapter describes how RTP provides the information needed to facilitate the synchronization of multiple media streams. The typical use for this technique is to align audio and video streams to provide lip synchronization, although the techniques described may be applied to the synchronization of any set of media streams.

Figure 7.1. Media Flows and the Need for Synchronization



A common question is why media streams are delivered separately, forcing the receiver to resynchronize them, when they could be delivered bundled together and presynchronized. The reasons include the desire to treat audio and video separately in the network, and the heterogeneity of networks, codecs, and application requirements.

It is often appropriate to treat audio and video differently at the transport level to reflect the preferences of the sender or receiver. In a video conference, for instance, the participants often favor audio over video. In a best-effort network, this preference may be reflected in differing amounts of error correction applied to each stream; in an integrated services network, using RSVP (Resource ReSerVation Protocol),11 this could correspond to reservations with differing quality-of-service (QoS) guarantees for audio and video; and in a Differentiated Services network,23,24 the audio and video could be assigned to different priority classes. If the different media types were bundled together, these options would either cease to exist or become considerably harder to implement. Similarly, if bundled transport were used, all receivers would have to receive all media; it would not be possible for some participants to receive only the audio, while others received both audio and video. This ability becomes an issue for multiparty sessions, especially those using multicast distribution.

However, even if it is appropriate to use identical QoS for all media, and even if all receivers want to receive all media, the properties of codecs and playout algorithms are such that some type of synchronization step is usually required. For example, audio and video decoders take different and varying amounts of time to decompress the media, perform error correction, and render the data for presentation. Also the means by which the playout buffering delay is adjusted varies with the media format, as we learned in Chapter 6, Media Capture, Playout, and Timing. Each of these processes can affect the playout time and can result in loss of synchronization between audio and video.

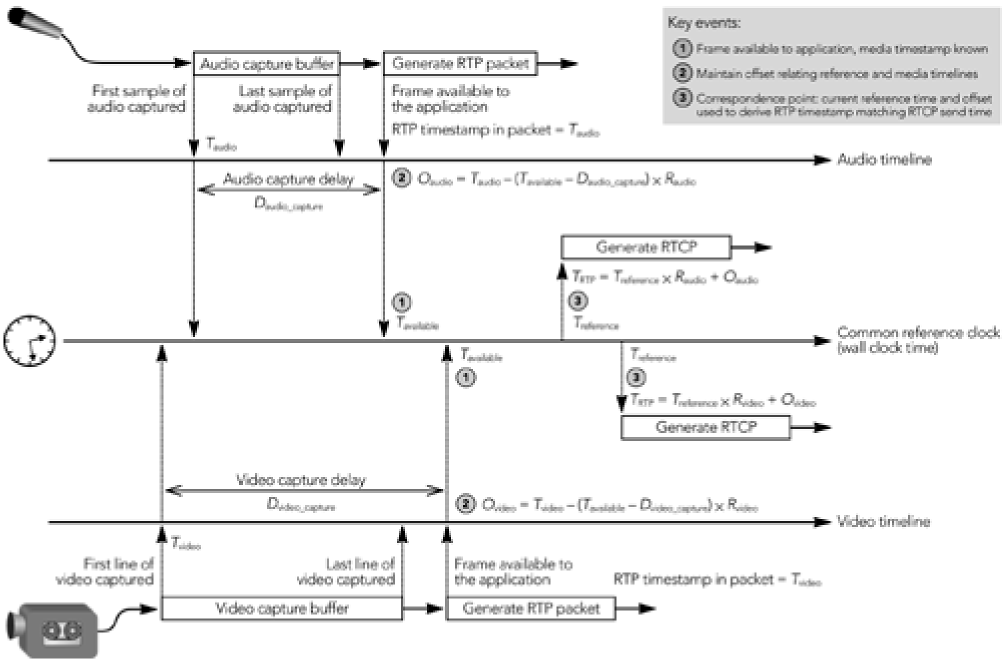
The result is that some type of synchronization function is needed, even if the media are bundled for delivery. As a result, we may as well deliver media separately, allowing them to be treated differently in the network because doing so does not add significant complexity to the receiver.

With these issues in mind, we now turn to a discussion of the synchronization process. There are two parts to this

Sender Behavior

The sender enables synchronization of media streams at the receiver by running a common reference clock and periodically announcing, through RTCP, the relationship between the reference clock time and the media stream time, as well as the identities of the streams to be synchronized. The reference clock runs at a fixed rate; correspondence points between the reference clock and the media stream allow the receiver to work out the relative timing relationship between the media streams. This process is shown in Figure 7.2.

Figure 7.2. Mapping Media Time Lines to a Common Clock at the Sender



The correspondence between reference clock and media clock is noted when each RTCP packet is generated: A sampling of the reference clock, Treference, is included in the packet along with a calculated RTP timestamp, TRTP = Treference x Raudio + Oaudio. The multiplication must be made modulo 232, to restrict the result to the range of the 32-bit RTP timestamp. The offset is calculated as Oaudio = Taudio – (Tavailable – Daudio_capture) x Raudio, being the conversion factor between media and reference timelines. Operating system latencies can delay Tavailable and cause variation in the offset, which should be filtered by the application to choose a minimum value. (The obvious changes to the formulae are made in the case of video.)

Each application on the sender that is transmitting RTP streams needs access to the common reference clock, Treference, and must identify its media with reference to a canonical source identifier. The sending applications should be aware of the media capture delay—for example, Daudio_capture—because it can be significant and should be taken into account in the calculation and announcement of the relationship between reference clock times and media clock times.

The common reference clock is the "wall clock" time used by RTCP. It takes the form of an NTP-format timestamp, counting seconds and fractions of a second since midnight UTC (Coordinated Universal Time) on January 1, 1900.5 (Senders that have no knowledge of the wall clock time may use a system-specific clock such as "system uptime" to calculate NTP-format timestamps as an alternative; the choice of a reference clock does not affect synchronization, as long as it is done consistently for all media.) Senders periodically establish a correspondence between the media clock for each stream and the common reference clock; this is communicated to receivers via RTCP sender report packets as described in the section titled RTCP SR: Sender Reports in Chapter 5, RTP Control Protocol.

In typical scenarios, there is no requirement for the sender or receiver to be synchronized to an external clock. In particular, although the wall clock time in RTCP sender report packets uses the format of an NTP timestamp, it is not required to be synchronized to an NTP time source. Sender and receiver clocks never have to be synchronized to each other. Receivers do not care about the absolute value of the NTP format timestamp in RTCP sender report packets, only that the clock is common between media, and of sufficient accuracy and stability to allow synchronization.

Receiver Behavior

A receiver is expected to determine which media streams should be synchronized, and to align their presentation, on the basis of the information conveyed to it in RTCP packets.

The first part of the process—determining which streams are to be synchronized—is straightforward. The receiver synchronizes those streams that the sender has given the same CNAME in their RTCP source description packets, as described in the previous section. Because RTCP packets are sent every few seconds, there may be a delay between receipt of the first data packet and receipt of the RTCP packet that indicates that particular streams are to be synchronized. The receiver can play out the media data during this time, but it is unable to synchronize them because it doesn't have the required information.

More complex is the actual synchronization operation, in which the receiver time-aligns audio and video for presentation. This operation is triggered by the reception of RTCP sender report packets containing the mapping between the media clock and a reference clock common to both media. Once this mapping has been determined for both audio and video streams, the receiver has the information needed to synchronize playout.

The first step of lip synchronization is to determine, for each stream to be synchronized, when the media data corresponding to a particular reference time is to be presented to the user. Because of differences in the network behavior or other reasons, it is likely that data from two streams that was captured at the same instant will not be scheduled for presentation at the same time if the playout times are determined independently according to the methods described in Chapter 6, Media Capture, Playout, and Timing. The playout time for one stream therefore has to be adjusted to match the other. This adjustment translates into an offset to be added to the playout buffering delay for one stream, such that the media are played out in time alignment. Figures 7.4 and 7.5 illustrate the process.

Figure 7.4. Lip Synchronization at the Receiver

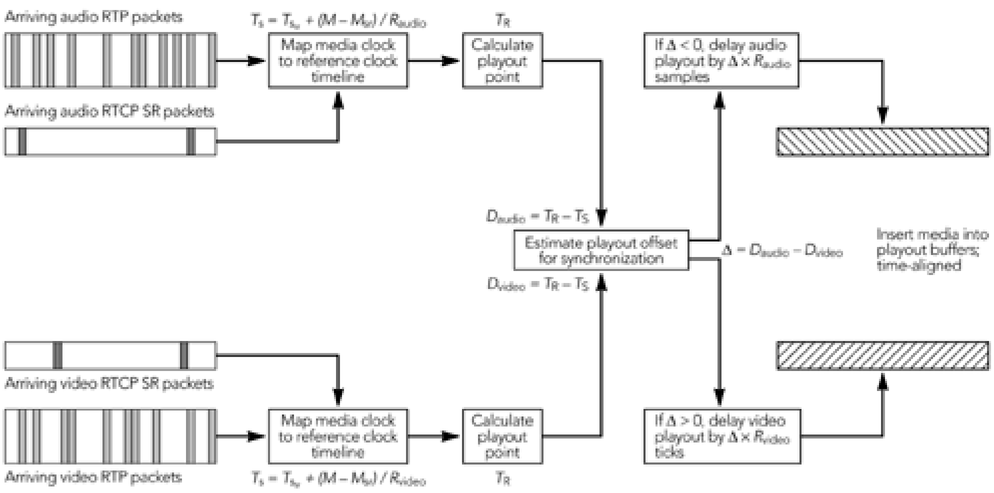
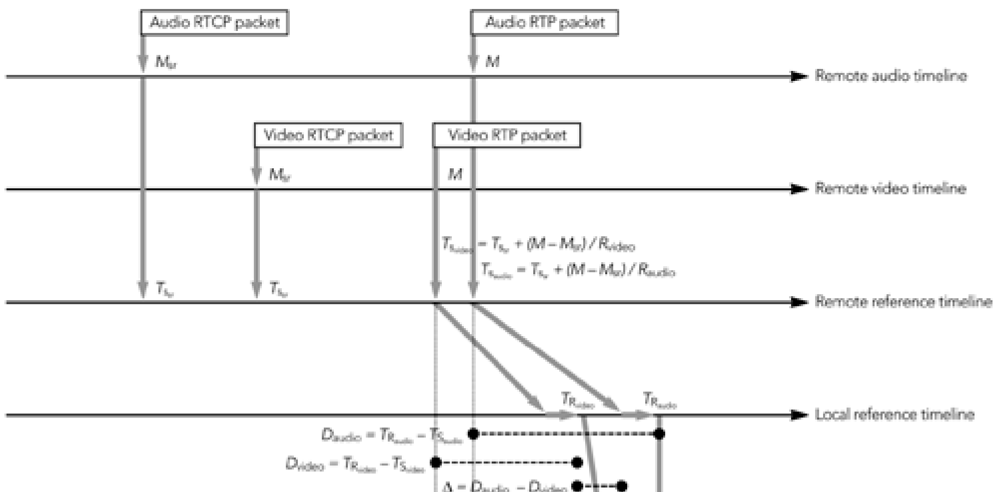


Figure 7.5. Mapping between Timelines to Achieve Lip Synchronization at the Receiver



Synchronization Accuracy

When you're implementing synchronization, the question of accuracy arises: What is the acceptable offset between streams that are supposed to be synchronized? There is no simple answer, unfortunately, because human perception of synchronization depends on what is being synchronized and on the task being performed. For example, the requirements for lip synchronization between audio and video are relatively lax and vary with the video quality and frame rate, whereas the requirements for synchronization of multiple related audio streams are strict.

If the goal is to synchronize a single audio track to a single video track, then synchronization accurate to a few tens of milliseconds is typically sufficient. Experiments with video conferencing suggest that synchronization errors on the order of 80 milliseconds to 100 milliseconds are below the limit of human perception (see Kouvelas et al. 1996,84 for example), although this clearly depends on the task being performed and on the picture quality and frame rate. Higher-quality pictures, and video with higher frame rates, make lack of synchronization more noticeable because it is easier to see lip motion. Similarly, if frame rates are low—less than approximately five frames per second—there is little need for lip synchronization because lip motion cannot be perceived as speech, although other visual cues may expose the lack of synchronization.

If the application is synchronizing several audio tracks—for example, the channels in a surround-sound presentation—requirements for synchronization are much stricter. In this scenario, playout must be accurate to a single sample; the slightest synchronization error will be noticeable because of the phase difference between the signals, which disrupts the apparent position of the source.

The information provided by RTP is sufficient for sample-accurate synchronization, if desired, provided that the sender has correctly calculated the mapping between media time and reference time contained in RTCP sender report packets, and provided that the receiver has an appropriate synchronization algorithm. Whether this is achievable in practice is a quality-of-implementation issue.

Summary

Synchronization—like many other features of RTP—is performed by the end systems, which must correct for the variability inherent in a best-effort packet network. This chapter has described how senders signal the time alignment of media, and the process by which receivers can resynchronize media streams. It has also discussed the synchronization accuracy required for lip synchronization between audio and video, and for synchronization among multiple audio streams.

Part III: Robustness

Chapter 8. Error Concealment

Earlier chapters described how RTP running over UDP/IP provides an unreliable packet delivery service, and how this means an application may have to deal with an incomplete media stream. There are two things the application can do when packet loss occurs: try to correct the error, or try to conceal it. Error correction is discussed in Chapter 9. In this chapter we discuss techniques by which a receiver can conceal the effects of loss.

Techniques for Audio Loss Concealment

When an RTP packet containing audio data—whether music or speech—is lost, the receiver has to generate a replacement to preserve the timing of the media stream. This can be done in many ways, and the choice of concealment algorithm can have a significant impact on the perceived quality of the system in the case of loss.

Measuring Audio Quality

Human perception of sound is a complex process, and the perceptual significance of distortion depends not just on the amount the signal has changed, but also on the type of damage caused, and where it occurred in the signal. Some types of distortion are more noticeable to listeners than others, even if—by some objective measure—they change the signal by the same amount. It is also common for different listeners to perceive a particular type of distortion in different ways, and to rate concealment schemes differently depending on the material to which they are applied.

This makes it very difficult to devise objective quality measurements for different repair schemes. It is not sufficient to measure the difference between the original waveform from the source and the waveform recovered at the receiver, because the perceived quality has no direct relation to the differences in the waveforms. Simple measures, such as the signal-to-noise ratio, are effectively useless. More complex schemes (for example, those in ITU recommendations P.861 and P.862[63,64]) give results that are approximately correct for speech, but even these are not 100% reliable.

When objective measurements fail, we need to resort to subjective tests. By conducting listening tests with a wide range of subjects, materials, and error conditions, we can measure the effectiveness of different repair schemes in a manner meaningful to the listener. These tests involve playing different types of music, or a range of words, phrases, and sentences subject to different error conditions and concealment techniques, with the listener rating their quality and/or intelligibility according to a particular scale.

The choices of material and rating scale depend on what is being measured. If you are attempting to measure the perceived quality of speech ("does it sound good?"), then rating samples on the basis of a Mean Opinion Score (MOS) is appropriate. The MOS is a five-point rating scale, the results of which are converted into numeric form (excellent = 5, good = 4, fair = 3, poor = 2, bad = 1) and averaged across all the listeners, giving a numeric result between 1 and 5. For the results to be statistically valid, it is necessary to make a large number of tests comparing different samples.

Typical MOS scores for unimpaired speech are 4.2 for the G.711 codec (that is, standard telephone quality) and between 3.5 and 4 for mobile telephony (for example, GSM, QCELP). Packet loss will lower these numbers, with the degree of loss and the type of concealment determining the actual result.

MOS scores provide a reasonable measure of perceived quality, allowing comparison between different codecs and repair techniques, but they do not measure intelligibility (that is, whether the audio is understandable). There is a difference between what sounds good and what conveys information; it is possible to define a concealment scheme that gets very good marks for sound quality but may not produce intelligible speech. In tests for intelligibility, listeners copy down sentences or words played with different impairments, or answer questions on a passage of text, and the result is a measure of how many errors are made. Again, a large number of tests must be conducted for the results to be statistically meaningful.

Perhaps the most important point learned from listening tests is that the results vary depending on the persons listening, the material they are listening to, the type of distortion present in the sound, and the task they are performing. Depending on the application, it may be important to conduct tests of both perceived quality and intelligibility, and it is always necessary to ensure that the test material and packet loss rates match those of typical usage.

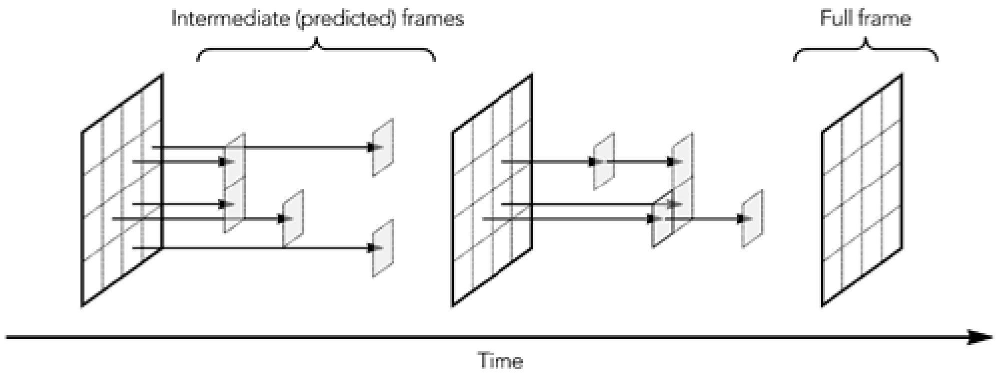
Silence Substitution

The simplest possible repair technique is silence substitution, in which gaps caused by packet loss are filled with silence of the appropriate duration, as shown in Figure 8.1. This is the cheapest and easiest method to implement, and one of the most commonly used techniques.

Techniques for Video Loss Concealment

Most video codecs use interframe compression, sending occasional full frames and many intermediate frames as updates to the parts of the frame that have changed or moved, as shown in Figure 8.8. This technique, known as predictive coding because each frame is predicted on the basis of the preceding frame, is essential for good compression.

Figure 8.8. Basic Operation of Video Codecs



Predictive coding has several consequences for loss concealment. The first is that loss of an intermediate frame may affect only part of a frame, rather than the whole frame (similar effects occur when a frame is split across multiple packets, some of which are lost). For this reason, concealment algorithms must be able to repair damaged regions of an image, as well as replace an entire lost image. A common way of doing this is through motion-compensated repetition.

The other consequence of predictive coding is that frames are no longer independent. This means that loss of data in one frame may affect future frames, making loss concealment more difficult. This problem is discussed in the section titled Dependency Reduction later in this chapter, along with possible solutions.

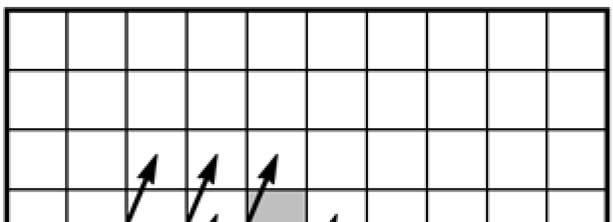
Motion-Compensated Repetition

One of the widely used techniques for video loss concealment is repetition in the time domain. When loss occurs, the part of the frame affected by the loss is replaced with a repeat of the preceding frame. Because most video codecs are block based, and the missing data will often constitute only a small part of the image, this type of repair is usually acceptable.

Of course, repetition works only if the image is relatively constant. If there is significant motion between frames, repeating a portion of the preceding frame will give noticeable visual artifacts. If possible, it is desirable to detect the motion and try to compensate for it when concealing the effects of loss. In many cases this is easier than might be imagined because common video codecs allow the sender to use motion vectors to describe changes in the image, rather than sending a new copy of moved blocks.

If only a single block of the image is lost, a receiver may use the motion vectors associated with the surrounding blocks to infer the correct position for the missing block, on the basis of the preceding packet. For example, Figure 8.9 shows how the motion of a single missing block of the image can be inferred. If the highlighted block is lost, the original position can be derived because the motion is likely the same as that for the surrounding blocks.

Figure 8.9. Motion-Compensated Repetition of a Missing Video Block

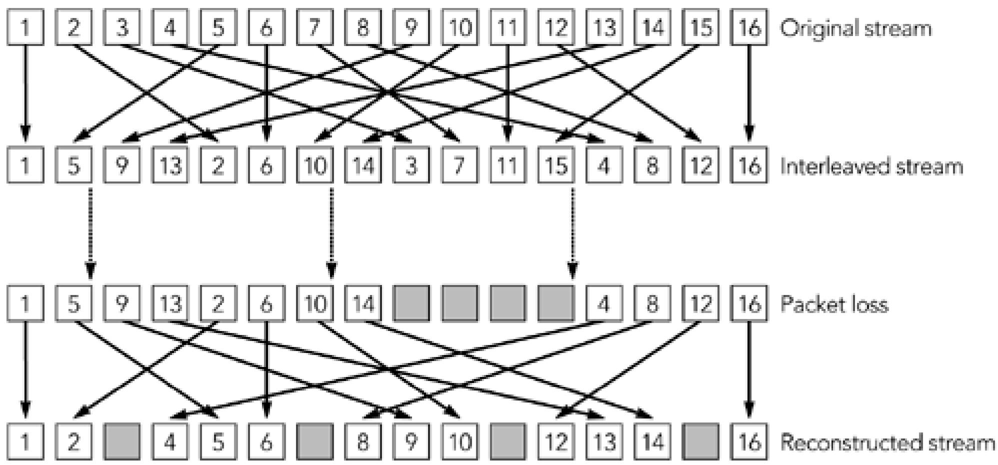


Interleaving

At the start of this chapter, it was noted that error concealment is something done by a receiver, without help from the sender. In general this is the case, but sometimes a sender can ease the task of error concealment without having to send extra information. One such example was noted for video, in which a sender can reduce the interframe dependency to ease the job of a receiver. A more general-purpose technique is interleaving, which can be used with both audio and video streams, as long as low delay is not a requirement.

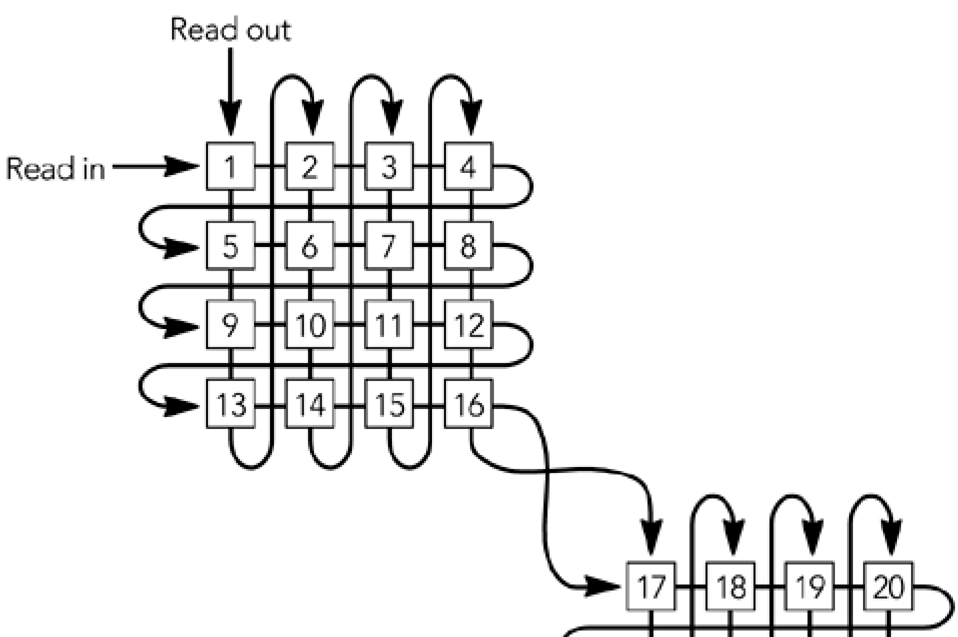
The interleaving process reorders data before transmission so that originally adjacent data is separated by a guaranteed distance during transport. Interleaving is useful because it makes bursts of consecutive packet loss in the transport stream appear as isolated losses when the original order is restored. In Figure 8.11, for example, the loss of four consecutive packets in the interleaved stream is transformed into four single-packet losses when the original order is reconstructed. The actual loss rate is unchanged, but it is typically easier for a receiver to conceal a series of single-packet losses than it is to conceal a longer burst of loss.

Figure 8.11. Interleaving, Transforming Burst Loss to Isolated Loss (From C. Perkins, O. Hodson, and V. Hardman, "A Survey of Packet Loss Recovery Techniques for Streaming Media," IEEE Network Magazine, September/October 1998. © 1998 IEEE.)



The simplest implementation of an interleaving function is with a pair of matrices, as shown in Figure 8.12. Frames of media data are read into the first matrix by rows until that matrix is full. At that time the two matrices are switched, with data being read out of the first by columns, as the second is filled by rows. The process continues, with frames being read into one matrix as they are read out of the other.

Figure 8.12. An Interleaving Matrix



Summary

One of the key points to remember when designing an RTP application is robustness. Error concealment is a major part of this, allowing the application to operate even when the network misbehaves. A good error concealment scheme provides the difference between a tool that can be used in the real world and one that continually fails when subjected to the wide variation in loss rates inherent in the Internet. For this reason, all applications should implement some form of error concealment.

Chapter 9. Error Correction

Although it is clearly important to be able to conceal the effects of transmission errors, it is better if those errors can be avoided or corrected. This chapter presents techniques that the sender can use to help receivers recover from packet loss and other transmission errors.

The techniques used to correct transmission errors fall into two basic categories: forward error correction and retransmission.[80] Forward error correction relies on additional data added by the sender to a media stream, which receivers can then use to correct errors with a certain probability. Retransmission, on the other hand, relies on explicit requests for additional copies of particular packets.

The choice between retransmission and forward error correction depends on the application and on the network characteristics. The details and trade-offs of the different approaches are discussed in more detail in this chapter.

Forward Error Correction

Forward error correction (FEC) algorithms transform a bit stream to make it robust for transmission. The transformation generates a larger bit stream intended for transmission across a lossy medium or network. The additional information in the transformed bit stream allows receivers to exactly reconstruct the original bit stream in the presence of transmission errors. Forward error correction algorithms are notably employed in digital broadcasting systems, such as mobile telephony and space communication systems, and in storage systems, such as compact discs, computer hard disks, and memory. Because the Internet is a lossy medium, and because media applications are sensitive to loss, FEC schemes have been proposed and standardized for RTP applications. These schemes offer both exact and approximate reconstruction of the bit stream, depending on the amount and type of FEC used, and on the nature of the loss.

When an RTP sender uses FEC, it must decide on the amount of FEC to add, on the basis of the loss characteristics of the network. One way of doing this is to look at the RTCP receiver report packets it is getting back, and use the loss fraction statistics to decide on the amount of redundant data to include with the media stream.

In theory, by varying the encoding of the media, it is possible to guarantee that a certain fraction of losses can be corrected. In practice, several factors indicate that FEC can provide only probabilistic repair. Key among those is the fact that adding FEC increases the bandwidth of a stream. This increase in bandwidth limits the amount of FEC that can be added on the basis of the available network capacity, and it may also have adverse effects if loss is caused by congestion. In particular, adding bandwidth to the stream may increase congestion, worsening the loss that the FEC was supposed to correct. This issue is discussed further in the section titled At the Sender, under Implementation Considerations, later in this chapter, as well as in Chapter 10, Congestion Control.

Note that although the amount of FEC can be varied in response to reception quality reports, there is typically no feedback about individual packet loss events, and no guarantee that all losses are corrected. The aim is to reduce the residual loss rate to something acceptable, then to let error concealment take care of any remaining loss.

If FEC is to work properly, the loss rate must be bounded, and losses must occur in particular patterns. For example, it is clear that an FEC scheme designed to correct 5% loss will not correct all losses if 10% of packets are missing. Less obviously, it might be able to correct 5% loss only if the losses are of nonconsecutive packets.

The key advantage of FEC is that it can scale to very large groups, or groups where no feedback is possible.54 The amount of redundant data added depends on the average loss rate and on the loss pattern, both of which are independent of the number of receivers. The disadvantage is that the amount of FEC added depends on the average loss rate. A receiver with below-average loss will receive redundant data, which wastes capacity and must be discarded. One with above-average loss will be unable to correct all the errors and will have to rely on concealment. If the loss rates for different receivers are very heterogeneous, it will not be possible to satisfy them all with a single FEC stream (layered coding may help; see Chapter 10, Congestion Control).

Another disadvantage is that FEC may add delay because repair cannot happen until the FEC packets arrive. If FEC packets are sent a long time after the data they protect, then a receiver may have to choose between playing damaged data quickly or waiting for the FEC to arrive and potentially increasing the end-to-end delay. This is primarily an issue with interactive applications, in which it is important to have low delay.

Many FEC schemes exist, and several have been adopted as part of the RTP framework. We will first review some techniques that operate independently of the media format—parity FEC and Reed–Solomon encoding—before studying those specific to particular audio and video formats.

Parity FEC

One of the simplest error detection/correction codes is the parity code. The parity operation can be described mathematically as an exclusive-or (XOR) of the bit stream. The XOR operation is a bitwise logic operation, defined for two inputs in this way:

```
0 XOR 0 = 0
```

Channel Coding

Forward error correction, which relies on the addition of information to the media stream to provide protection against packet loss, is one form of channel coding. The media stream can be matched to the loss characteristics of a particular network path in other ways as well, some of which are discussed in the following sections.

Partial Checksum

Most packet loss in the public Internet is caused by congestion in the network. However, as noted in Chapter 2, Voice and Video Communication over Packet Networks, in some classes of network—for example, wireless—noncongestive loss and packet corruption are common. Although discarding packets with corrupted bits is appropriate in many cases, some RTP payload formats can make use of corrupted data (for example, the AMR audio codecs41). You can make use of partially corrupt RTP packets either by disabling the UDP checksum (if IPv4 is used) or by using a transport with a partial checksum.

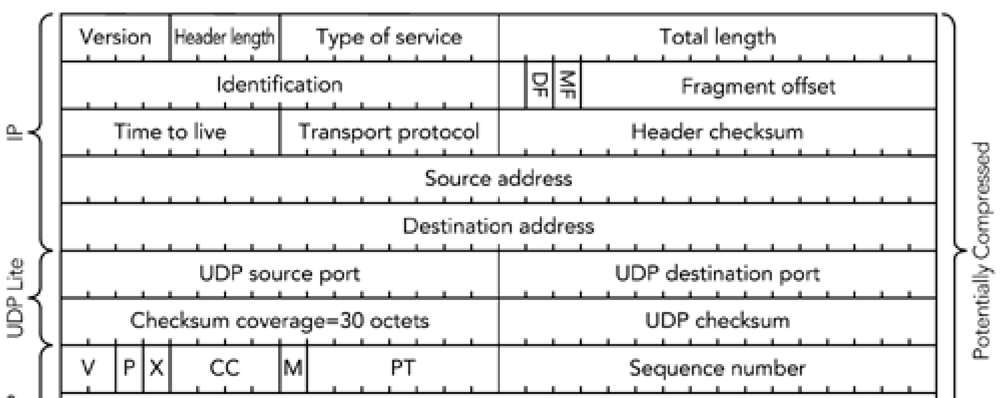
When using RTP with a standard UDP/IPv4 stack, it is possible to disable the UDP checksum entirely (for example, using sysctlnet.inet.udp.checksum=0 on UNIX machines supporting sysctl, or using the UDP_NOCHECKSUM socket option with Winsock2). Disabling the UDP checksum has the advantage that packets with corrupted payload data are delivered to the application, allowing some part of the data to be salvaged. The disadvantage is that the packet header may be corrupted, resulting in packets being misdirected or otherwise made unusable.

Note that some platforms do not allow UDP checksums to be disabled, and others allow it as a global setting but not on a per-stream basis. In IPv6-based implementations, the UDP checksum is mandatory and must not be disabled (although the UDP Lite proposal may be used).

A better approach is to use a transport with a partial checksum, such as UDP Lite.53 This is a work in progress that extends UDP to allow the checksum to cover only part of the packet, rather than all or none of it. For example, the checksum could cover just the RTP/UDP/IP headers, or the headers and the first part of the payload. With a partial checksum, the transport can discard packets in which the headers—or other important parts of the payload—are corrupted, yet pass those that have errors only in the unimportant parts of the payload.

The first RTP payload format to make significant use of partial checksum was the AMR audio codec.41 This is the codec selected for many third-generation cellular telephony systems, and hence the designers of its RTP payload format placed high priority on robustness to bit errors. Each frame of the codec bit stream is split into class A bits, which are vital for decoding, and class B and C bits, which improve quality if they are received, but are not vital. One or more frames of AMR output are placed into each RTP packet, with the option of using a partial checksum that covers the RTP/UDP/IP headers and class A bits, while the other bits are left unprotected. This lack of protection allows an application to ignore errors in the class B and class C bits, rather than discarding the packets. In Figure 9.9, for example, the shaded bits are not protected by a checksum. This approach appears to offer little advantage, because there are relatively few unprotected bits, but when header compression (see Chapter 11) is used, the IP/UDP/RTP headers and checksum are reduced to only four octets, increasing the gain due to the partial checksum.

Figure 9.9. An Example of the Use of Partial Checksums in the AMR Payload Format



Retransmission

Losses may also be recovered if the receivers send feedback to the sender, asking it to retransmit packets lost in transit. Retransmission is a natural approach to error correction, and it works well in some scenarios. It is, however, not without problems that can limit its applicability. Retransmission is not a part of standard RTP; however, an RTP profile is under development44 that provides an RTCP-based framework for retransmission requests and other immediate feedback.

RTCP as a Framework for Retransmission

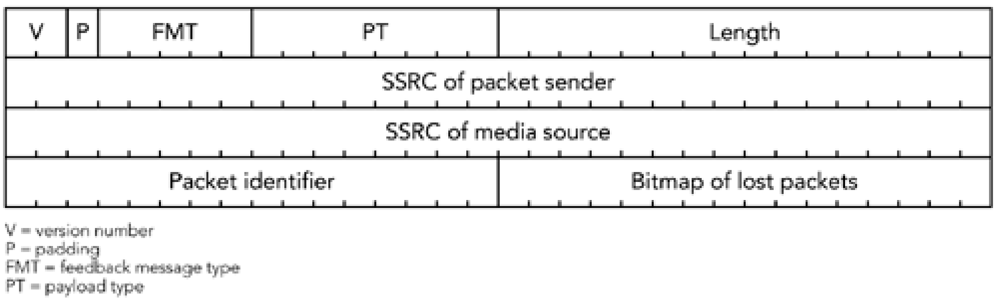
Because RTP includes a feedback channel—RTCP—for reception reports and other data, it is natural to use that channel for retransmission requests too. Two steps are required: Packet formats need to be defined for retransmission requests, and the timing rules must be modified to allow immediate feedback.

PACKET FORMATS

The profile for retransmission-based feedback defines two additional RTCP packet types, representing positive and negative acknowledgments. The most common type is expected to be negative acknowledgments, reporting that a particular set of packets was lost. A positive acknowledgment reports that packets were correctly received.

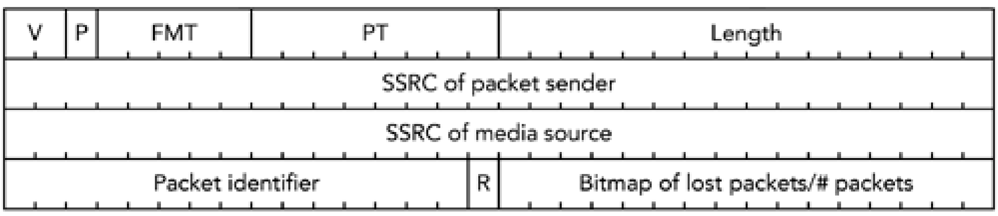
The format of a negative acknowledgment (NACK) is shown in Figure 9.11. The NACK contains a packet identifier representing a lost packet, and a bitmap showing which of the following 16 packets were lost, with a value of 1 indicating loss. The sender should not assume that a receiver has received a packet just because the corresponding position in the bit mask is set to zero; all it knows is that the receiver has not reported the packet lost at this time. On receiving a NACK, the sender is expected to retransmit the packets marked as missing, although it is under no obligation to do so.

Figure 9.11. Format of an RTCP Feedback Negative Acknowledgment



The format of a positive acknowledgment (ACK) is shown in Figure 9.12. The ACK contains a packet identifier representing a correctly received packet, and either a bitmap or a count of the following packets. If the R bit is set to 1, the final field is a count of the number of correctly received packets following the packet identifier. If the R bit is set to zero, the final field is a bitmap showing which of the following 15 packets were also received. The two options allow both long runs of ACKs with few losses (R = 1) and occasional ACKs interspersed with loss (R = 0) to be signaled efficiently.

Figure 9.12. Format of an RTCP Feedback Positive Acknowledgment



The choice between ACK and NACK depends on the repair algorithm in use, and on the desired semantics. An ACK signals that some packets were received; the sender may assume others were lost. On the other hand, a

Implementation Considerations

If error correction is used, an RTP implementation can be made significantly more robust to the adverse effects of IP networks. These techniques come at a price, though: The implementation becomes somewhat more complex, with the receiver needing a more sophisticated playout buffer algorithm, and the sender needing logic to decide how much recovery data to include and when to discard that data.

At a Receiver

Use of these error correction techniques requires that the application have a more sophisticated playout buffer and channel-coding framework than it might otherwise need. In particular, it needs to incorporate FEC and/or retransmission delay into its playout point calculation, and it needs to allow for the presence of repair data in playout buffers.

When calculating the playout point for the media, a receiver has to allow sufficient time for the recovery data to arrive. This may mean delaying the playout of audio/video beyond its natural time, depending on the time needed to receive the recovery data, and the desired playout point of the media.

For example, an interactive voice telephony application might want to operate with a short jitter buffer and a playout delay of only one or two packets' worth of audio. If the sender uses a parity FEC scheme such as that shown in Figure 9.2, in which an FEC packet is sent after every four data packets, the FEC data will be useless because it will arrive after the application has played out the original data it was protecting.

How does an application know when recovery data is going to arrive? In some cases the configuration of the repair is fixed and can be signaled in advance, allowing the receiver to size its playout buffers. Either the signaling can be implicit (for example, RFC 2198 redundancy in which the sender can insert zero-length redundant data into the first few packets of an audio stream, allowing the receiver to know that real redundancy data will follow in later packets), or it can be explicit as part of session setup (for example, included in the SDP during a SIP invitation).

Unfortunately, advance signaling is not always possible, because the repair scheme can change dynamically, or because the repair time cannot be known in advance (for example, when retransmission is used, the receiver has to measure the round-trip time to the sender). In such cases it is the responsibility of the receiver to adapt to make the best use of any repair data it receives, by either delaying media playout or discarding repair data that arrives late. Generally the receiver must make such adaptation without the help of the sender, relying instead on its own knowledge of the application scenario.

A receiver will need to buffer arriving repair data, along with the original media packets. How this is done depends on the form of repair: Some schemes are weakly coupled with the original media, and a generic channel-coding layer can be used; others are tightly coupled to the media and must be integrated with the codec.

Examples of weak coupling include parity FEC and retransmission in which repairs can be made by a general-purpose layer, with no knowledge of the contents of the packets. The reason is that the repair operates on the RTP packets, rather than on the media data itself.

In other cases the repair operation is tightly coupled with the media codec. For example, the AMR payload format[41] includes support for partial checksums and redundant transmission. Unlike the audio redundancy defined in RFC 2198, this form of redundant transmission has no separate header and is specific to AMR: Each packet contains multiple frames, overlapping in time with the following packet. In this case the AMR packetization code must be aware of the overlap, and it must ensure that the frames are correctly added to the playout buffer (and that duplicates are discarded). Another example is the reference picture selection available in MPEG-4 and some modes of H.263, in which the channel coding depends on the shared state between encoder and decoder.

At the Sender

When error correction is in use, the sender is also required to buffer media data longer than it normally would. The amount of buffering depends on the correction technique in use: An FEC scheme requires the sender to hold on to

Summary

In this chapter we have discussed various ways in which errors due to packet loss can be corrected. The schemes in use today include various types of forward error correction and channel coding, as well as retransmission of lost packets.

When used correctly, error correction provides a significant benefit to the perceived quality of a media stream, and it can make the difference between a system being usable or not. If used incorrectly, however, it can lead to a worsening of the problems it was intended to solve, and it can cause significant network problems. The issue of congestion control—adapting the amount of data sent to match the network capacity, as discussed in more detail in Chapter 10, Congestion Control—forms an essential counterpoint to the use of error correction.

One thing that should be clear from this chapter is that error correction usually works by adding some redundancy to a media stream, which can be used to repair lost data. This mode of operation is somewhat at odds with the goal of media compression, which seeks to remove redundancy from the stream. There is a trade-off to be made between compression and error tolerance: At some stage, extra effort spent compressing a media stream is counterproductive, and it is better to use the inherent redundancy for error resilience. Of course, the point at which that line is passed depends on the network, the codec, and the application.

Chapter 10. Congestion Control

Until now we have assumed that it is possible to send a media stream at its natural rate, and that attempts to exceed the natural rate result in packet loss that we must conceal and correct. In the real world, however, our multimedia stream most likely shares the network with other traffic. We must consider the effects we are having on that traffic, and how to be good network citizens.

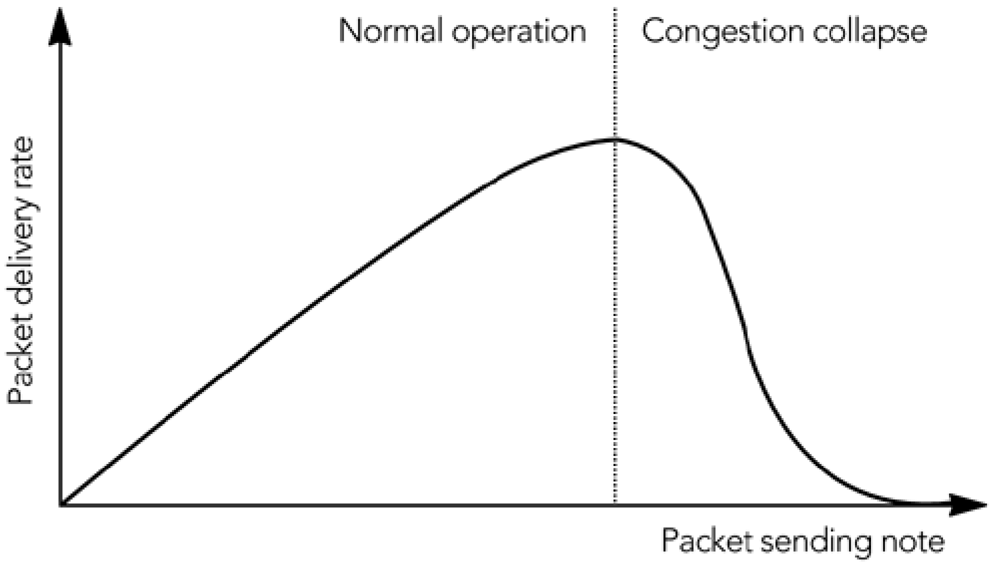
The Need for Congestion Control

Before we discuss the congestion control mechanisms employed on the Internet, and the implications they have for multimedia traffic, it is important to understand what congestion control is, why it is needed, and what happens if applications are not congestion-controlled.

As we discussed in Chapter 2, Voice and Video Communication over Packet Networks, an IP network provides a best-effort packet-switched service. One of the defining characteristics of such a network is that there is no admission control. The network layer accepts all packets and makes its best effort to deliver them. However, it makes no guarantees of delivery, and it will discard excess packets if links become congested. This works well, provided that the higher-layer protocols note packet drops and reduce their sending rate when congestion occurs. If they do not, there is a potential for congestion collapse of the network.

Congestion collapse is defined as the situation in which an increase in the network load results in a decrease in the number of packets delivered usefully by the network. Figure 10.1 shows the effect, with a sudden drop in delivery rate occurring when the congestion collapse threshold is exceeded. Congestion collapse arises when capacity is wasted sending packets through the network that are dropped before they reach their destination—for example, because of congestion at intermediate nodes.

Figure 10.1. Congestion Collapse



Consider a source sending a packet that is dropped because of congestion. That source then retransmits the packet, which again is dropped. If the source continues to resend the packet, and the other network flows remain stable, the process can continue with the network being fully loaded yet no useful data being delivered. This is congestion collapse. It can be avoided if sources detect the onset of congestion and use it as a signal to reduce their sending rate, allowing the congestion to ease.

Congestion collapse is not only a theoretical problem. In the early days of the Internet, TCP did not have an effective congestion control algorithm, and the result was networkwide congestion collapse on several occasions in the mid-1980s.3 To prevent such collapse, the TCP protocol was augmented with the mechanisms developed by Van Jacobson,81 which are described in the next section, Congestion Control on the Internet.

With the increase in non-congestion-controlled multimedia traffic, the potential for congestion collapse is again becoming a concern. Various mechanisms have been proposed by which routers can detect flows that do not respond to congestion73 and penalize those flows with higher loss rates than those that are responsive. Although these mechanisms are not widely deployed at present, it is likely that in the future the network will be more actively policed for compliance with the principles of congestion control.

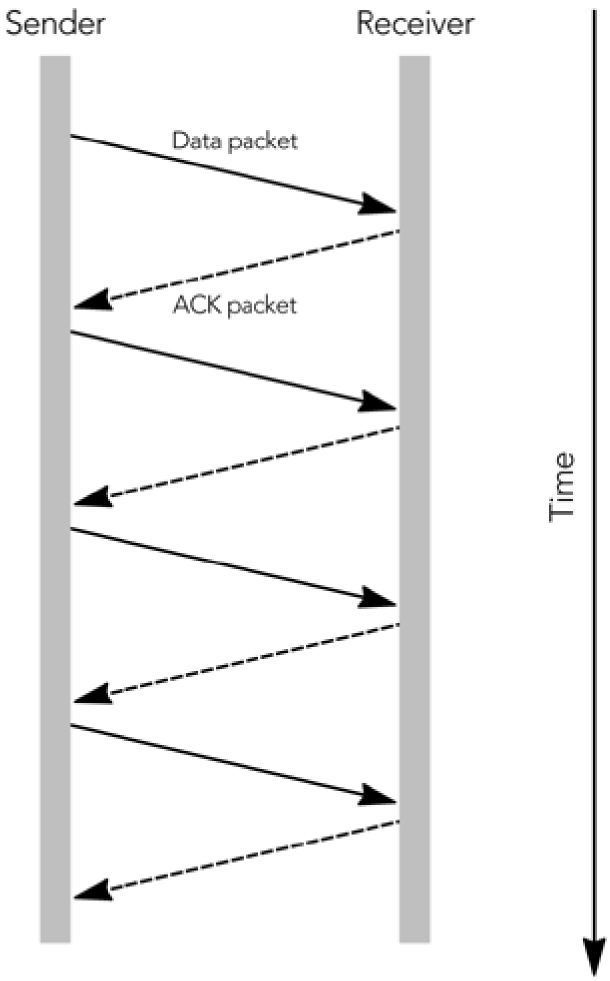
In addition to congestion collapse, there is another reason for employing congestion control: fairness. The policy for sharing network capacity among flows is not defined by IP, but by the congestion control algorithms of the transport

Congestion Control on the Internet

Congestion control on the Internet is implemented by the transport protocols layered above IP. It is a simple matter to explain the congestion control functions of UDP—there are none unless they are implemented by the applications layered above UDP—but TCP has an extensive set of algorithms for congestion control.29,81,112

TCP is an example of a sliding window protocol. The source includes sequence numbers with each data packet it sends, and these are echoed back to it in ACK (positive acknowledgment) packets from the receiver. The simplest sliding window protocol requires each packet to be acknowledged immediately, before the next can be sent, as shown in Figure 10.2. This is known as a stop-and-wait protocol because the sender must wait for the acknowledgment before it can send the next packet. Clearly, a stop-and-wait protocol provides flow control—preventing the sender from overrunning the receiver—but it also hinders performance, especially if the round-trip time between sender and receiver is long.

Figure 10.2. A Simple Stop-and-Wait Protocol



Use of a larger window allows more than one packet to be sent before an ACK is received. Each TCP ACK packet includes a receiver window, indicating to the source how many octets of data can be accepted at any time, and the highest continuous sequence number from the data packets. The source is allowed to send enough packets to fill the receive window, before it receives an ACK. As ACKs come in, the receive window slides along, allowing more data to be sent. This process is shown in Figure 10.3, in which the window allows three outstanding packets. The larger receive window improves performance compared to a simple stop-and-wait protocol.

Figure 10.3. Use of a Sliding Receiver Window

Implications for Multimedia

As noted in Chapter 2, Voice and Video Communication over Packet Networks, the vast majority of traffic—over 95% of octets transported—uses TCP as its transport protocol. Because it is desirable for multimedia traffic to coexist peacefully with other traffic on the network, the multimedia traffic should employ a congestion control algorithm that is fair to that of TCP.

As we have seen, TCP adapts to network congestion on very short timescales, and it gives a somewhat higher share of the network bandwidth to connections with short network round-trip times. Also the evolution of TCP has involved several variants in the congestion control algorithms, each having slightly different behavior. These factors make it difficult to define fairness for TCP: Over what timescale is fairness measured—instantaneous or long-term average? Is it a problem that long-distance connections get less bandwidth than local connections? What about changes in the protocol that affect behavior in some conditions?

The more studies that are undertaken, the more it becomes clear that TCP is not entirely fair. Over the short term, one flow will always win out over another. Over the long term, these variations mostly average out, except that connections with longer round-trip times generally achieve lower throughput on average. The different variants of TCP also have an effect—for example, SACK-TCP gets better throughput with certain loss patterns. At best, TCP is fair within a factor of 2 or 3, over the long term.

The variation in TCP behavior actually makes our job, as designers of congestion control for multimedia applications, easier. We don't have to worry too much about being exactly fair to any particular type of TCP, because TCP is not exactly fair to itself. We have to implement some form of congestion control—to avoid the danger of congestion collapse—but as long as it is approximately fair to TCP, that's the best that can be expected.

Some have argued that a multimedia application doesn't want to be fair to TCP, and that it is acceptable for such traffic to take more than its fair share of the bandwidth. After all, multimedia traffic has strict timing requirements that are not shared by the more elastic TCP flows. To some extent this argument has merit, but it is important to understand the problems that can arise from such unfairness.

For example, consider a user browsing the Web while listening to an online radio station. In many cases it might be argued that the correct behavior is for the streaming audio to be more aggressive than the TCP flow so that it steals capacity from the Web browser. The result is music that does not skip, but less responsive Web downloads. In many cases this is the desired behavior. Similarly, a user sending e-mail while making a voice-over-IP telephone call usually prefers having the e-mail delivered more slowly, rather than having the audio break up.

This prioritization of multimedia traffic over TCP traffic cannot be guaranteed, though. Perhaps the Web page the user is submitting is a bid in an online auction, or a stock-trading request for which it's vital that the request be acted on immediately. In these cases, users may be less than pleased if their online radio station delays the TCP traffic.

If your application needs higher priority than normal traffic gets, this requirement should be signaled to the network rather than being implied by a particular transport protocol. Such communication allows the network to make an intelligent choice between whether to permit or deny the higher priority on the basis of the available capacity and the user's preferences. Protocols such as RSVP/Integrated Services11 and Differentiated Services24 provide signaling for this purpose. Support for these services is extremely limited at the time of this writing; they are available on some private networks but not the public Internet. Applications intended for the public Internet should consider some form of TCP-friendly congestion control.

Congestion Control for Multimedia

At the time of this writing, there are no standards for congestion control of audio/video streams on the Internet. It is possible either to use TCP directly or to emulate its behavior, as discussed in the next section, TCP-Like Rate Control, although mimicking TCP has various problems in practice. There is also work in progress in the IETF to define a standard for TCP-friendly rate control (see the section titled TCP-Friendly Rate Control) that will likely be more suitable for unicast multimedia applications. The state of the art for multicast congestion control is less clear, but the layered coding techniques discussed later in this chapter have, perhaps, the most promise.

TCP-Like Rate Control

The obvious congestion control technique for audio/video applications is either to use TCP or to emulate the TCP congestion control algorithm.

As discussed in Chapter 2, Voice and Video Communication over Packet Networks, TCP has several properties that make it unsuitable for real-time applications, in particular the emphasis on reliability over timeliness. Nevertheless, some multimedia applications do use TCP, and an RTP-over-TCP encapsulation is defined for use with RTSP (Real-Time Streaming Protocol).14

Instead of using TCP directly, it might also be possible to emulate the congestion control algorithm of TCP without the reliability mechanisms. Although no standards exist yet, there have been several attempts to produce such a protocol, with perhaps the most complete being the Rate Adaptation Protocol (RAP), by Rejaie et al.99 Much like TCP, a RAP source sends data packets containing sequence numbers, which are acknowledged by the receiver. Using the acknowledgment feedback from the receiver, a sender can detect loss and maintain a smoothed average of the round-trip time.

A RAP sender adjusts its transmission rate using an additive-increase, multiplicative-decrease (AIMD) algorithm, in much the same manner as a TCP sender, although since it is rate-based, it exhibits somewhat smoother variation than TCP. Unlike TCP, the congestion control in RAP is separate from the reliability mechanisms. When loss is detected, a RAP sender must reduce its transmission rate but is under no obligation to resend the lost packet. Indeed, the most likely response would be to adapt the codec output to match the new rate and continue without recovering the lost data.

Protocols, like RAP, that emulate—to some degree—the behavior of TCP congestion control exhibit behavior that is most fair to existing traffic. They also give an application more flexibility than it would have with standard TCP, allowing it to send data in any order or format desired, rather than being stuck with the reliable, in-order delivery provided by TCP.

The downside of using TCP, or a TCP-like protocol, is that the application has to adapt its sending rate rapidly, to match the rate of adaptation of TCP traffic. It also has to follow the AIMD model of TCP, with the sudden rate changes that that implies. This is problematic for most audio/video applications because few codecs can adapt quickly and over such large ranges, and because rapid changes in picture or sound quality have been found to be disturbing to the viewer.

These problems do not necessarily mean that TCP, or TCP-like, behavior is inappropriate for all audio/video applications, merely that care must be taken to determine its applicability. The main problem with these congestion control algorithms is the rapid rate changes that are implied. To some extent you can insulate the application from these changes by buffering the output, hiding the short-term variation in rate, and feeding back a smoothed average rate to the codec. This can work well for noninteractive applications, which can tolerate the increased end-to-end delay implied by the buffering, but it is not suitable for interactive use.

Ongoing research into protocols combines TCP-like congestion control with unreliable delivery. If one of these is found suitable for use with RTP, it is expected to be possible to extend RTP to support the necessary feedback (using, for example, the RTCP extensions described in Chapter 9, Error Correction44). The difficulty remains in the design of a suitable congestion control algorithm.

Summary

Congestion control is a necessary evil. As an application designer, you can ignore it, and your application may even appear to work better if you do. However, you may be affecting other traffic on the network—perhaps other instances of your own application—in adverse ways, and you have no way of knowing the relative importance of that traffic. For this reason, and to avoid the possibility of causing congestion collapse, applications should implement some form of congestion control. It is also desirable that the chosen algorithm be approximately fair to TCP traffic, allowing unfairness—priority—to be introduced into the network in a controlled manner.

The standards for congestion control are still evolving, so it is difficult to provide a detailed specification for implementers. Nonetheless, it is strongly recommended that unicast applications implement a TCP-friendly rate control algorithm, because these are the best developed. For multicast applications, the choice of congestion control is less clear, but layered coding seems to be the best alternative at present.

Part IV: Advanced Topics

Chapter 11. Header Compression

One area where RTP is often criticized is the size of its headers. Some argue that 12 octets of RTP header plus 28 octets of UDP/IPv4 header is excessive for, say, a 14-octet audio packet, and that a more efficient reduced header could be used instead. This is true to some extent, but the argument neglects the other uses of RTP in which the complete header is necessary, and it neglects the benefit to the community of having a single open standard for audio/video transport.

Header compression achieves a balance between these two worlds: When applied to a link layer, it can compress the entire 40-octet RTP/UDP/IP header into 2 octets, giving greater efficiency than a proprietary protocol over UDP/IP could achieve, with the benefits of a single standard. This chapter provides an introduction to the principles of header compression, and a closer look at two compression standards: Compressed RTP (CRTP) and Robust Header Compression (ROHC).

Introductory Concepts

The use of header compression has a well-established history in the Internet community, since the definition of TCP/IP header compression in 1990[4] and its widespread implementation along with PPP for dial-up links. More recently, the standard for TCP/IP header compression has been revised and updated,[25] and new standards for UDP/IP and RTP/UDP/IP header compression have been developed.[26,27,37]

A typical scenario for header compression comprises a host connected to the network via a low-speed dial-up modem or wireless link. The host and the first-hop router compress packets passing over the low-speed link, improving efficiency on that link without affecting the rest of the network. In almost all cases, there is a particular bottleneck link where it is desirable to use the bandwidth more efficiently, and there is no need for compression in the rest of the network. Header compression works transparently, on a per-link basis, so it is well suited to this scenario: The compressed link looks like any other IP link, and applications cannot detect the presence of header compression.

These features—per-link operation and application transparency—mean that header compression is usually implemented as part of the operating system (often as part of a PPP implementation). An application usually does not need to be aware of the presence of header compression, although in some circumstances, consideration for the behavior of header compression can increase performance significantly. These circumstances are discussed in the section titled Considerations for RTP Applications later in this chapter.

Patterns, Robustness, and Local Implementation

Compression relies on patterns in the packet headers: Many fields are constant or change in a predictable manner between consecutive packets belonging to the same packet stream. If we can recognize these patterns, we can compress those fields to an indication that "the header changed in the expected manner," rather than explicitly sending them. Only header fields that change in an unpredictable manner need to be transmitted in every header.

An important principle in the design of header compression standards has been robustness. There are two aspects to this: robustness to packet loss, and robustness to misidentified streams. Network links—especially wireless links—can lose or corrupt packets, and the header compression scheme must be able to function in the presence of such damage. The most important requirement is that damage to the compressed bit stream not cause undetectable corruption in the uncompressed stream. If a packet is damaged, the following packets will either be decompressed correctly or discarded. The decompressor should never produce a corrupted packet.

The second robustness issue is that RTP flows are not self-identifying. The UDP header contains no field indicating that the data being transported is RTP, and there is no way for the compressor to determine unambiguously that a particular sequence of UDP packets contains RTP traffic. The compressor needs to be informed explicitly that a stream contains RTP, or it needs to be capable of making an educated guess on the basis of observations of packet sequences. Robust engineering requires that compression must not break anything if mistakenly applied to a non-RTP flow. A misinformed compressor is not expected to compress other types of UDP flows, but it must not damage them.

Together, the principles of pattern recognition and robustness allow the formulation of a general principle for header compression: The compressor will send occasional packets containing full headers, followed by incremental updates indicating which header fields changed in the expected manner and containing the "random" fields that cannot be compressed. The full headers provide robustness: Damage to the incremental packets may confuse the decompressor and prevent operation, but this will be corrected when the next full header arrives.

Finally, it is important that header compression be locally implementable. The aim is to develop a header compression scheme that operates over a single link without end-to-end support: If saving bandwidth is important, two systems can spend processor cycles to compress; otherwise, if processing is too expensive, uncompressed headers are sent. If two systems decide to compress headers on a single link, they should be able to do so in a manner that is invisible to all other systems. As a consequence, header compression should be implemented in the network protocol stack, not in the application. Applications are not expected to implement header compression themselves; an application designer should understand header compression and its consequences, but the implementation should be done as part of the protocol stack of the machines on either side of the link.

Compressed RTP

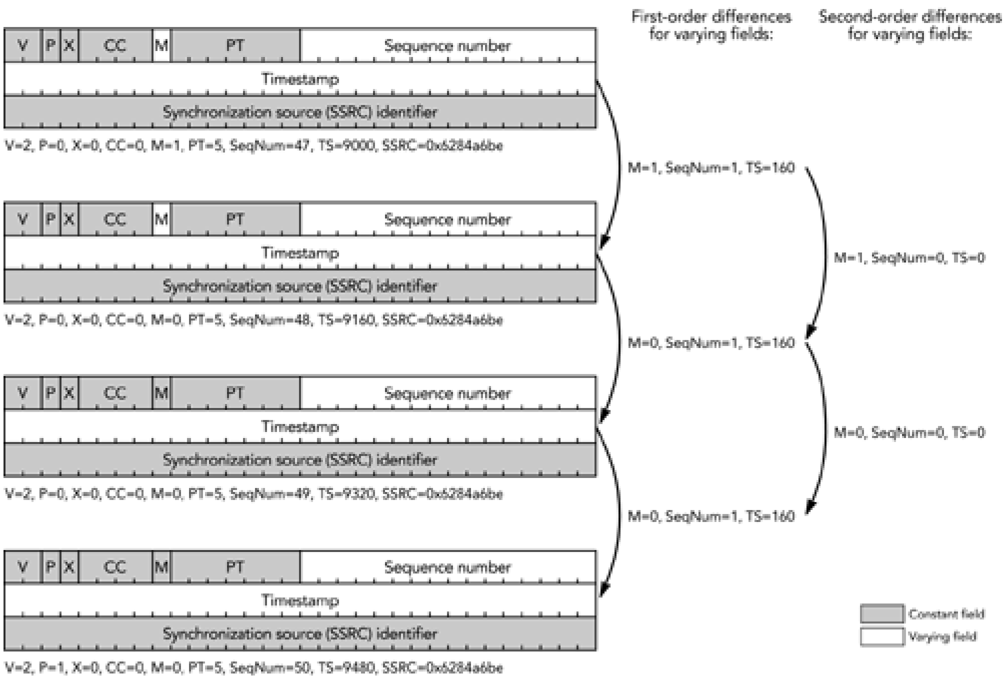
The standard for CRTP is specified in RFC 2508.26 It was designed for operation over low-speed serial links, such as dial-up modems, that exhibit low bit-error rates. CRTP is a direct out-growth of the Van Jacobson header compression algorithm used for TCP,4,25 having similar performance and limitations.

The principal gain of TCP header compression comes from the observation that half of the octets in the TCP/IP headers are constant between packets. These are sent once—in a full header packet—and then elided from the following update packets. The remaining gain comes from differential coding on the changing fields to reduce their size, and from eliminating the changing fields entirely for common cases by calculating the changes from the length of the packet.

RTP header compression uses many of the same techniques, extended by the observation that although several fields change in every packet, the difference from packet to packet is often constant and therefore the second-order difference is zero. The constant second-order difference allows the compressor to suppress the unvarying fields and the fields that change predictably from packet to packet.

Figure 11.1 shows the process in terms of just the RTP header fields. The shaded header fields are constant between packets—they have a first-order difference of zero—and do not need to be sent. The unshaded fields are varying fields; they have a nonzero first-order difference. However, their second-order difference is often constant and zero, making the varying fields predictable.

Figure 11.1. Principles of Header Compression



In Figure 11.1 all the fields except the M bit either are constant or have a zero second-order difference. Therefore, if the initial values for the predictable fields are known, only the change in the M bit needs to be communicated.

Operation of CRTP: Initialization and Context

Compressed RTP starts by sending an initial packet containing full headers, thereby establishing the same state in the compressor and decompressor. This state is the initial context of the compressed stream. Subsequent packets contain reduced headers, which either indicate that the decompressor should use the existing context to predict the headers, or contain updates to the context that must be used for the future packets. Periodically a full header packet can be sent to ensure that any loss of synchronization between compressor and decompressor is corrected.

Full header packets comprise the uncompressed original packet, along with two additional pieces of information—a context identifier and a sequence number—as shown in Figure 11.2. The context identifier is an 8- or 16-bit field that

Robust Header Compression

As noted earlier, CRTP does not work well over links with loss and long round-trip times, such as many cellular radio links. Each lost packet causes several subsequent packets to be lost because the context is out of sync during at least one link round-trip time.

In addition to reducing the quality of the media stream, the loss of multiple packets wastes bandwidth because some packets that have been sent are simply discarded, and because a full header packet must be sent to refresh the context. Robust Header Compression (ROHC)37 was designed to solve these problems, providing compression suitable for use with third-generation cellular systems. ROHC gets significantly better performance than CRTP over such links, at the expense of additional complexity of implementation.

Observation of the changes in header fields within a media stream shows that they fall into three categories:

1. Some fields are static, or mostly static. Examples include the RTP SSRC, UDP ports, and IP addresses. These fields can be sent once when the connection is established, and either they never change or they change very infrequently.
2. Some fields change in a predictable manner with each packet sent, except for occasional sudden changes. Examples include the RTP timestamp and sequence number, and (often) the IPv4 ID field. During periods when these fields are predictable, there is usually a constant relation between them. When sudden changes occur, often only a single field changes unpredictably.
3. Some fields are unpredictable, having essentially random values, and have to be communicated as is, with no compression. The main example is the UDP checksum.

ROHC operates by establishing mapping functions between the RTP sequence number and the other predictable fields, then reliably transferring the RTP sequence number and the unpredictable header fields. These mapping functions form part of the compression context, along with the values of static fields that are communicated at startup or when those fields change.

The main differences between ROHC and CRTP come from the way they handle the second category: fields that usually change in a predictable manner. In CRTP, the value of the field is implicit and the packet contains an indication that it changed in the predictable fashion. In ROHC, the value of a single key field—the RTP sequence number—is explicitly included in all packets, and an implicit mapping function is used to derive the other fields.

Operation of ROHC: States and Modes

ROHC has three states of operation, depending on how much context has been transferred:

1. The system starts in initialization and refresh state, much like the full header mode of CRTP. This state conveys the necessary information to set up the context, enabling the system to enter first- or second-order compression state.
2. First-order compression state allows the system to efficiently communicate irregularities in the media stream—changes in the context—while still keeping much of the compression efficiency. In this state, only a compressed representation of the RTP sequence number, along with the context identifier, and a reduced representation of the changed fields are conveyed.
3. Second-order state is the highest compression level, when the entire header is predictable from the RTP

Considerations for RTP Applications

RTP header compression—whether by CRTP or ROHC—is transparent to the application. When header compression is in use, the compressed link becomes a more efficient conduit for RTP packets, but aside from increased performance, an application should not be able to tell that compression is being used.

Nevertheless, there are ways in which an application can aid the operation of the compressor. The main idea is regularity: An application that sends packets with regular timestamp increments, and with a constant payload type, will produce a stream of RTP packets that compresses well, whereas variation in the payload type or interpacket interval will reduce the compression efficiency. Common causes of variation in the interpacket timing include silence suppression with audio codecs, reverse predicted video frames, and interleaving:

- Audio codecs that suppress packets during silent periods affect header compression in two ways: They cause the marker bit to be set, and they cause a jump in the RTP timestamp. These changes cause CRTP to send a packet containing an updated timestamp delta; ROHC will send a first-order packet containing the marker bit and a new timestamp mapping. Both approaches add at least one octet to the size of the compressed header. Despite this reduction in header compression efficiency, silence suppression almost always results in a net saving in bandwidth because some packets are not sent.

- Reverse predicted video frames—for example, MPEG B-frames—have a timestamp less than that of the previous frame.12 As a result, CRTP sends multiple timestamp updates, seriously reducing compression efficiency. The effects on ROHC are less severe, although some reduction in compression efficiency occurs there also.

- Interleaving is often implemented within the RTP payload headers, with the format designed so that the RTP timestamp increment is constant. In this case, interleaving does not affect header compression, and it may even be beneficial. For example, CRTP has a loss multiplier effect when operating on high-delay links, which is less of an issue for inter-leaved streams than for noninterleaved streams. In some cases, though, interleaving can result in packets that have RTP timestamps with nonconstant offsets. Thus, interleaving will reduce the compression efficiency and is best avoided.

The use of UDP checksums also affects compression efficiency. When enabled, the UDP checksum must be communicated along with each packet. This adds two octets to the compressed header, which, because fully compressed RTP/UDP/IP headers are two octets for CRTP and one octet for ROHC, is a significant increase.

The implication is that an application intended to improve compression efficiency should disable the checksum, but this may not necessarily be appropriate. Disabling the checksum improves the compression ratio, but it may make the stream susceptible to undetected packet corruption (depending on the link layer; some links include a checksum that makes the UDP checksum redundant). An application designer must decide whether the potential for receiving corrupt packets outweighs the gain due to improved compression. On a wired network it is often safe to turn off the checksum because bit errors are rare. However, wireless networks often have a relatively high bit-error rate, and applications that may be used over a wireless link might want to enable the checksum.

The final factor that may affect the operation of header compression is generation of the IPv4 ID field. Some systems increment the IPv4 ID by one for each packet sent, allowing for efficient compression. Others use a pseudorandom sequence of IPv4 ID values, making the sequence unpredictable in an attempt to avoid certain security problems. The use of unpredictable IPv4 ID values significantly reduces the compression efficiency because it is necessary to convey the two-octet IPv4 ID in every packet, rather than allowing it to be predicted. It is recommended that IP implementations increment the IPv4 ID by one when sending RTP packets, although it is recognized that the IP layer will not typically know the contents of the packets (an implementation might provide a call to inform the system that the IPv4 ID should increment uniformly for a particular socket).

Summary

The use of RTP header compression—whether CRTP or ROHC—can significantly improve performance when RTP is operating over low-speed network links. When the payload is small—say, low-rate audio, with packets several tens of octets in length—the efficiency to be gained from CRTP with 2-octet compressed headers, compared to 40-octet uncompressed headers, is significant. The use of ROHC can achieve even greater gains in some environments, at the expense of additional complexity.

Header compression is beginning to be widely deployed. ROHC is an essential part of third-generation cellular telephony systems, which use RTP as their voice bearer channel. CRTP is widely implemented in routers and is beginning to be deployed in end hosts.

Chapter 12. Multiplexing and Tunneling

- The Motivation for Multiplexing
- Tunneling Multiplexed Compressed RTP
- Other Approaches to Multiplexing

Multiplexing and tunneling provide an alternative to header compression, improving the efficiency of RTP by bundling multiple streams together inside a single transport connection. The aim is to amortize the size of the headers across multiple streams, reducing the per-stream overhead. This chapter discusses the motivation for multiplexing and outlines the mechanisms that can be used to multiplex RTP streams.

The Motivation for Multiplexing

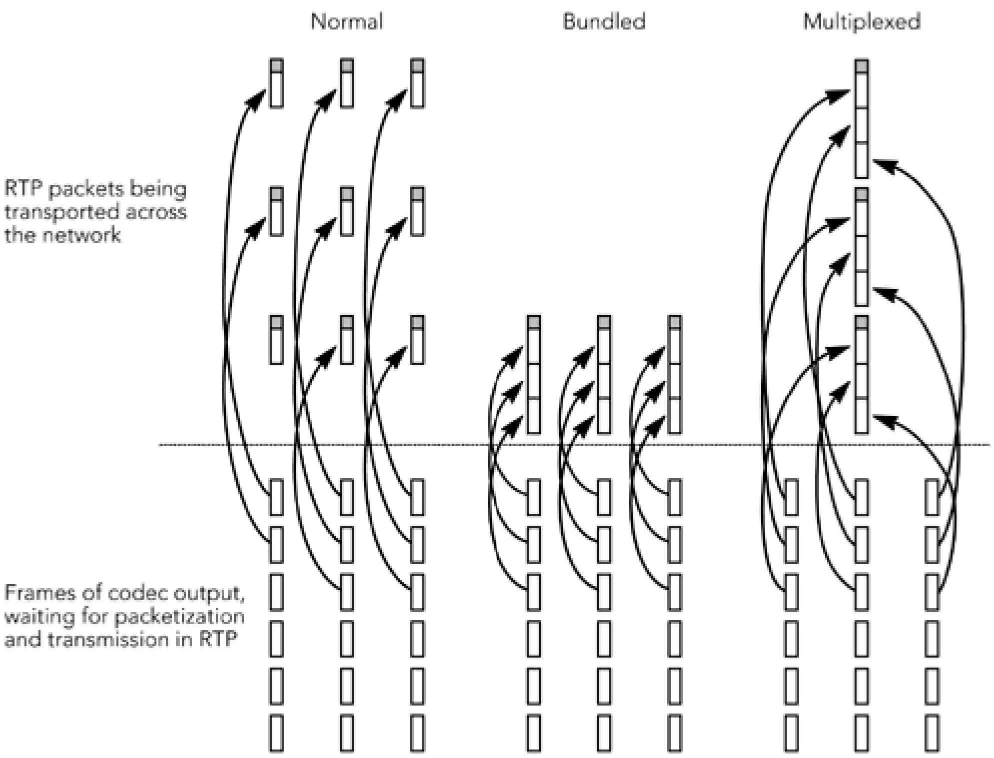
Header compression reduces the size of the headers of a single RTP stream, on a hop-by-hop basis. It provides very efficient transport but requires cooperation from the network (because header compression works hop-by-hop, rather than end-to-end). Header compression adds to the load on routers in terms of additional computation and flow-specific state, both of which may be unacceptable in systems that have to support many hundreds, or even thousands, of simultaneous RTP streams.

The traditional solution to the issues of computation and state within the network has been to push the complexity to the edge, simplifying the center of the network at the expense of additional complexity at the edges: the end-to-end argument. Application of this solution leads to the suggestion that headers should be reduced end-to-end if possible. You can reduce the header in this way by performing header compression end-to-end, thereby reducing the size of each header, or by placing multiple payloads within each packet to reduce the number of headers.

Applying RTP header compression end-to-end is possible, but unfortunately it provides little benefit. Even if the RTP headers were removed entirely, the UDP/IP headers would still be present. Thus a 28-octet overhead would remain for the typical IPv4 case, a size that is clearly unacceptable when the payload is, for example, a 14-octet audio frame. So there is only one possible end-to-end solution: Place multiple payloads within each packet, to amortize the overhead due to UDP/IP headers.

The multiple frames of payload data may come from a single stream or from multiple streams, as shown in [Figure 12.1](#). Placing multiple frames of payload data from a single stream into each RTP packet is known is bundling. As explained in [Chapter 4](#), RTP Data Transfer Protocol, bundling is an inherent part of RTP and needs no special support. It is very effective at reducing the header overhead, but it imposes additional delay on the media stream because a packet cannot be sent until all the bundled data is present.

Figure 12.1. Bundling versus Multiplexing



The alternative, multiplexing, is the process by which multiple frames of payload data, from different streams, are placed into a single packet for transport. Multiplexing avoids the delay penalty inherent in bundling, and in some cases it can significantly improve efficiency. It is, however, not without problems, which may render it unsuitable in many cases:

-

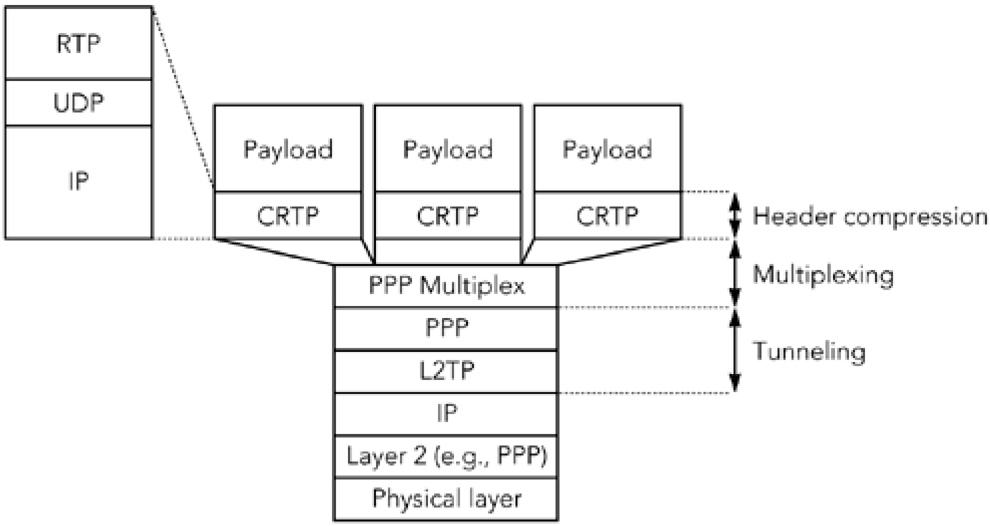
Tunneling Multiplexed Compressed RTP

The IETF Audio/Video Transport working group received numerous proposals for RTP multiplexing solutions. Many of these were driven by the needs of specific applications, and although they may have solved the needs of those applications, they generally failed to provide a complete solution. One of the few proposals to keep the semantics of RTP was Tunneling Multiplexed Compressed RTP (TCRTP), which was adopted as the recommended "best current practice" solution.52

Basic Concepts of TCRTP

The TCRTP specification describes how existing protocols can be combined to provide a multiplex. It does not define any new mechanisms. The combination of RTP header compression, the Layer Two Tunneling Procotol (L2TP),31 and PPP multiplexing39 provides the TCRTP system, with the protocol stack shown in Figure 12.2. Header compression is used to reduce the header overhead of a single RTP payload. Tunneling is used to transport compressed headers and payloads through a multiple-hop IP network without having to compress and decompress at each link. Multiplexing is used to reduce the overhead of tunnel headers by amortizing a single tunnel header over many RTP payloads.

Figure 12.2. TCRTP Protocol Stack



The first stage of TCRTP is to compress the header, which it does in the usual way, negotiating the use of either CRTP or ROHC over a PPP link. The difference is that the PPP link is a virtual interface representing a tunnel rather than a real link. The tunnel is essentially invisible to the header compression, making its presence known only because of the loss and delay characteristics imposed. The concept is much as if the RTP implementation were running over a virtual private network (VPN) link, except that the aim is to provide more efficient transport, rather than more secure transport.

Compared to a single physical link, a tunnel typically has a longer round-trip time, may have a higher packet loss rate, and may reorder packets. As discussed in Chapter 11, Header Compression, links with these properties have adverse effects on CRTP header compression and may lead to poor performance. There are enhancements to CRTP under development that will reduce this problem.43 ROHC is not a good fit because it requires in-order delivery, which cannot be guaranteed with a tunnel.

The tunnel is created by means of L2TP providing a general encapsulation for PPP sessions over an IP network.31 This is a natural fit because both CRTP and ROHC are commonly mapped onto PPP connections, and L2TP allows any type of PPP session to be negotiated transparently. If the interface to the PPP layer is correctly implemented, the CRTP/ROHC implementation will be unaware that the PPP link is a virtual tunnel.

Unfortunately, when the overhead of a tunnel header is added to a single compressed RTP payload, there is very little bandwidth savings compared to uncompressed transport of RTP streams. Multiplexing is required to amortize the overhead of the tunnel header over many RTP payloads. The TCRTP specification proposes the use of PPP

Other Approaches to Multiplexing

Multiplexing has been an area of some controversy, and considerable discussion, within the IETF. Although TCRTP is the recommended best current practice, there are other proposals that merit further discussion. These include Generic RTP Multiplexing (GeRM), which is one of the few alternatives to TCRTP that maintains RTP semantics, and several application-specific multiplexes.

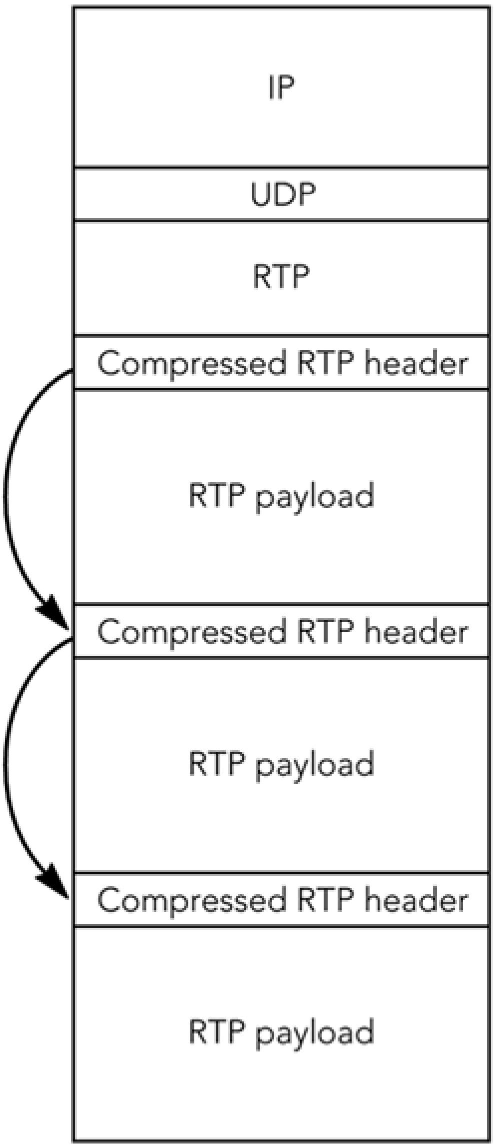
GeRM

Generic RTP Multiplexing (GeRM) was proposed at the IETF meeting in Chicago in August 1998 but was never developed into a complete protocol specification.45 GeRM uses the ideas of RTP header compression, but instead of compressing the headers between packets, it applies compression to multiple payloads multiplexed within a single packet. All compression state is reinitialized in each new packet, and as a result, GeRM can function effectively end-to-end.

CONCEPTS AND PACKET FORMAT

Figure 12.4 shows the basic operation of GeRM. A single RTP packet is created, and multiple RTP packets—known as subpackets—are multiplexed inside it. Each GeRM packet has an outer RTP header that contains the header fields of the first subpacket, but the RTP payload type field is set to a value indicating that this is a GeRM packet.

Figure 12.4. A GeRM Packet Containing Three Subpackets



The first subpacket header will compress completely except for the payload type field and length because the full

Summary

Multiplexing is not usually desirable. It forces all media streams to have a single transport, preventing the receiver from prioritizing them according to its needs, and making it difficult to apply error correction. In almost all cases, header compression provides a more appropriate and higher-performance solution.

Nevertheless, in some limited circumstances multiplexing can be useful, primarily when many essentially identical flows are being transported between two points, something that occurs often when voice-over-IP is being used as a backbone "trunk" to replace conventional telephone lines.

Chapter 13. Security Considerations

Until now we have considered an RTP implementation only in isolation. In the real world, however, there are those who seek to eavesdrop on sessions, impersonate legitimate users, and "hack into" systems for malicious purposes. A correctly implemented RTP system is secure against such attackers and can provide privacy, confidentiality, and authentication services to its users. This chapter describes the features of RTP that enhance user privacy, and outlines various potential attacks and how they may be prevented.

Privacy

The obvious privacy issue is how to prevent unauthorized third parties from eavesdropping on an RTP session. This issue is discussed in the next section, Confidentiality. Confidentiality is not the only privacy issue, though; users of an RTP implementation may want to limit the amount of personal information they give out during the session, or they may want to keep the identities of their communication partners secret.

Information regarding a source, possibly including personal details, is communicated in RTCP source description packets, as noted in Chapter 5, RTP Control Protocol. This information has several uses, most of which are legitimate, but some may be regarded as inappropriate by some users. An example of legitimate use might be a teleconferencing application that uses source description packets to convey the names and affiliations of participants, or to provide other caller identification information. Inappropriate use might include providing personal details via RTCP while the user is listening to an online radio station, hence allowing the station and its advertisers to track their audience. Because of these concerns, it is recommended that applications not send source description packets without first informing the user that the information is being made available.

The exception is the canonical name (CNAME), which is mandatory to send. The canonical name includes the IP address of the participant, but this should not represent a privacy issue, because the same information is available from the IP header of the packet (unless the packets pass through an RTCP-unaware Network Address Translation (NAT) device, in which case CNAME packets will expose the internal address, which some sites might want to hide). The canonical name also exposes the user name of the participant, which may be a greater privacy issue. Applications may omit or rewrite the user name to obscure this information, provided that this is done consistently by any set of applications that will associate sessions via the CNAME.

The CNAME provides a mostly unique identifier, which could be used to track participants. This issue is avoided if the participant is joining from a machine with a temporary IP address (for example, a dial-up user whose IP address is assigned dynamically). If the system has a static IP address, little can be done to protect that address from being tracked, but the CNAME provides little more information than can be obtained from the IP headers (especially if the user name portion of the CNAME is obscured).

Some receivers may not want their presence to be visible at all. It may be acceptable for those receivers not to send RTCP, although doing so prevents the sender from using the reception quality information to adapt its transmission to match the receiver (and it may cause a source to assume that the receiver has failed, and stop transmission). Furthermore, some content providers may require RTCP reports, to gauge the size of their audience.

A related privacy concern involves being able to keep the identities of the partners in a conversation secret. Even if the confidentiality measures described in the next section are used, it is possible for an eavesdropper to observe the RTP packets in transit and gain some knowledge of the communication by looking at the IP headers (for example, your boss might be interested in knowing that your voice-over-IP call is destined for an IP address owned by a recruitment agency). This problem cannot easily be solved with IP, but it can be mitigated if traffic is routed through a trusted third-party gateway that acts as an intermediary with a neutral IP address. (Traffic analysis of the gateway might still reveal communication patterns, unless the gateway is both busy and designed to withstand such analysis.)

Confidentiality

One of the key security requirements for RTP is confidentiality, ensuring that only the intended receivers can decode your RTP packets. RTP content is kept confidential by encryption, either at the application level—encrypting either the entire RTP packet or just the payload section—or at the IP layer.

Application-level encryption has advantages for RTP, but also some disadvantages. The key advantage is that it allows header compression. If only the RTP payload is encrypted, then header compression will work normally, which is essential for some applications (for example, wireless telephony using RTP). If the RTP header is encrypted too, the operation of header compression is disrupted to some extent, but it is still possible to compress the UDP/IP headers.

The other advantage of application-level encryption is that it is simple to implement and deploy, requiring no changes to host operating systems or routers. Unfortunately, this is also a potential disadvantage because it spreads the burden of correct implementation to all applications. Encryption code is nontrivial, and care must be taken to ensure that security is not compromised through poor design or flaws in implementation.

It's worth saying again: Encryption code is nontrivial, and care must be taken to ensure that security is not compromised through poor design or flaws in implementation. I strongly recommend that you study the relevant standards in detail before building a system using encryption, and make sure that you use well-known, publicly analyzed, cryptographic techniques.

Another potential disadvantage of application-level encryption is that it leaves some header fields unencrypted. In some cases, the lack of encryption might reveal sensitive information. For example, knowledge of the payload type field may allow an attacker to ascertain the values of parts of the encrypted payload data, perhaps because each frame starts with a payload header with a standard format. This should not be a problem, provided that an appropriate encryption algorithm is chosen, but it has the potential to compromise an already weak solution.

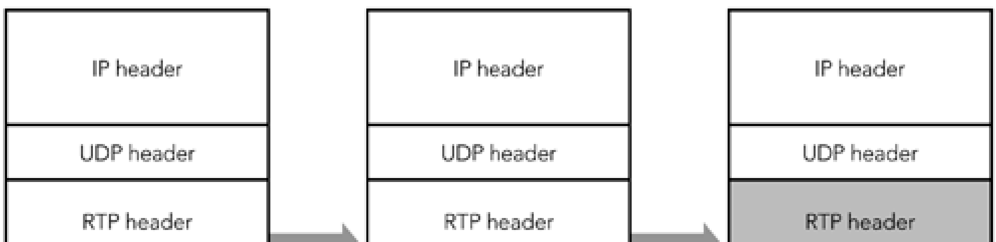
As an alternative, encryption can be performed at the IP layer—for example, using the IP security (IPsec) protocols. This approach has the advantage of being transparent to RTP, and of providing a single—presumably well-tested—suite of encryption code that can be trusted to be correct. The disadvantages of IP-layer encryption are that it disrupts the operation of RTP header compression and its deployment requires extensive changes to host operating systems.

Confidentiality Features in the RTP Specification

The RTP specification provides support for encryption of both RTP data packets (including headers) and RTCP packets.

All octets of RTP data packets—including the RTP header and the payload data—are encrypted. Implementations have a choice of the encryption schemes they support. Depending on the encryption algorithm used, it may be necessary to append padding octets to the payload before encryption can be performed. For example, DES encryption56 operates on blocks of 64 bits, so payloads will need to be padded if they are not multiples of eight octets in length. Figure 13.1 illustrates the process. When padding is used, the P bit in the RTP header is set, and the last octet of the padding indicates the number of padding octets that have been appended to the payload.

Figure 13.1. Standard RTP Encryption of a Data Packet



Authentication

There are two types of authentication: proof that the packets have not been tampered with, known as integrity protection, and proof that the packets came from the correct source, known as source origin authentication.

Integrity protection is achieved through the use of message authentication codes. These codes take a packet to be protected, and a key known only to the sender and receivers, and use these to generate a unique signature. Provided that the key is not known to an attacker, it is impossible to change the contents of the packet without causing a mismatch between the packet contents and the message authentication code. The use of a symmetric shared secret limits the capability to authenticate the source in a multiparty group: All members of the group are able to generate authenticated packets.

Source origin authentication is a much harder problem for RTP applications. It might first be thought equivalent to the problem of generating message authentication codes, but it is more difficult because a shared secret between sender and receiver is not sufficient. Rather, it is necessary to identify the sender in the signature, meaning that the signature is larger and more expensive to compute (in much the way public key cryptography is more expensive than symmetric cryptography). This requirement often makes it infeasible to authenticate the source of each packet in an RTP stream.

Like confidentiality, authentication can be applied at either the application level or the IP level, with much the same set of advantages and disadvantages. Both alternatives have been developed for use with RTP.

Authentication Using Standard RTP

Standard RTP provides no support for integrity protection or source origin authentication. Implementations that require authentication should either implement secure RTP or use a lower-layer authentication service such as that provided by the IP security extensions.

Authentication Using the Secure RTP Profile

SRTP supports both message integrity protection and source origin authentication. For integrity protection, a message authentication tag is appended to the end of the packet, as was shown in Figure 13.4. The message authentication tag is calculated over the entire RTP packet and is computed after the packet has been encrypted. At the time of this writing, the HMAC-SHA-1 integrity protection algorithm is specified for use with SRTP. Other integrity protection algorithms may be defined in the future, and negotiated for use with SRTP.

Source origin authentication using the TESLA (Timed Efficient Stream Loss-tolerant Authentication) authentication algorithm has been considered for use with SRTP, but TESLA is not fully defined at the time of this writing. You are advised to consult the SRTP specification, when it is completed, for details.

The authentication mechanisms of SRTP are not mandatory, but I strongly recommend that all implementations use them. In particular, you should be aware that it is trivially possible for an attacker to forge data encrypted using AES in counter mode unless authentication is used. The secure RTP profile specification describes this issue in detail.

Authentication Using IP Security

The IP security extensions can provide integrity protection and authentication for all packets sent from a host. There are two ways this can be done: as part of the encapsulating security payload, or as an authentication header.

The Encapsulating Security Payload was described earlier, in the section titled Confidentiality Using IP Security. As shown in Figure 13.9, ESP includes an optional authentication data section as part of the trailer. If present, the authentication provides a check on the entire encapsulated payload, plus the ESP header and trailer. The outer IP header is not protected.

An alternative to ESP is the Authentication Header (AH) defined in RFC 2402[18] and shown in Figure 13.11. The fields in this header are much like their counterparts in ESP, and the AH can be used in both tunnel mode and

Replay Protection

Related to authentication is the issue of replay protection: stopping an attacker from recording the packets of an RTP session and reinjecting them into the network later for malicious purposes. The RTP timestamp and sequence number provide limited replay protection because implementations are supposed to discard old data. However, an attacker can observe the packet stream and modify the recorded packets before playback such that they match the expected timestamp and sequence number of the receiver.

To provide replay protection, it is necessary to authenticate messages for integrity protection. Doing so stops an attacker from changing the sequence number of replayed packets, making it impossible for old packets to be replayed into a session.

Denial of Service

A potential denial-of-service threat exists with payload formats using compression that has nonuniform receiver-end computational load. If the payload format has such properties, an attacker might be able to inject pathological packets into a media stream, which are complex to decode and cause the receiver to be overloaded. There is little a receiver can do to avoid this problem, short of stopping processing packets in overload situations.

Another problem that can cause denial of service is failure to implement the RTCP timing rules correctly. If participants send RTCP at a constant rate rather than increasing the interval between packets as the size of the group increases, the RTCP traffic will grow linearly with the size of the group. For large groups, this growth can result in significant network congestion. The solution is to pay close attention to the correct implementation of RTCP.

Similarly, failure to implement the RTCP reconsideration algorithms can result in transient congestion when rapid changes in membership occur. This is a secondary concern because the effect is only transitory, but it may still be an issue.

Network congestion can also occur if RTP implementations respond inappropriately to packet loss. Chapter 10, Congestion Control, describes the issues in detail; for our purposes here, suffice it to say that congestion can cause significant denial of service.

Mixers and Translators

It is not possible to use an RTP mixer to combine encrypted media streams unless the mixer is trusted and has access to the encryption key. This requirement typically prevents mixers from being used if the session needs to be kept confidential.

Translation is also difficult, although not necessarily impossible, if the media streams are encrypted. Translation that involves recoding of the media stream is typically impossible unless the translator is trusted. Some translations, however, do not require recoding of the media stream—for example, translation between IPv4 and IPv6 transport—and these may be provided without affecting the encryption, and without having to trust the translator.

Source origin authentication is similarly difficult if the mixer or translator modifies the media stream. Again, if the mixer or translator is trusted, it may re-sign the modified media stream; otherwise the source cannot be authenticated.

Active Content

Most audiovisual content is passive: It comprises encoded data that is passed through a static decryption algorithm to produce the original content. With the exception of the issues due to nonuniform processing requirements noted earlier, such content is not dangerous.

There is, however, another class of content: that which contains executable code—for example, Java applets or ECMAScript, which can be included with MPEG-4 streams. Such active content has the potential to execute arbitrary operations on the receiving system, unless carefully restricted.

Other Considerations

The text of SDES items is not null-terminated, and manipulating SDES items in languages that assume null-terminated strings requires care. This is a particular problem with C-based implementations, which must take care to ensure that they use lengthchecking string manipulation functions—for example, strncpy() rather than strcpy(). Careless implementations may be vulnerable to buffer overflow attacks.

The text of SDES items is entered by the user, and thus it cannot be trusted to have safe values. In particular, it may contain metacharacters that have undesirable side effects. For example, some user interface scripting languages allow command substitution to be triggered by metacharacters, potentially giving an attacker the means to execute arbitrary code.

Implementations should not assume that packets are well formed. For example, it might be possible for an attacker to produce packets whose actual length does not correspond to the expected length. Again, there is a potential for buffer overflow attacks against careless implementations.

Summary

This chapter has outlined the features of RTP that provide privacy and confidentiality of communication, as well as authentication of participants and the media stream. It has also described various potential attacks on a system, and how they can be prevented.

These are not the only issues that a secure implementation needs to consider, though; it is also necessary to secure the session initiation and control protocols. This chapter has highlighted some issues relating to the security of these initiation and control protocols, but a full treatment of this topic is outside the scope of this book.

References

This book includes various references to Internet RFC (Request for Comments) standards and informational documents. These can be retrieved directly from the RFC Editor (at http://www.rfc-editor.org) or from the Internet Engineering Task Force (IETF, at http://www.ietf.org). RFC documents do not change after they've been published, but they can be made obsolete by new standards. The RFC Editor maintains a list giving the current status of each RFC (http://www.rfc-editor.org/rfc-index.html); it is advisable to check that an RFC is still current before beginning implementation (for example, the RTP specification RFC 1889 is under revision, and the revision will result in a new RFC that makes RFC 1889 obsolete).

In some cases it has been necessary to reference work in progress, in the form of Internet-Drafts. These are the working documents of the IETF; some are eventually published as RFCs, and others are discarded. The Internet-Drafts referenced here are expected to be available in RFC form shortly. The RFC Editor Web site has a search function that allows you to find the RFC form of these documents when published. The IETF maintains an archive of Internet-Drafts online for up to six months after their submission; at that time they must be published as an RFC, revised, or discarded.

It is important to remember that Internet-Drafts are not reference material and may change at any time. Each draft carries the following note as a reminder:

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as work in progress.

Implementing from an Internet-Draft is not wise, unless you are willing to face the possibility that the draft will change while you are writing code. In particular, it is not appropriate to ship products based on Internet-Drafts, because they may be made obsolete by the final standard.

IETF RFC Standards

1. D. Cohen. "Specifications for the Network Voice Protocol," Network Working Group, RFC 741, November 1977.

2. D. Crocker. "Standard for the Format of ARPA Internet Text Messages," Network Working Group, RFC 822, August 1982.

3. J. Nagle. "Congestion Control in IP/TCP Internetworks," Network Working Group, RFC 896, January 1984.

4. V. Jacobson. "Compressing TCP/IP Headers for Low-Speed Serial Links," Internet Engineering Task Force, RFC 1144, February 1990.

5. D. L. Mills. "Network Time Protocol (Version 3): Specification, Implementation and Analysis," Internet Engineering Task Force, RFC 1305, March 1992.

6. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. "RTP: A Transport Protocol for Real-Time Applications," Internet Engineering Task Force, RFC 1889, January 1996.

7. H. Schulzrinne. "RTP Profile for Audio and Video Conferences with Minimal Control," Internet Engineering Task Force, RFC 1890, January 1996.

8. S. Bradner. "The Internet Standards Process—Revision 3," Internet Engineering Task Force, RFC 2026, October 1996.

9. T. Turletti and C. Huitama. "RTP Payload Format for H.261 Video Streams," Internet Engineering Task Force, RFC 2032, October 1996.

10. C. Perkins, I. Kouvelas, O. Hodson, V. Hardman, M. Handley, J. C. Bolot, A. Vega-Garcia, and S. Fosse-Parisis. "RTP Payload for Redundant Audio Data," Internet Engineering Task Force, RFC 2198, September 1997.

11. B. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. "Resource ReSerVation Protocol (RSVP)—Version 1 Functional Specification," Internet Engineering Task Force, RFC 2205, September 1997.

12. D. Hoffman, G. Fernando, V. Goyal, and M. Civanlar. "RTP Payload Format for MPEG1/MPEG2 Video," Internet Engineering Task Force, RFC 2250, January 1998.

13. F. Yergeau. "UTF-8, a Transformation Format of ISO 10646," Internet Engineering Task Force, RFC 2279, January 1998.

14. H. Schulzrinne, A. Rao, and R. Lanphier. "Real Time Streaming Protocol (RTSP)," Internet Engineering Task Force, RFC 2326, April 1998.

15. M. Handley and V. Jacobson. "SDP: Session Description Protocol," Internet Engineering Task Force, RFC 2327, April 1998.

16. R. Hinden and S. Deering. "IP Version 6 Addressing Architecture," Internet Engineering Task Force, RFC 2373, July 1998.

17. S. Kent and R. Atkinson. "Security Architecture for the Internet Protocol," Internet Engineering Task Force, RFC 2401, November 1998.

18. S. Kent and R. Atkinson. "IP Authentication Header," Internet Engineering Task Force, RFC 2402, November 1998.

IETF Internet-Drafts

43. T. Koren, S. Casner, J. Geevarghese, B. Thompson, and P. Ruddy. "Enhanced Compressed RTP (CRTP) for Links with High Delay, Packet Loss and Reordering," Internet Engineering Task Force, Work in Progress (Update to RFC 2508), February 2003.

44. J. Ott, S. Wenger, N. Sato, C. Burmeister, and J. Rey. "Extended RTP Profile for RTCP-Based Feedback (RTP/AVPF)," Internet Engineering Task Force, Work in Progress, February 2003.

45. M. Handley. "GeRM: Generic RTP Multiplexing," Internet Engineering Task Force, Work in Progress, November 1998.

46. A. Valencia and T. Koren. "L2TP Header Compression ("L2TPHC")," Internet Engineering Task Force, Work in Progress, November 2002.

47. A. Li, F. Liu, J. Villasenor, J-H. Park, D-S. Park, Y. L. Lee, J. Rosen-berg, and H. Schulzrinne. "An RTP Payload Format for Generic FEC with Uneven Level Protection," Internet Engineering Task Force, Work in Progress, November 2002.

48. G. Liebl, M. Wagner, J. Pandel, and W. Weng. "An RTP Payload Format for Erasure-Resilient Transmission of Progressive Multimedia Streams," Internet Engineering Task Force, Work in Progress, March 2003.

49. H. Schulzrinne and S. Casner. "RTP Profile for Audio and Video Conferences with Minimal Control," Internet Engineering Task Force, Work in Progress (Update to RFC 1890), March 2003.

50. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. "RTP: A Transport Protocol for Real-Time Applications," Internet Engineering Task Force, Work in Progress (Update to RFC 1889), March 2003.

51. S. Casner and P. Hoschka. "MIME Type Registration of RTP Payload Formats," Internet Engineering Task Force, Work in Progress, November 2001.

52. B. Thompson, T. Koren, and D. Wing. "Tunneling Multiplexed Compressed RTP ('TCRTP')," Internet Engineering Task Force, Work in Progress, November 2002.

53. L-A. Larzon, M. Degermark, S. Pink, L-E. Jonsson (Editor), and G. Fairhurst (Editor). . "The UDP Lite Protocol," Internet Engineering Task Force, Work in Progress, December 2002.

54. S. Bhattacharyya, C. Diot, L. Giuliano, R. Rockell, J. Meylor, D. Meyer, G. Shepherd, and B. Haberman. "An Overview of Source-Specific Multicast (SSM)," Internet Engineering Task Force, Work in Progress, November 2002.

55. M. Baugher, D. McGrew, D. Oran, R. Blom, E. Carrara, M. Naslund, and K. Norrman. "The Secure Real Time Transport Protocol," Internet Engineering Task Force, Work in Progress, June 2002.

Other Standards

56. NIST. "Data Encryption Standard (DES)," Federal Information Processing Standard, FIPS 46-2, December 1993.

57. NIST. "Data Encryption Standard (DES)," Federal Information Processing Standard, FIPS 46-3, October 1999.

58. NIST. "Advanced Encryption Standard (AES)," Federal Information Processing Standard, FIPS-197, 2001 (http://www.nist.gov/aes).

59. ETSI SAGE 3GPP Standard Algorithms Task Force. "Security Algorithms Group of Experts (SAGE); General Report on the Design, Specification and Evaluation of 3GPP Standard Confidentiality and Integrity Algorithms," Public Report, Draft Version 1.0, December 1999.

60. ETSI Recommendation GSM 6.11 "Substitution and Muting of Lost Frames for Full Rate Speech Channels," 1992.

61. ITU-T Recommendation G.114. "One-way Transmission Time," May 2000.

62. ITU-T Recommendation H.323. "Packet-Based Multimedia Communications Systems," November 2000.

63. ITU-T Recommendation P.861. "Objective Quality Measurement of Telephone-Band (300-3400 Hz) Speech Codecs," February 1998.

64. ITU-T Recommendation P.862. "Perceptual Evaluation of Speech Quality (PESQ), an Objective Method for End-to-End Speech Quality Assessment of Narrowband Telephone Networks and Speech Codecs," January 2001.

Conference and Journal Papers

65. D. D. Clark and D. L. Tennenhouse. "Architectural Considerations for a New Generation of Protocols," Proceedings of the SIGCOMM Symposium on Communications Architectures and Protocols, Computer Communications Review, Volume 20, Number 4, September 1990.

66. J. C. Bolot and A. Vega Garcia. "The Case for FEC-Based Error Control for Packet Audio in the Internet," to appear in ACM Multimedia Systems.

67. J. C. Bolot and A. Vega Garcia. "Control Mechanisms for Packet Audio in the Internet," Proceedings of IEEE Infocom '96, San Francisco, CA, March 1996.

68. D. Chiu and R. Jain. "Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks," Journal of Computer Networks and ISDN Systems, Volume 17, Number 1, June 1989.

69. B. J. Dempsey, J. Liebeherr, and A. C. Weaver. "On Retransmission-Based Error Control for Continuous Media Traffic in Packet Switched Networks," Computer Networks and ISDN Systems, Volume 28, 1996, pages 719–736.

70. J. H. Saltzer, D. P. Reed, and D. D. Clark. "End-to-End Arguments in System Design," ACM Transactions on Computer Systems, Volume 2, Number 4, November 1984.

71. S. Floyd and V. Paxson. "Difficulties in Simulating the Internet," IEEE/ACM Transactions on Networking, Volume 9, Number 4, August 2001. An earlier version appeared in the Proceedings of the 1997 Winter Simulation Conference, Atlanta, GA, December 1997.

72. S. Floyd, M. Handley, J. Padhye, and J. Widmer. "Equation-Based Congestion Control for Unicast Applications," Proceedings of ACM SIGCOMM 2000, Stockholm, August 2000.

73. S. Floyd and K. Fall. "Promoting the Use of End-to-End Congestion Control in the Internet," IEEE/ACM Transactions on Networking, Volume 7, Number 4, August 1999.

74. D. J. Goodman, G. B. Lockhart, and O. J. Wasem. "Waveform Substitution Techniques for Recovering Missing Speech Segments in Packet Voice Communications," IEEE Transactions on Acoustics, Speech and Signal Processing, Volume 34, Number 6, December 1986.

75. J. G. Gruber and L. Strawczynski. "Subjective Effects of Variable Delay and Speech Clipping in Dynamically Managed Voice Systems," IEEE Transactions on Communications, Volume 33, Number 8, August 1985.

76. M. Handley, J. Crowcroft, C. Bormann, and J. Ott. "Very Large Conferences on the Internet: The Internet Multimedia Conferencing Architecture," Journal of Computer Networks and ISDN Systems, Volume 31, Number 3, February 1999.

77. V. Hardman, M. A. Sasse, M. Handley, and A. Watson. "Reliable Audio for Use over the Internet," Proceedings of INET '95, Honolulu, HI, June 1995.

78. M. Handley, J. Padhye, S. Floyd, and J. Widmer. "TCP Friendly Rate Control (TFRC): Protocol Specification," Internet Engineering Task Force, RFC 3448, January 2003.

79. O. Hodson, C. Perkins, and V. Hardman. "Skew Detection and Compensation for Internet Audio Applications," Proceedings of the IEEE International Conference on Multimedia and Expo, New York, July 2000.

80. C. Perkins, O. Hodson, and V. Hardman. "A Survey of Packet Loss Recovery Techniques for Streaming Media", IEEE Network Magazine, September/October 1998.

81. V. Jacobson. "Congestion Avoidance and Control," Proceedings of ACM SIGCOMM '88, Stanford, CA,

Books

110. N. Doraswamy and D. Harkins, IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks, Prentice Hall, Upper Saddle River, NJ, 1999.

111. A. B. Johnston. SIP: Understanding the Session Initiation Protocol, Artech House, Norwood, MA, 2001.

112. W. R. Stevens. TCP/IP Illustrated, Volume 1, Addison Wesley, Reading, MA, 1994.

113. A. S. Tanenbaum. Computer Networks, 3rd Edition, Prentice Hall, Upper Saddle River, NJ, 1996.

114. R. M. Warren. Auditory Perception—A New Analysis and Synthesis, Cambridge University Press, Cambridge, England, 1999.

Web Sites

115. The 3rd Generation Partnership Project, http://www.3gpp.org.

116. The Internet Traffic Archive, http://www.acm.org/sigcomm/ITA.

117. Cooperative Association of Internet Data Analysis, http://www.caida.org.

118. Telcordia Internet Sizer, Internet Growth Forecasting Tool, 2000, http://www.telcordia.com/research/netsizer.

119. National Laboratory for Applied Network Research, http://www.nlanr.net.

120. The Internet Weather Report, http://www.internetweather.com.

Other References

121. M. Eubanks. Personal communication and discussion on the Mbone@isi.edu mailing list, April 2001.

122. M. Handley. "An Examination of Mbone Performance," University of Southern California, Information Sciences Institute, Research Report ISI/RR-97-450, 1997.

123. S. Leinen. "UDP vs. TCP Distribution," message to the end2end-interest mailing list, February 2001.

124. V. Paxson. "Measurements and Analysis of End-to-End Internet Dynamics," Ph.D. dissertation, University of California, Berkeley, 1997.

125. D. Sisalem, H. Schulzrinne, and F. Emanuel. "The Direct Adjustment Algorithm: A TCP-Friendly Adaptation Scheme," Technical Report, GMD-FOKUS, August 1997.