CHAPTER 1

Managing Your IP Address Space

The first step in achieving a scalable and effective IP network is devising a solid addressing plan. Your addressing plan lays down the foundation for the network by portioning your IP address space into smaller, manageable ranges, or *blocks*. The addressing plan also defines the deployment of these blocks into various parts of the network for supporting devices.

Unlike such protocols as IPX or AppleTalk, IP requires a respectable amount of address planning at the outset. This is true for large and small networks alike, because the growth of the Internet has made IP addresses a precious and scarce resource.

The Internet's IP address space is finite. With the growth of the Internet, the number of available addresses is diminishing and addresses are becoming more difficult to obtain. Although addressing is a rather mundane task, a solid addressing plan will save you many headaches in the future (and protect your reputation when others inherit your work). Also, IP networks can—and generally should—have a hierarchical addressing structure. This is achieved by summarizing, or *aggregating*, addresses. Summarization heightens the importance of address planning even more (see "Planning for Address Summarization," later in this chapter).

Devising your address strategy is akin to planning the layout of a house. You are going to spend a lot of time in your house, so a crucial step is spending enough time on the design and allocation of the floor space for now and in the future. Are there enough rooms? Is the size of each room adequate and appropriate? What is the most efficient use of the floor space? Although you cannot guarantee a final house design that meets all future requirements, you need to come up with a plan that makes the most sense. You want a well-thought-out design that will postpone any remodeling efforts until far off in the future. By all means, you want to avoid having to demolish the whole thing and start over with a new floor plan. Like floor plans, IP addressing plans generally do not change for long periods of time and, when they do change, overhauling them can be a major effort.

This chapter covers IP addressing concepts, design techniques, strategies for maximizing efficiency, and services for scaling network addressing.

The main topics of this chapter are

- Review of Traditional IP Addressing
- Subnetting a Classful Address Space

- Subnetting with Variable Length Subnet Masks
- Overview of Classless Addressing
- Planning for Address Summarization
- Conserving Subnets with IP Unnumbered
- Scaling the Address Space with Network Address Translation

Review of Traditional IP Addressing

Traditional IP addressing organizes the entire 32-bit IP address space into blocks called *classes* and further breaks down each class into network numbers. Early Internet standards defined five classes, outlined in Table 1-1.

Table 1-1 *The Original Organization of the 32-bit Address Space*

Class Name	Address Range	# of Addresses per Network	Purpose
A	0.1.0.0 to 126.0.0.0	16,777,216	Unicast; very large networks
B	128.0.0.0 to 191.255.0.0	65,536	Unicast; large networks
C	192.0.1.0 to 223.255.255.0	256	Unicast; small networks
D	224.0.0.0 to 239.255.255.255	N/A	Multicast
E	240.0.0.0 to 247.255.255.255	N/A	Experimental use

NOTE Network 127.0.0.0 is a special range of addresses reserved for *loopback addresses* (addresses used locally by IP hosts). Such addresses should never appear on a network.

As Table 1-1 illustrates, a 32-bit IP address is written as four *octets* (8-bit groups) separated by periods, with each octet expressed as a decimal number. This is known as *dotted decimal notation*. The following is an example IP address in its binary and dotted decimal forms:

32-bit IP address: 10101100000100000000101000010100
Same address grouped into four octets: 10101100.00010000.00001010.00010100
Same address in dotted decimal notation: 172.16.10.20
Class of the network: B
Network the address belongs to: 172.16.0.0

The class scheme served as a starting point for easy and rapid deployment of the Internet address space. Much like acquiring land for their buildings, organizations obtained network numbers from the three classes (classes A, B, and C) based on the number of IP addresses they needed. Two classes were reserved for special purposes: class D addresses for IP multicast and class E addresses for experimental use.

After an organization secured a class B network, for example, it could autonomously deploy the addresses contained in that range to its computers, or *hosts*. With the additional deployment of internetworking services (*routing*), that class B network could communicate with other class A, class B, and class C networks within the organization and throughout the Internet.

NOTE This book covers IP version 4, which is the most prevalent form of IP on private networks and the public Internet at the time of this writing. The next version of IP, version 6, has a different addressing format and intends to provide a much larger address space than IP version 4 (IPv6 increases the address space from 32 bits to 128 bits). See the bibliography for sources of IP version 6 information.

To gain more efficient use of the address space, the Internet community adopted a practice of dividing a network into subnetworks called *subnets*. When a network is divided into subnets, its original network number is called the *major network number* or *major net*. Routing is still required to interconnect subnets just as it is required to interconnect major nets.

For most organizations, subnetting is a necessary part of managing an address space—it portions a single major net of limited use into smaller subnets that can be deployed more effectively.

Still, networking professionals are faced with addressing problems that subnetting alone cannot solve. The scarce supply of major nets and pressure from an ever-growing IP population have taken the menial task of addressing to the top of the priority list. Later sections of this chapter offer solutions that will help you get more efficient use of your address space and alleviate the shortage problem.

Subnetting a Classful Address Space

As mentioned previously, the Internet's original address plan was organized into classes: classes A, B, C, D, and E. Networks deployed with this plan are said to be *classful networks* or networks with *classful addressing*. Many privately owned networks still use classful addressing, even though the public Internet has abolished classful addressing in favor of *classless addressing* (covered in "Overview of Classless Addressing" later in this chapter).

In brief, classless addressing discontinues the grouping of addresses into classes A, B, and C and treats the address space as a large, contiguous block of addresses.

NOTE The Internet community adopted classless addressing to get efficient use of the existing address space and to avoid address depletion. See "Overview of Classless Addressing" later in this chapter.

Why care about classful addressing versus classless addressing? Addresses are addresses, aren't they? The distinction between classful and classless addressing is important when it comes to routing protocols. Some routing protocols—Routing Information Protocol (RIP) and Interior Gateway Routing Protocol (IGRP), for example—were created before the practice of classless addressing and support only the rules defined by traditional classful addressing (these rules are simple, but restrictive). Classful routing protocols, such as RIP, do not support newer and more advanced features developed in classless routing protocols, such as Open Shortest Path First (OSPF) and Enhanced IGRP (EIGRP). These advanced features include variable length masking and summarization and are covered later in this chapter (see "Subnetting with Variable Length Subnet Masks," "Overview of Classless Addressing," and "Planning for Address Summarization"). Routing protocols are also covered in Chapter 2, "Deploying Interior Routing Protocols," and Chapter 3, "Managing Routing Protocols."

Although the Internet has ceased using classful addressing, many organizations need to support networks that were designed with classful networks and classful routing protocols, such as RIP and IGRP. This section covers the basics of subnetting because the technique is crucial for supporting a classful network and is a prerequisite to deploying classless networks. The section includes discussion on

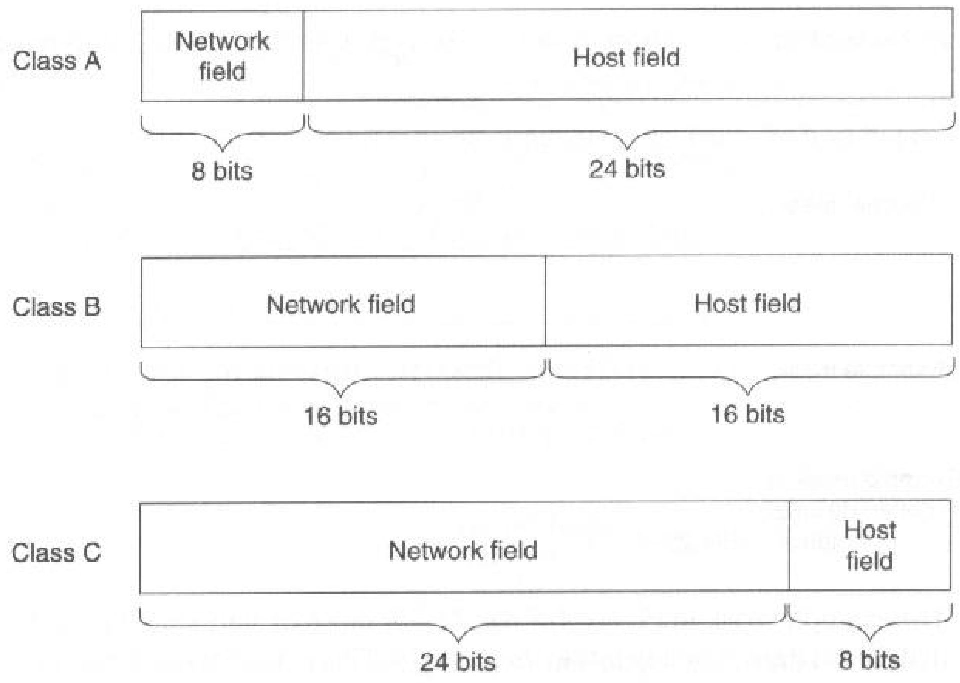
- Major Nets and Subnet Masks
- Classful Subnetting: An Example
- Calculating the Number of Host Addresses in a Subnet
- Finding Subnet Information, Given a Host Address and the Mask
- Disadvantages of Subnetting
- The Rules on Top and Bottom Subnets
- Using Subnet-Zero to Get Around the Rules

Major Nets and Subnet Masks

Every major net has two fields: the *network field*, which uniquely identifies the major net, and the *host field*, which uniquely identifies hosts within the major net. Figure 1-1 illustrates the number of bits in the network and host fields for each class.

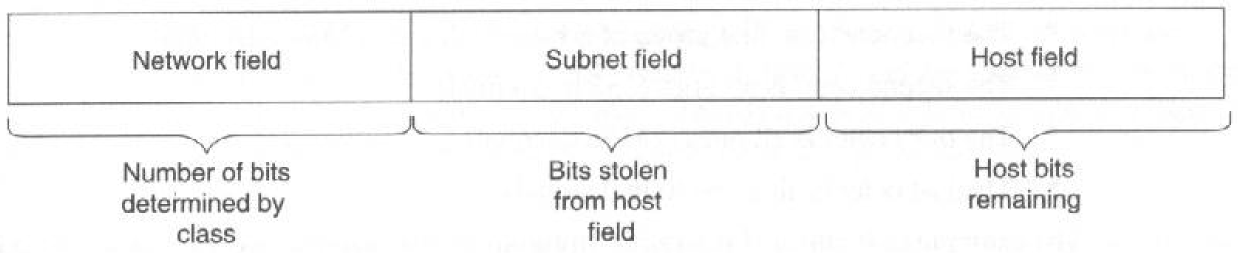
As mentioned in the previous section, subnetting is the process of dividing a major net into smaller (and generally more useful) subnets. This is accomplished by "stealing" some bits from the host field of the major net and using those bits to designate the subnet addresses. The host field varies in length, depending on the class of major net being subnetted (see Figure 1-1).

Figure 1-1 *Lengths of the Network and Host Fields by Class*



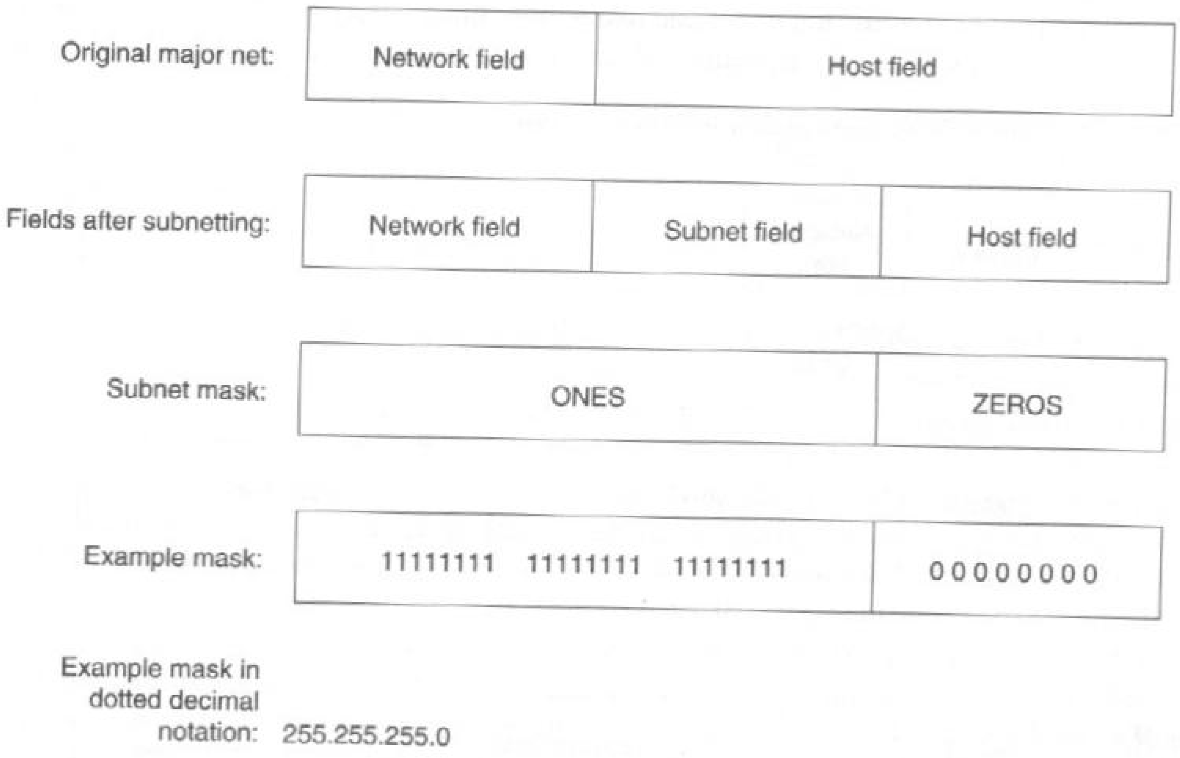
When you consume some of the bits in the host field for subnets, you are left with three fields: the original network field, a newly created subnet field, and a reduced-size host field. Figure 1-2 illustrates the three fields you get after subnetting.

Figure 1-2 *Subnetting Results in Network, Subnet, and Host Fields*


Network field
Subnet field
Host field
Number of bits determined by class
Bits stolen from host field
Host bits remaining

You declare the number of bits you are stealing from the host field with a 32-bit *subnet mask*. The subnet mask contains a contiguous series of ones that start from the left-most bit (also called the *most significant bit*). Where the ones end and the zeros begin is the boundary between the subnet field and the host field. Figure 1-3 describes a subnet mask and provides an example.

Figure 1-3 *Defining the Subnet and Host Fields with a Subnet Mask*



The example mask in Figure 1-3 has 24 one bits that start from the far left and 8 zero bits that fill out the remaining bits to the far right. This mask defines a host field of 8 bits because the boundary between the ones and the zeros is between the 24th and 25th bits (bits 25 through 32 are zero and represent the host field). The size of the subnet field depends on whether this mask is applied to a class A, class B, or class C major net. Recall from Figure 1-1 that the network field is defined by the class of the major net.

When you convert the mask from Figure 1-3 into dotted decimal notation, you get 255.255.255.0, because

- The first octet (the first group of 8 bits) is all ones (255 in decimal)
- The second octet is all ones (255 in decimal)
- The third octet is all ones (255 in decimal)
- The last octet is all zeros (0 in decimal)

The example in Figure 1-3 is a rather straightforward example because each octet is either all ones or all zeros. Things get more interesting when the boundary between the ones and zeros falls within an octet. Consider another mask:

```
11111111111111111111111111000000
```

To make this mask easier to read, separate the octets like this:

```
11111111.11111111.11111111.11000000
```

Now, convert each octet into decimal:

```
255.255.255.192
```

The preceding mask defines the subnet-host field boundary between the 26th and 27th bits, resulting in a host field of 6 bits (bits 27 through 32). Again, the size of the subnet field depends on the class of the major net to which you apply this mask. It's time for an example.

Classful Subnetting: An Example

The best way to get familiar with subnetting is to practice. Consider the following example that subnets major net 192.168.1.0 by stealing three bits from the host field to make a three-bit subnet field as shown in Example 1-1.

Example 1-1 *Subnetting a Class C Major Net with a Three-Bit Subnet Mask*

```
Major net: 192.168.1.0
Class: C
Length of original host field: 8 bits (from Figure 1-1)
Number of host bits to steal for subnet field: 3 bits
Number of host bits remaining after subnetting: 8-3=5 bits

                                     Network field
                                       192.168.1

Major net in binary:     1100-0000.1010-1000.0000-0001.0000-0000
Subnet mask in binary: 1111-1111.1111-1111.1111-1111.1110-0000

                                                        Subnet Host
                                                         field  field

Subnet mask in
dotted decimal notation: 255.255.255.224
```

The common way to write a major net together with its subnet mask is by using the shorthand notation of the major net followed by a slash (/) and the number of ones in the mask. The shorthand notation for 192.168.1.0 masked with 255.255.255.224 (see Example 1-1) is 192.168.1.0/27 (there are 27 contiguous ones in 255.255.255.224).

NOTE Both the dotted decimal and slash notations are acceptable, and both notations are used when working with Cisco routers. For example, configuring an address on a router interface requires the mask in dotted decimal notation, but the output of **show ip route** favors slash notation in most versions of IOS. Also, some people prefer one notation over the other, so a good idea is to be familiar with both.

As you can see from Example 1-1, converting from dotted-decimal notation to binary when subnetting is often convenient. A separator, such as a hyphen, makes it easier to read eight bits in a row.

Example 1-1 uses three bits for the subnet field. This yields eight unique combinations that are used to identify the subnets: 000, 001, 010, 011, 100, 101, 110, and 111. The eight subnets for Example 1-1 are listed in Table 1-2. The three bits that make up the subnet field are printed in boldface to emphasize the distinction between the subnet bits and the host bits.

Table 1-2 *The Eight Subnets for Example 1-1*

Subnet Field	Octet *x* in 192.168.1.*x* (bin)	Octet *x* in 192.168.1.*x* (dec)	Subnet Number
111	**111**0-0000	224	192.168.1.224/27
110	**110**0-0000	192	192.168.1.192/27
101	**101**0-0000	160	192.168.1.160/27
100	**100**0-0000	128	192.168.1.128/27
011	**011**0-0000	96	192.168.1.96/27
010	**010**0-0000	64	192.168.1.64/27
001	**001**0-0000	32	192.168.1.32/27
000	**000**0-0000	0	192.168.1.0/27

In traditional subnetting, you are not allowed to use the so-called *top* and *bottom* subnets. The top subnet has all ones in the subnet field and the bottom subnet contains all zeros. For the preceding example, 192.168.1.224/27 is the top subnet and 192.168.1.0/27 is the bottom subnet. This leaves the middle six subnets available for deployment, but the top and bottom subnets are wasted. The section "Using Subnet-Zero to Get Around the Rules" later in this chapter covers how you can use the bottom subnet.

Calculating the Number of Host Addresses in a Subnet

Calculating the number of hosts that can be addressed per subnet is not difficult. Each bit position can be either a one or a zero, so starting with one bit, there are two possible combinations. The number of possible combinations doubles each time you add an additional bit. Two bits yields four combinations, three bits yields eight combinations, four bits yields 16 combinations, and so on.

The formula for the number of combinations is 2^n, where n is the number of bits in the field. Example 1-1 has five bits in the host field after three bits are stolen for the subnet field. This yields $2^5=32$ unique combinations for addressing hosts; however, the all-zeros and all-ones patterns are reserved for the subnet number and subnet broadcast address, respectively. After subtracting these two reserved addresses, 30 addresses per subnet remain for host addresses.

Finding Subnet Information, Given a Host Address and the Mask

Given a host address and the subnet mask, you can determine the subnet on which that host lives. This is another common exercise and is useful anytime you need to track the subnet number for a host (in a routing table, for example). Suppose you are given the following host address and subnet mask:

```
172.16.9.136/22
```

To start the process, convert the host address and mask to binary and write the mask below the host address (for clarity, the host field bits are printed in boldface here):

```
1010-1100.0001-0000.0000-1001.1000-1000 = 172.16.9.136
1111-1111.1111-1111.1111-1100.0000-0000 = /22
```

Now, focus on the boundary defined by the mask (where the ones end and the zeros begin). This is the boundary between the subnet field and the host field and tells you that the last 10 bits of the address make up the host field. An easy way to determine the subnet number is to take the host address and set all of the bits in the *host field* to zero, like this:

```
1010-1100.0001-0000.0000-1000.0000-0000 = 172.16.8.0
```

Thus, host 172.16.9.136/22 is on subnet 172.16.8.0/22.

NOTE You might notice that the subnet number is the result of a binary "AND" operation on the address and mask at each bit position. This is how computers (and routers) calculate the subnet number.

Additionally, you can easily find the IP broadcast address for the subnet. This is done by setting all of the bits in the host field (printed again in boldface) to one, like this:

```
1010-1100.0001-0000.0000-1011.1111-1111 = 172.16.11.255
```

Thus, the broadcast address of subnet 172.16.8.0/22 is 172.16.11.255. Sending a packet (a ping, for example) to 172.16.11.255 is a transmission to every host in the subnet.

Last, you can find the range of valid host addresses for this subnet. The range contains the addresses *between* the subnet number (host field of all zeros) and the broadcast address (host field of all ones), so the host address range for subnet 172.16.8.0/22 is

```
1010-1100.0001-0000.0000-1000.0000-0001 = 172.16.8.1
```

through

```
1010-1100.0001-0000.0000-1011.1111-1110 = 172.16.11.254
```

You can verify that the host address 172.16.9.136, introduced at the start of this section, indeed falls within this address range.

Disadvantages of Subnetting

Note that subnetting is restrictive because the technique forces you to commit to the number of subnets you need now and in the future. You also need to commit to the number of hosts per subnet, because every bit you steal for the subnet field means one less bit you can use for host addresses.

Making matters worse, the technique produces subnets that are all of equal size in the number of hosts that can be supported per subnet. Therefore, you often have to do the sizing based on the largest subnet needed and waste addresses when deploying the remaining subnets to areas with fewer hosts. These issues apply when you're using a routing protocol that only supports a fixed-size mask. "Subnetting with Variable Length Subnet Masks," later in this chapter, covers a method of subnetting that mitigates some of the problems with fixed-size masks.

The Rules on Top and Bottom Subnets

Arguments exist both in theory and in practice for not using the top and bottom subnets in a classful network. Theoretically, a bit field has two special patterns:

- **All-zeros pattern**—usually means "this" as in "this host" or "this network."
- **All-ones pattern**—usually means "all" as in "all hosts" or "all networks."

Early Internet documents said it was a good idea to keep these meanings and apply them to the subnet field, thus disallowing the use of the bottom subnet of all zeros and the top subnet of all ones. As a result, IP software in devices obeyed these rules and checked if users erroneously attempted to configure a device in violation of the rules.

NOTE ✓ The advent of classless addressing abolished the notion of the top and bottom subnets (and subnets in general). In a classless environment, devices can use the address space that the classful world knows as the top and bottom subnets. See "Overview of Classless Addressing" later in this chapter for information on classless addressing.

In practice, using the top or bottom subnet can be problematic, because not all devices, especially legacy devices, allow these to be configured. Although you might be successful at deploying some hosts and routers on these outer subnets, you might find that other devices forbid you to configure an address from the top or bottom subnet. You'll then have to find another subnet for those devices. To avoid problems, a good idea is to be familiar with the diversity of devices in your environment and determine the addressing allowed on those devices.

The root of the controversy lies in the ambiguity of addresses when you're using the top or bottom subnets. Take, for example, a bottom subnet field that contains all zeros (the host field also contains all zeros)—the subnet number is the same as the major net number. This is apparent in Example 1-1, where the bottom subnet 192.168.1.0/27 is the same address as the major net (see Table 1-2). This ambiguity can be a source of confusion for some devices because a reference to the subnet is indistinguishable from a reference to the major net. Similarly, an all-ones broadcast to the top subnet could be interpreted as a broadcast address to all of the major net, because the top subnet and major net broadcasts are also indistinguishable. Looking again at the example in Table 1-2, a broadcast to the upper subnet 192.168.1.224/27 is 192.168.1.255—the same address as a broadcast to the entire class C (192.168.1.0).

Using Subnet-Zero to Get Around the Rules

Keeping in mind the caveats listed in the preceding section, you can configure Cisco routers to use the bottom subnet so that you gain one more subnet out of your subnetting efforts. To enable the use of the bottom subnet, use the **ip subnet-zero** global command:

```
Router#conf t
Router(config)#ip subnet-zero
```

If you forget to configure this, the router will "complain" when it comes time to assign an address to an interface. The following is an attempt to configure an interface with an address from a bottom subnet on a router without the **ip subnet-zero** command (notice the output **Bad mask**):

```
Router(config)#int s0
Router(config-if)#ip address 192.168.1.2 255.255.255.224
Bad mask /27 for address 192.168.1.2
```

Because the broadcast address for the top subnet is the same as the broadcast address to the entire major net, deploying the top subnet with such classful routing protocols as RIP and IGRP is not recommended. This is not a problem for classless routing protocols, such as OSPF and EIGRP.

A Word on Semantics

For the remainder of this book, the term *network* defines a general service of TCP/IP communication, as in the "corporate network" or "enterprise network." This is also known as an organization's *intranet* and is usually built of campus networks and wide-area networks. The term *major net* refers to a specific IP address space that follows classful addressing, and *subnet* refers to an address space that is extracted from the major net with the subnetting procedure covered earlier in "Subnetting a Classful Address Space."

Subnetting with Variable Length Subnet Masks

With Variable Length Subnet Masks (VLSMs), you carve an address space (such as a major net) with masks of varying lengths to design subnets of different sizes. This allows you to deploy subnets that are appropriate in size to the number of hosts you need to support in a given part of the network. As a result, you can gain efficient consumption of your address space and—depending on how you deploy the addresses—flexibility in the future as you adjust the size of each subnet to handle growth.

NOTE Your routers must be running a routing protocol that supports VLSM, such as OSPF or EIGRP. RIP and IGRP are classful routing protocols and do not support VLSM. Classful routing protocols are limited to a single subnet mask per major net.

Here is the basic technique for variably subnetting a major net:

1. Subnet the space (for example, a major net) into large address blocks based on the large subnets you need in your network.
2. Deploy these large blocks of addresses to support your large subnets.
3. Take any unused large blocks and subnet them further to support smaller subnets with fewer hosts. You can think of this as a second round of subnetting.
4. Deploy the subnets from the second round of subnetting.
5. With additional rounds of subnetting, continue dividing unused blocks of addresses into multiple smaller subnets and deploying them as needed.

Some binary is involved here. Subnetting requires that you understand and visualize binary patterns and apply those patterns to masks. Consider the following example that uses a class C major net.

Using VLSM for Address Space Efficiency: An Example

Suppose Widget, Inc., asks you to subnet one of its class C major nets and tells you it needs the following:

- Two subnets that can support at least 60 hosts
- Four subnets that can support at least 10 hosts
- As many subnets as possible that can support two hosts

The subnets are needed to support some new additions to its network, as summarized in Table 1-3.

Table 1-3 *Subnets Needed by Widget, Inc.*

Subnet Size	Quantity Needed	Purpose
60+ hosts	2	Branch offices
10+ hosts	4	Server farms
2 hosts	As many as possible (use the remaining space)	Point-to-point home offices

First, you should do a quick check of the quantity of addresses needed. The branch offices require at least 120 host addresses (60 addresses times 2 branch offices), and the server farms require at least 40 host addresses (10 addresses times 4 farms). Any remaining addresses will be used for the point-to-point home offices, but this is not a hard requirement, so the basic need is for 160 (120 plus 40) addresses. This seems to be a reasonable request, because a class C has an 8-bit host field (see Figure 1-1), and an 8-bit host field with no subnetting can support up to 254 addresses (see "Calculating the Number of Host Addresses in a Subnet" earlier in this chapter). At least Widget, Inc., is not asking for the impossible; for example, it is not asking you to support 500 addresses with a single class C.

Next, tackle the largest subnets—the subnets for the branch offices. To accommodate the branch offices, you need to subnet the class C address space into chunks of at least 60 host addresses each. This is done in the following section and represents an initial round of subnetting.

Round 1 of Subnetting

To start, you create four subnets that can support 62 hosts each. You can accomplish this by applying a 26-bit subnet mask to Widget's class C. Two of the resulting subnets will be deployed for branch offices, and the other two will be subnetted further to accommodate the other requirements. The following is Widget's class C and mask (the last octet of the mask is expanded into binary to help illustrate what's happening):

Widget, Inc.'s Major Net: 192.168.1.0 (8-bit host field)
Mask for round 1: 255.255.255.**11**00-0000 (/26 mask that supports 62 hosts per subnet)

The two bits printed in boldface represent the bits that were stolen to make a 2-bit subnet field.

Table 1-4 lists the subnets created by the first round of subnetting. The two bits that make up the subnet field are printed in boldface to emphasize the distinction between the subnet bits and the host bits.

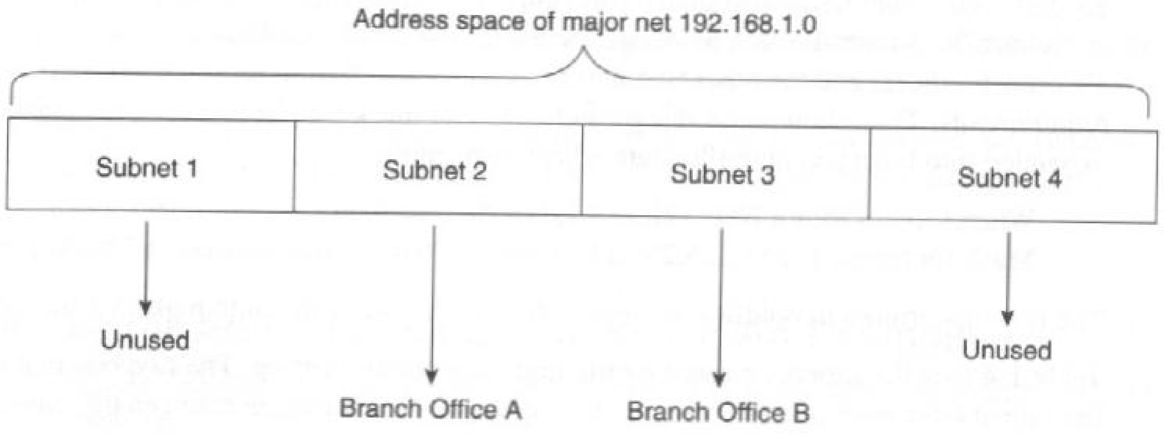
Table 1-4 *Subnets Created by the Mask for Round 1*

Name	Subnet Number in Binary (Last Octet)	Subnet Number in Decimal	Proposed Use
Subnet 1	192.168.1.**00**00-0000	192.168.1.0/26	Subnet further; see round 2
Subnet 2	192.168.1.**01**00-0000	192.168.1.64/26	Branch Office A
Subnet 3	192.168.1.**10**00-0000	192.168.1.128/26	Branch Office B
Subnet 4	192.168.1.**11**00-0000	192.168.1.192/26	Subnet further; see round 2

This first round of subnetting is nothing new—it's the same as traditional subnetting covered in "Subnetting a Classful Address Space" earlier in this chapter. Stealing two bits for the subnet field leaves six bits in the host field and yields 2^6, or 64 combinations. Subtracting the two reserved addresses for the subnet and broadcast address leaves 62 addresses for hosts. This meets Widget, Inc.'s requirement for two subnets of at least 60 hosts, so set aside Subnet 2 and Subnet 3 for the two branch offices—they are ready for deployment. Subnet 2 and Subnet 3 are selected because they are middle subnets rather than top or bottom subnets (see "The Rules on Top and Bottom Subnets" earlier in this chapter).

Figure 1-4 depicts the subnets that are set aside and unused after round 1.

Figure 1-4 *Widget, Inc.'s Address Space After Round 1 of Subnetting*



If you were doing traditional subnetting, you would now be finished, and you would have only two subnets remaining after setting aside Subnets 2 and 3. Clearly, this would not meet Widget, Inc.'s requirements, so start a second round of subnetting. This is where VLSM starts. You do not need Subnets 1 and 4 in their full size (62 host addresses), so subnet them further with a second round of subnetting and a new mask.

Round 2 of Subnetting

Perform a second round of subnetting on Subnets 1 and 4 by extending the subnet mask two bits more for a total of four bits in the mask (you are stealing two more bits from the host field and making the subnet field bigger). This further divides Subnets 1 and 4 into multiple smaller subnets.

The following is the second round of subnetting for Subnet 1. The bits printed in boldface represent the expanded subnet field (now a 4-bit field):

Subnet 1: 192.168.1.0/26 (6-bit host field)
Mask for round 2: 255.255.255.**1111**-0000 (/28 mask that supports 14 hosts per subnet)

Table 1-5 lists the new subnets created out of Subnet 1 by a second round of subnetting. For clarity, the new subnets are named Subnet 1.*x*, where *x* represents a piece of the original Subnet 1. As before, the bits that make up the subnet field are printed in boldface to emphasize the distinction between the subnet bits and the host bits. The new bits that expanded the subnet field are underlined.

Table 1-5 *Subnets Created by the Mask for Round 2 When Applied to Subnet 1*

Name	Binary (Last Octet)	Decimal	Proposed Use
Subnet 1.1	192.168.1.**0000**-0000	192.168.1.0/28	Subnet further; see round 3
Subnet 1.2	192.168.1.**0001**-0000	192.168.1.16/28	Server Farm A
Subnet 1.3	192.168.1.**0010**-0000	192.168.1.32/28	Server Farm B
Subnet 1.4	192.168.1.**0011**-0000	192.168.1.48/28	Server Farm C

NOTE Subnet 1's first two subnet bits are 00, as defined by the first round of subnetting. It is very important not to alter these two bits—any change to the 00 bits means you are no longer working with Subnet 1.

Now, perform a second round of subnetting on Subnet 4 with the same /28 mask:

Subnet 4: 192.168.1.192/26 (6-bit host field)
Mask for round 2: 255.255.255.**1111**-0000 (/28 mask that supports 14 hosts per subnet)

Table 1-6 lists the new subnets created out of Subnet 4 by a second round of subnetting. For clarity, the new subnets are named Subnet 4.*x*, where *x* represents a piece of the original Subnet 4. The new bits that expanded the subnet field are underlined.

Table 1-6 *Subnets Created by the Mask for Round 2 When Applied to Subnet 4*

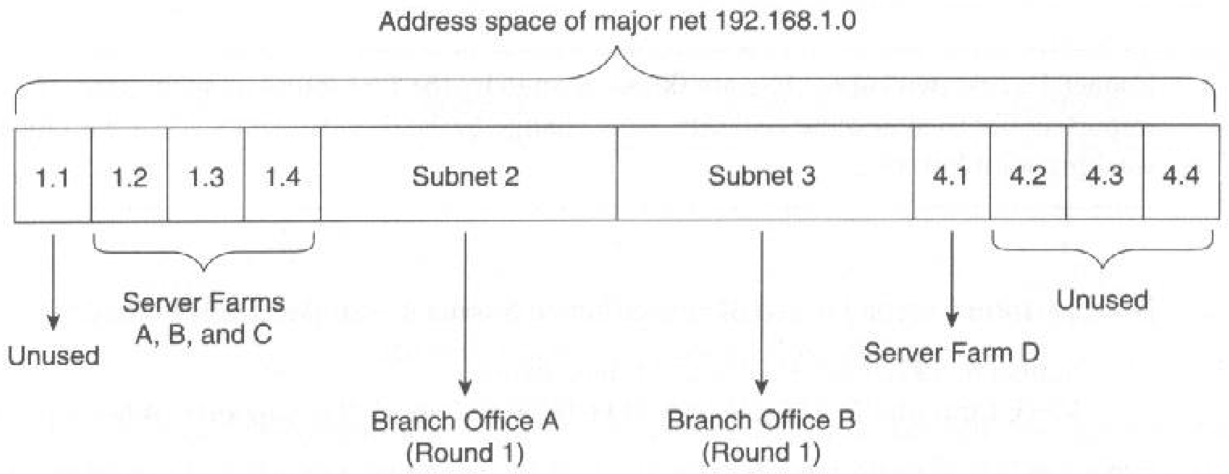
Name	Binary (Last Octet)	Decimal	Proposed Use
Subnet 4.1	192.168.1.**1100**-0000	192.168.1.192/28	Server Farm D
Subnet 4.2	192.168.1.**1101**-0000	192.168.1.208/28	Subnet further; see round 3
Subnet 4.3	192.168.1.**1110**-0000	192.168.1.224/28	Subnet further; see round 3
Subnet 4.4	192.168.1.**1111**-0000	192.168.1.240/28	Subnet further; see round 3

This second round of subnetting yields eight more subnets—eight additional subnets for Widget, Inc., out of the same address space. Each of the eight subnets (1.1 through 1.4 and 4.1 through 4.4) can support up to 14 hosts. This meets Widget, Inc.'s requirement for the server farm subnets. Widget, Inc., needs four of these subnets, so set aside Subnets 1.2, 1.3, 1.4, and 4.1 for the four server farms.

Avoid using Subnets 1.1 and 4.4, because they are the bottom and top subnets in the major net. You can deploy them if you are certain that hosts and networking devices in Widget, Inc.'s network are not affected by the caveats about using the top and bottom subnets discussed earlier.

Figure 1-5 depicts the subnets that are set aside and still unused after round 2.

Figure 1-5 *Widget, Inc.'s Address Space After Round 2 of Subnetting*



Round 3 of Subnetting

The unused subnets from round 2 can be used to satisfy Widget, Inc.'s requirement for the home office subnets (two hosts each), so now perform a third and final round of subnetting. Extend the mask from the last round by two more bits for a total of 6 bits in the mask. This further divides the unused subnets (1.1, 4.2, 4.3, and 4.4) into smaller, two-host subnets.

The following is the third round of subnetting applied to the unused Subnet 4.2 (from round 2). The bits printed in boldface represent the expanded subnet field (now a 6-bit field):

Subnet 4.2: 192.168.1.208/28 (4-bit host field)
Mask for round 3: 255.255.255.**1111-11**00 (/30 mask that supports two hosts per subnet)

Table 1-7 lists the new subnets created out of Subnet 4.2 by a third round of subnetting. For clarity, the new subnets are named Subnet 4.2.*x*, where *x* represents a piece of the Subnet 4.2. As before, the bits that make up the subnet field are printed in boldface to emphasize the distinction between the subnet bits and the host bits. The new bits that expanded the subnet field are underlined.

Table 1-7 *Subnets Created by the Mask for Round 3 When Applied to Subnet 4.2*

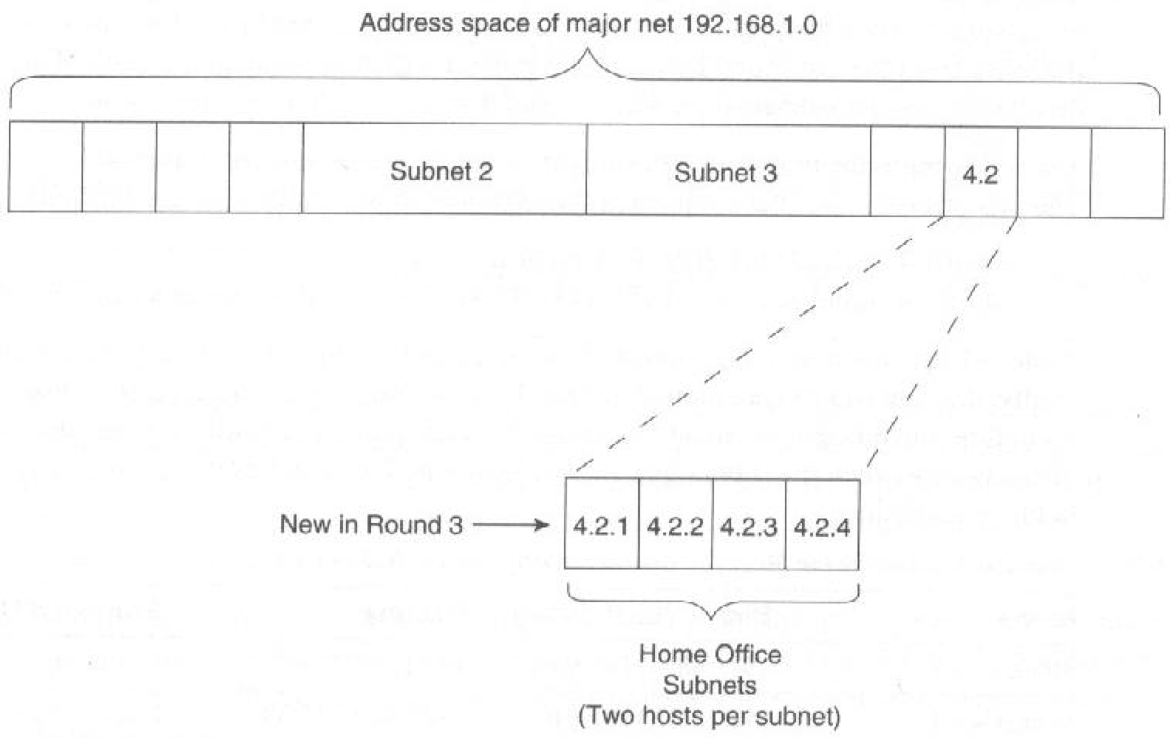
Name	Binary (Last Octet)	Decimal	Proposed Use
Subnet 4.2.1	192.168.1.**1101-00**00	192.168.1.208/30	Home Office
Subnet 4.2.2	192.168.1.**1101-01**00	192.168.1.212/30	Home Office
Subnet 4.2.3	192.168.1.**1101-10**00	192.168.1.216/30	Home Office
Subnet 4.2.4	192.168.1.**1101-11**00	192.168.1.220/30	Home Office

NOTE Subnet 4.2's first four subnet bits are 1101, as defined by the second round of subnetting. It is very important not to alter these four bits—any change to the 1101 bits means you are no longer working with Subnet 4.2.

This third round of subnetting uses a /30 mask and creates four smaller subnets out of Subnet 4.2. A subnet with a /30 mask can support only two hosts—perfect for Widget, Inc.'s home offices that connect over point-to-point links.

Figure 1-6 depicts the subnets created after subnetting Subnet 4.2 with the mask from round 3 (/30 mask).

Figure 1-6 *Subnet 4.2 After the Third Round of Subnetting*


Address space of major net 192.168.1.0
Subnet 2
Subnet 3
4.2
New in Round 3
4.2.1
4.2.2
4.2.3
4.2.4
Home Office Subnets (Two hosts per subnet)

Widget, Inc., wants to use all of the unused address space from round 2 for home offices, so with Subnet 4.2 complete (Table 1-7), simply repeat the third round of subnetting. That is, apply the same /30 mask to the other unused subnets from round 2: Subnets 1.1, 4.3, and 4.4. This results in a total of 16 two-host subnets for home offices, as summarized by Table 1-8.

Table 1-8 *A Summary of the Subnets Created by Round 3*

Name	Binary (Last Octet)	Subnet
1.1.1	192.168.1.**0000-00**00	192.168.1.0/30
1.1.2	192.168.1.**0000-01**00	192.168.1.4/30
1.1.3	192.168.1.**0000-10**00	192.168.1.8/30
1.1.4	192.168.1.**0000-11**00	192.168.1.12/30
4.2.1	192.168.1.**1101-00**00	192.168.1.208/30
4.2.2	192.168.1.**1101-01**00	192.168.1.212/30
4.2.3	192.168.1.**1101-10**00	192.168.1.216/30
4.2.4	192.168.1.**1101-11**00	192.168.1.220/30
4.3.1	192.168.1.**1110-00**00	192.168.1.224/30
4.3.2	192.168.1.**1110-01**00	192.168.1.228/30
4.3.3	192.168.1.**1110-10**00	192.168.1.232/30

Table 1-8 *A Summary of the Subnets Created by Round 3 (Continued)*

Name	Binary (Last Octet)	Subnet
4.3.4	192.168.1.**1110-11**00	192.168.1.236/30
4.4.1	192.168.1.**1111-00**00	192.168.1.240/30
4.4.2	192.168.1.**1111-01**00	192.168.1.244/30
4.4.3	192.168.1.**1111-10**00	192.168.1.248/30
4.4.4	192.168.1.**1111-11**00	192.168.1.252/30

As in the earlier rounds, you still have a top and bottom subnet after round 3; they are 192.168.1.252/30. and 192.168.1.0/30. Although these are generally not deployable, they are small two-host subnets, so you are wasting just a few addresses out of the entire major net space. The third-round VLSM process has effectively reduced the wasted address space from 128 addresses in round 1 (where Subnet 4 and Subnet 1 were the top and bottom subnets) to just 8 addresses in round 3 (where Subnets 4.4.4 and 1.1.1 are the top and bottom subnets). This represents significantly better use of the address space over fixed-length subnet masks.

Final VLSM Results for Widget, Inc.

After the third round of subnetting, you cannot use VLSM to subnet any further—a two-host subnet is the smallest you can make. The totals from all three rounds are listed in Table 1-9.

Table 1-9 *Final Results of Subnetting for Widget, Inc.*

Round	Subnets Created	Subnets Set Aside	Maximum Hosts per Subnet
1	4	2	62
2	8	4	14
3	16	14 (2 wasted)	2

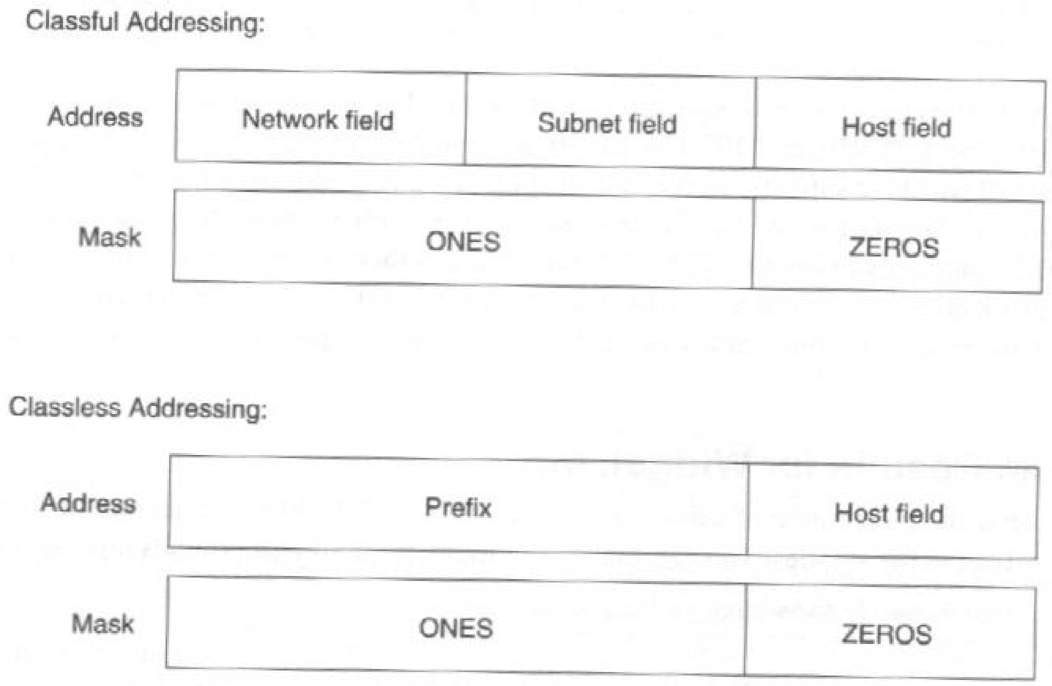
The VLSM process yields a total of 20 deployable subnets of three different sizes and meets the stated requirements of Widget, Inc.

NOTE RFC 1219 describes a VLSM subnetting strategy that allows subnets to grow in size after they are deployed and also avoids address changes. See Appendix A for information on how to retrieve RFCs.

Overview of Classless Addressing

Classless addressing (described in RFC 1519) abolishes the idea of traditional classes A, B, and C major nets and the notion of a subnet field. Subnets and major nets do not exist in a classless world; instead, there is only a network prefix and a host field. Figure 1-7 describes the difference between classful and classless addressing.

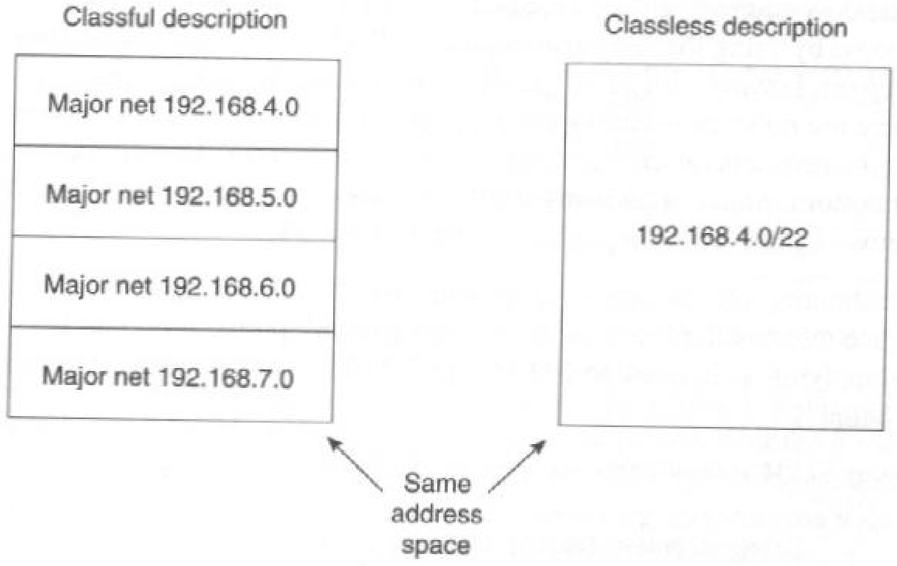
Figure 1-7 *Classful Versus Classless Addressing*



The length of the network prefix is determined by a prefix mask. The prefix mask is a contiguous series of ones that starts with the left-most bit (the most significant bit). Although the prefix mask looks like a subnet mask, it's important to realize that there is no subnet field.

An advantage of classless addressing is the capability to combine what were multiple class C addresses into a contiguous block of addresses called a *supernet* or classless interdomain routing (CIDR) block. Figure 1-8 describes an address space in two ways: as four class C major nets (classful sense) and as one supernet (classless sense).

Figure 1-8 *An Address Space Written as Four Class C Major Nets and as One Supernet*



The number after the slash in the classless notation is the prefix length and indicates how many one bits are in the prefix mask. For example, 192.168.4.0/22 represents a prefix of 192.168.4.0 with a mask of 22 contiguous ones. The mask /22 is equivalent to 255.255.252.0 in dotted decimal notation.

With the prefix and the mask, you can determine the addresses covered by the supernet 192.168.4.0/22 (see Example 1-2):

Example 1-2 *Determining the Address Range Covered by 192.168.4.0/22*

```
Prefix: 192.168.4.0  ——→ 192.168.0000-01 | 00.0000-0000
Mask: /22            ——→ 255.255.1111-11 | 00.0000-0000
Address: 192.168.0000-0100.0000-0000
                    Through
           192.168.0000-0111.1111-1111
                      (or)
         192.168.4.0 through 192.168.7.255
```

In a classless world, the address space depicted in Example 1-2 is one block and, if desired, may be deployed as one "subnet" supporting up to 1022 hosts (2^{10}, subtracting for the network prefix address itself and the all-ones pattern for the network prefix broadcast address). This demonstrates the power of classless addressing. If you had to use classful addressing, you would be stuck with four separate class C major nets—the largest subnet you could make would be 254 hosts. (A class C major net with no subnetting yields 8 bits in the host field, which is $2^8 - 2 = 254$ hosts.)

Using VLSM Techniques with Classless Addressing

Classless addressing doesn't stop there. You can also break up the space any way you choose by using the same techniques of VLSM. (It's now called *variable length network prefixes*; however, the term VLSM is still commonly used, semantics aside). Remember, there are no more subnets by the true definition of the word because there is no subnet field—only a network prefix and its prefix mask. This means there's no such thing as a top or bottom subnet, so in using the techniques of VLSM, you can use all of the possible network prefixes for deployment into the network.

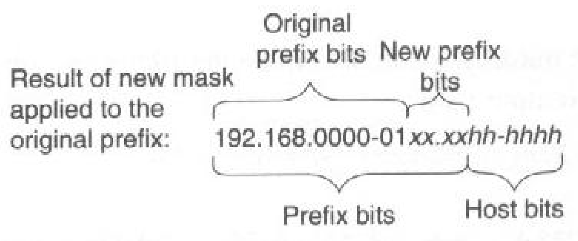
Continuing with Example 1-2, you can use VLSM techniques to divide the 192.168.4.0/22 space into smaller blocks with a longer network prefix. Example 1-3 illustrates the results of applying a /26 mask to 192.168.4.0/22 (this could be the first round of VLSM, for example).

Example 1-3 *Using VLSM to Divide Supernet 192.168.4.0/22 into Smaller Address Blocks*

```
Original prefix:  192.168.4.0/22

Original prefix
in binary:        192.168.0000-0100.00|00-0000   (/22)

New mask:         255.255.1111-1111.11|00-0000   (/26)
```


Original prefix bits
New prefix bits
Result of new mask applied to the original prefix:
192.168.0000-01xx.xxhh-hhhh
Prefix bits
Host bits

In Example 1-3, each *x* represents a bit that can be used to create new network prefixes and each *h* represents a host bit.

The new prefix mask /26 yields four bits that divide the original /22 space into 16 (2^4) smaller blocks. Each of these smaller blocks contains 6 bits for host addresses (up to 62 hosts each). Table 1-10 lists the blocks created with the /26 mask. The four new prefix bits resulting from the VLSM operation are printed in boldface for clarity.

Table 1-10 *Listing of the New Prefixes Created from Example 1-3*

Network Prefix (Subnet); *h* Represents Host Bits	Network Prefix (Subnet) in Dotted Decimal Notation
192.168.0000-01**00.00***hh-hhhh*	192.168.4.0/26
192.168.0000-01**00.01***hh-hhhh*	192.168.4.64/26
192.168.0000-01**00.10***hh-hhhh*	192.168.4.128/26
192.168.0000-01**00.11***hh-hhhh*	192.168.4.192/26

Table 1-10 *Listing of the New Prefixes Created from Example 1-3 (Continued)*

Network Prefix (Subnet); *h* Represents Host Bits	Network Prefix (Subnet) in Dotted Decimal Notation
192.168.0000-01**01**.**00***hh-hhhh*	192.168.5.0/26
192.168.0000-01**01**.**01***hh-hhhh*	192.168.5.64/26
192.168.0000-01**01**.**10***hh-hhhh*	192.168.5.128/26
...	...
192.168.0000-01**11**.**11***hh-hhhh*	192.168.7.192/26

These blocks are equal in meaning to subnets—you deploy them as you would subnets with the same VLSM strategies. Note that the new prefix mask crosses over the traditional class C boundary (the dot after the 24th bit) without concern. This again demonstrates the power and flexibility of classless addressing. Also note that during deployment, you will need to verify that all of your network devices (including hosts) support classless addressing.

NOTE The terms *subnet* and *network prefix* are often used interchangeably. Many people are familiar with subnet and prefer using it even if they are routing with VLSM and classless routing protocols. This book uses the term *subnet* when it is more descriptive than *network prefix*. Just keep in mind the semantics for any situation in which you have to adhere to strict definitions.

Routing Protocols and Classless Addressing

Having waded through all the theory and binary, consider routing protocols for a moment. To reap the benefits of classless addressing (such as supernetting), you must use a routing protocol that supports classless addressing—perhaps OSPF, EIGRP, or Border Gateway Protocol (BGP). These classless protocols carry both network prefixes and their corresponding prefix masks in routing updates.

RIP and IGRP, on the other hand, do not support classless addressing. RIP and IGRP also do not support variable length masks within a major net, because they do not carry mask information in routing updates as classful routing protocols do. Instead, RIP and IGRP assume there is a fixed subnet mask per major net, and that mask is determined from the mask that was configured on the interface.

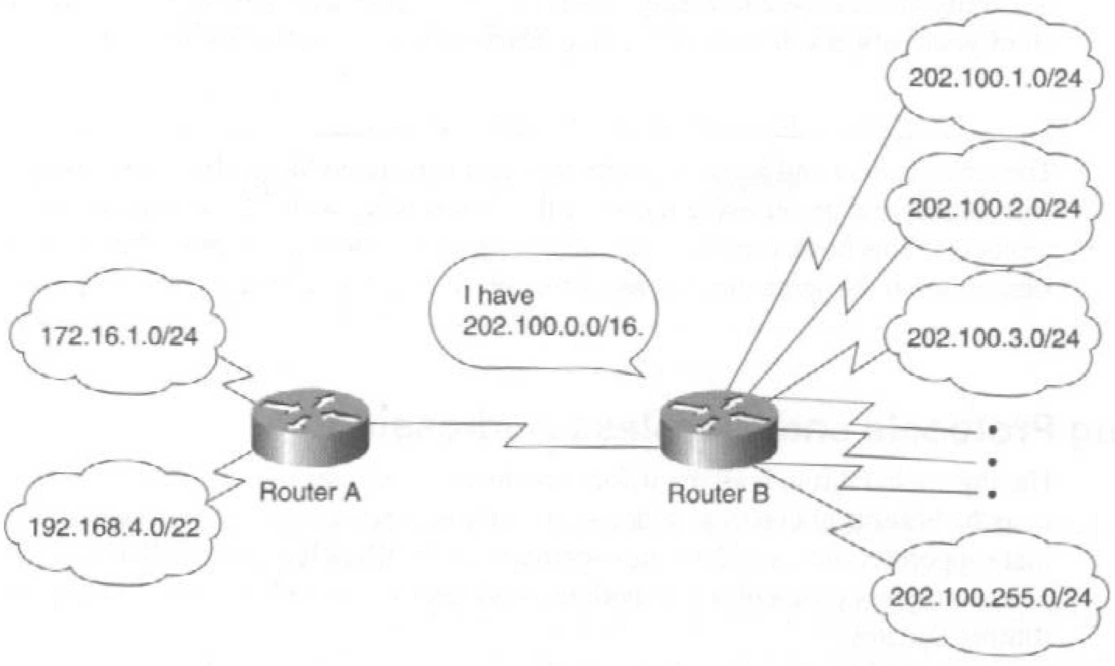
NOTE RIPv2, version 2 of RIP, supports VLSM but is less widely used than OSPF and EIGRP.

Planning for Address Summarization

In a classless world, address *summarization* (also called *aggregation*) allows a router to consolidate multiple network prefixes into a single, less specific prefix. Example 1-2 in this chapter uses a single prefix 192.168.4.0/22 to summarize the address space of four prefixes that resemble class C addresses (192.168.4.0/24, 192.168.5.0/24, 192.168.6.0/24, and 192.168.7.0/24). A router can view the address block as the four /24 prefixes or as the single /22 prefix—it's the same address space, but using the single /22 prefix is more efficient. To extend the idea further, you could summarize all prefixes that start with 192.168 with a single less specific prefix, 192.168.0.0/16. Again, a prefix is equivalent in meaning to a subnet.

Figure 1-9 illustrates an address summarization scenario.

Figure 1-9 *A Router Using Address Summarization*



In Figure 1-9, Router B advertises a summary route 202.100.0.0/16 to tell Router A that all prefixes starting with 202.100 are reachable through it. Advertising a single generalized route is more efficient than babbling 255 specific routes with a /24 mask. Router A needs to receive and process only one route—not 255 separate ones.

Summarization reduces the number of network prefixes managed and communicated between routers. With large networks, especially the Internet, managing too many specific prefixes wastes router memory and network bandwidth; therefore, if at all possible, plan for summarization by deploying addresses as contiguous groups. Then, you can use routing protocols, such as OSPF, EIGRP, or BGP, to summarize address blocks and exchange fewer and less specific routes between routers.

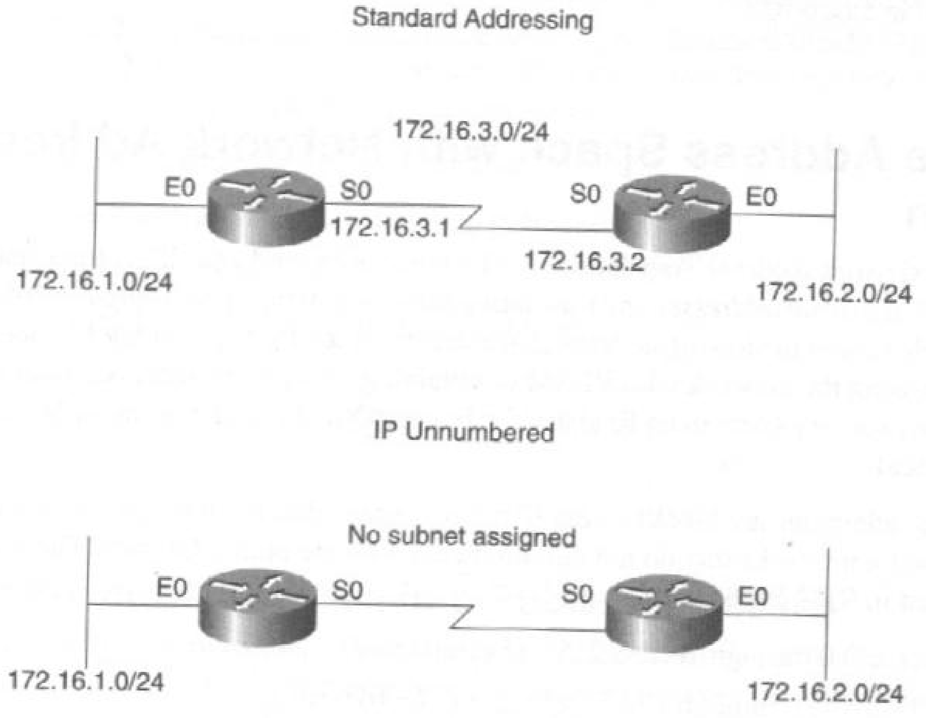
Conserving Subnets with IP Unnumbered

Typically, a link between two routers requires a subnet. With classful routing protocols, such as RIP, this is problematic because you waste a multihost subnet for just two routers. Better solutions are to use VLSM and create small, two-host subnets (255.255.255.252 or /30 subnet mask) or to use the IOS *IP unnumbered* feature.

With IP unnumbered, you can save substantial address space by deploying router links without assigned subnets. This feature is applicable to point-to-point networks between router pairs, such as point-to-point leased line, frame relay, and ATM links.

Figure 1-10 illustrates the difference between standard addressing and IP unnumbered.

Figure 1-10 *Standard Addressing Versus IP Unnumbered*



In standard addressing, you assign a subnet to each router interface. For Figure 1-10, this means the interfaces Ethernet0 (E0) and Serial0 (S0) on both routers are assigned specific subnets. One subnet, 172.16.3.0/24, exists only to connect the two routers—a waste of addresses. The subnet could be used more effectively; it could support a LAN with clients and servers, for example.

With IP unnumbered, the point-to-point serial interfaces have no assigned addresses and have no subnet between them. This would normally cause problems because a router uses the interface address as the source address for routing updates it sends out that interface. IP unnumbered resolves the problem by borrowing an address from one of the router's *other* interfaces (a LAN interface, for example) and using the borrowed address for the source address of routing updates it generates out of the unnumbered interface.

To configure IP unnumbered and designate the interface from which to borrow an address, use the **ip unnumbered** interface configuration command. The following example starts from enable mode:

```
Router#config terminal
Router(config)#interface s0
Router(config-intf)#ip unnumbered e0
```

The command **ip unnumbered e0** configures Serial0 (**s0**) as an unnumbered interface and designates the address configured on Ethernet0 (**e0**) as the borrowed address (the source address for routing updates going out the unnumbered interface Serial0).

NOTE You should be familiar with configuring Cisco routers from the IOS command line. For a quick-start tutorial on navigating around IOS and entering commands, refer to Appendix E, "A Crash Course in Cisco IOS."

Scaling the Address Space with Network Address Translation

With Network Address Translation (NAT), you can expand your IP address space by deploying so-called *private addresses* and translating them into publicly registered addresses. NAT can be a viable option in slowing address space depletion, and using it might be more feasible than redesigning the network with VLSM or obtaining new public addresses with your ISP or Internet registry (American Registry for Internet Numbers, if you are in North or South America).

Private addresses are blocks of the IP address space that the Internet community has set aside for use by networks that do not communicate with the public Internet. The address blocks are defined in RFC 1918 and include

- 10.0.0.0 through 10.255.255.255 (10.0.0.0/8)
- 172.16.0.0 through 172.31.255.255 (172.16.0.0/12)
- 192.168.0.0 through 192.168.255.255 (192.168.0.0/16)

Any organization may freely deploy these addresses without notifying the Internet registry. Thus, multiple organizations can use these addresses, each in their private networks, with the understanding that the public Internet does not route traffic to or from these addresses. This might be applicable for hosts that do not need to communicate over the Internet and have no intention to communicate over the Internet in the future (private computer labs are an example). These addresses are deployed within the organization just as any ordinary IP address space, and the same subnetting rules and VLSM techniques apply to these private addresses as to normal public addresses.

Public addresses, on the other hand, are administered by the Internet registry and are routable by the Internet. Every public address is unique (no two hosts on the Internet have the same IP address) and has a registered owner if it's in use. A host addressed with a public address can communicate with hosts both inside the organization and outside on the Internet.

If everyone could have as many public addresses as they want, there would be no need to use private addresses. But the Internet has a finite number of addresses, and getting a share of the public addresses can become difficult as more scrutiny and tighter control are used to determine who gets them. Private addresses are readily available for use, but they come with the big disadvantage that they cannot be used to communicate over the Internet. What you want is the best of both worlds: use of the private address space *and* the ability to communicate over the Internet. This requires a way to translate addresses as they flow between the private and public domains—which is where NAT comes into play.

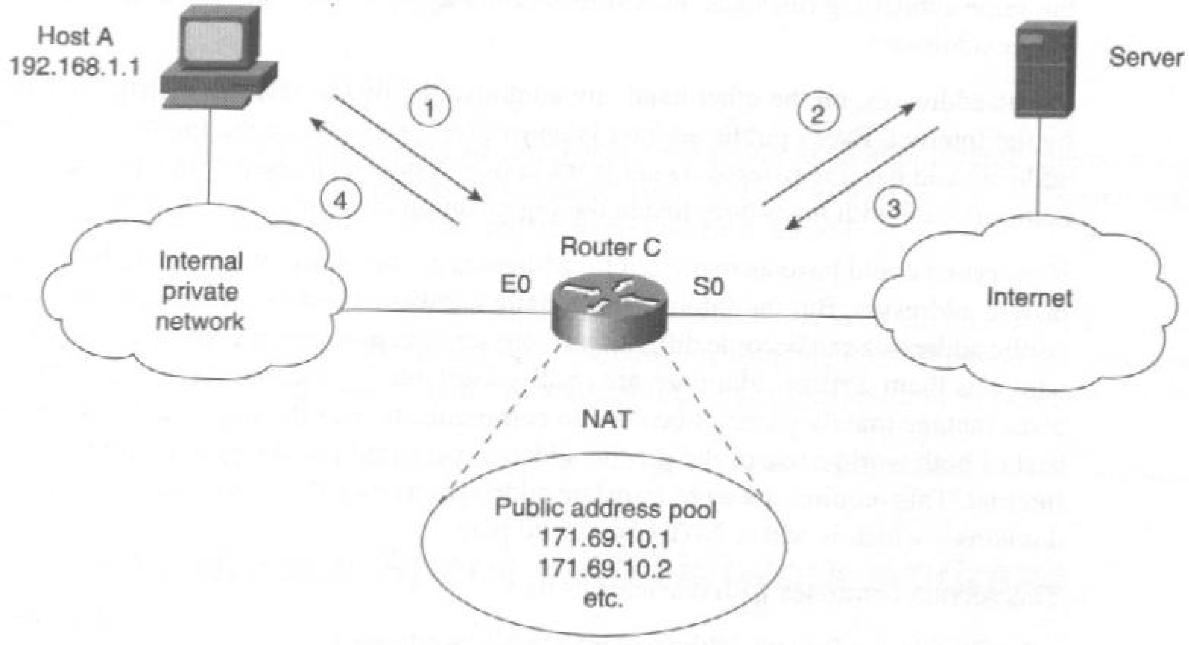
This section continues with discussions on

- Translating Private Addresses into Public Addresses
- Configuring NAT
- Creating a Pool of Discontiguous Addresses
- Configuring Static NAT
- Special Applications and NAT
- More Important Points on NAT

Translating Private Addresses into Public Addresses

Cisco routers can dynamically translate private addresses into public addresses, allowing hosts with private addresses to communicate with hosts on the Internet without modification. That is, the privately addressed hosts can function as if they are connected to the Internet. You can configure a router to maintain a pool of public addresses that is smaller than the population of privately addressed hosts. The router then manages the pool and dynamically translates private addresses into public addresses as necessary for communicating with the Internet. Hosts on the Internet have no idea they are communicating with a privately addressed host; they communicate with legitimate public addresses from the router's pool. Figure 1-11 shows an example of NAT in action.

Figure 1-11 *Router C Performing NAT for Host A*



① Packet from Host A: source=192.168.1.1 (private address)
② Packet from Host A: source=171.69.10.1 (public address)
③ Packet to Host A: destination=171.69.10.1 (public address)
④ Packet to Host A: destination=192.168.1.1 (private address)

In Figure 1-11, privately addressed Host A needs to communicate with a server on the Internet. The following sequence describes a round-trip NAT operation, starting with Host A's initial packet (refer to the numbered arrows in Figure 1-11):

1 **From Host A (source = 192.168.1.1, private)**—Host A's traffic gets routed through the internal network and arrives at the edge router that connects to the Internet, Router C. The source address of the packet is 192.168.1.1. Router C detects that Host A's packets are sourced from a private address and require address translation. The router looks in its pool of public addresses and selects an available address, 171.69.10.1, to use for translating packets to and from Host A.

2 **From Host A (source = 171.69.10.1, public)**—Next, the router translates the outgoing packets. For the original private source address (192.168.1.1), it substitutes public address 171.69.10.1 and sends the modified packets to the Internet. The Internet routes Host A's modified packets to the server (Host A's intended destination).

3 **To Host A (destination = 171.69.10.1, public)**—The server responds to 171.69.10.1, unaware that Host A's address is really 192.168.1.1. The Internet routes the packets from the server to Router C, the keeper and originator of the address 171.69.10.1.

4 **To Host A (destination = 192.168.1.1, private)**—Packets from the server arrive at Router C, which translates 171.69.10.1 (now the destination address) back to 192.168.1.1 and forwards the traffic to the internal network. The internal network routes the traffic to Host A, completing the two-way communication between Host A and the server.

The NAT router (Router C) maintains an idle timer such that if Host A stops sending packets to the Internet for a certain period of time, the router expires the address and returns it to the pool to be used by other hosts. The length of the idle timer is configurable.

Now for some definitions:

- **Inside local address**—The address of the privately addressed host. In the preceding example, 192.168.1.1 is the address of Host A, so it's the inside local address.
- **Inside global addresses**—The pool of legitimate public addresses.
- **Outside global address**—The address of the server on the Internet.

Familiarity with these terms is important when you're configuring and verifying NAT. This will be apparent in the next sections.

Configuring NAT

Consider the following NAT configuration for Router C of Figure 1-11 (for brevity, only the NAT-specific lines are listed):

```
hostname RTC
!
ip nat pool mypool 171.69.10.1 171.69.10.254 prefix-length 24
ip nat inside source list 2 pool mypool overload
!
interface Serial0
ip nat outside
!
interface Ethernet0
ip nat inside
!
access-list 2 permit 192.168.1.0 0.0.0.255
```

The line **ip nat pool mypool 171.69.10.1 171.69.10.254 prefix-length 24** creates the pool of addresses for NAT—the inside global addresses. This pool contains 254 addresses, from 171.69.10.1 to 171.69.10.254. These addresses are legal, public addresses that the router will substitute for the private addresses (inside local addresses).

The line **ip nat inside source list 2 pool mypool overload** configures the router to translate internal private addresses that match access list 2 (configured in a following line), using the pool **mypool** that was created in the preceding line. Internal traffic that does not match access list 2 will not be translated and will be routed normally.

The **overload** keyword means the router may use a single public address to represent multiple privately addressed hosts. This, in effect, multiplexes many private addresses over one public address. Overload might be needed if the public address pool is exhausted of any available addresses because there are many active translations. With overload, the router uses unique TCP and UDP port numbers to differentiate multiple private hosts. Because over 64,000 TCP/UDP port numbers are available per address, you can theoretically support tens of thousands of private hosts with a single IP address; however, you will likely reach practical limits before that.

NOTE You can create a pool with just one address and use the **overload** keyword. This enables you to translate many private addresses by using a single IP address. That one address in the pool may also be an IP address belonging to one of the router's interfaces.

The line **ip nat outside** is configured in interface configuration mode for the serial interface (Serial0). This tells the router that this interface faces the publicly addressed world. In most cases, this points to the public Internet.

The line **ip nat inside** tells the router that the ethernet interface (Ethernet0) faces the internal network. This is where our privately addressed hosts are: the hosts that need translation to communicate with the Internet.

The line **access-list 2 permit 192.168.1.0 0.0.0.255** creates an access list numbered 2 that defines the hosts that need translation. This access list is used by the previous command, **ip nat inside source list 2 pool mypool**. The router identifies packets to and from inside local addresses by matching the access list criteria, allocates addresses from **mypool**, and translates the addresses as it passes packets between the internal network and the Internet. For information on how to configure access lists and the syntax used, see Chapter 6, "Deploying Basic Security Services."

Instead of using an access list, you can use a route map to trigger translation based on such information as next-hop address and outbound interface. To do this, use the command **ip nat inside source route-map** instead of **ip nat inside source list**. This can be particularly useful if you are connected to two ISPs and want to use different pools for each ISP. See Chapter 3 for information on route maps (covered under policy routing).

Creating a Pool of Discontiguous Addresses

You might need to exclude some addresses from a pool of inside global addresses (for static addresses assigned to hosts or routers, for example). The following example configuration creates a pool of discontiguous addresses:

```
2509(config)#ip nat pool testpool prefix-length 24
2509(config-ipnat-pool)#address 171.69.1.1 171.69.1.4
2509(config-ipnat-pool)#address 171.69.1.6 171.69.1.10
2509(config-ipnat-pool)#exit
```

The preceding commands create a pool called **testpool** that contains addresses 171.69.1.1 through 171.69.1.4 and 171.69.1.6 through 171.69.1.10 (it skips 171.69.1.5). The addresses in the pool have a prefix-length of 24 bits as defined by the keywords **prefix-length 24**.

Configuring Static NAT

You can configure some private addresses for *static* translation, such that they are always translated by using the same public IP address. This could be useful for a privately addressed host that has to be reachable from the Internet with a public address that remains constant. Here is an example configuration of static translation:

```
2509(config)#ip nat inside source static 192.168.1.2 171.69.5.2
```

The preceding command configures a static translation for a private host (192.168.1.2). NAT will translate the private address to and from the public address 171.69.5.2.

Special Applications and NAT

For most traffic, NAT only changes the source and destination addresses in the IP header and does not inspect or modify the data payload contained in the packet. Therefore, applications that carry source or destination IP addresses in the payload of the packet might fail to work because the IP header will be changed by NAT but the payload will be left unchanged. Aware that this could be a problem, Cisco has made and continues to make enhancements to NAT so it can inspect data payloads and support applications that are sensitive to translation. Contact Cisco and get the most recent list of these supported applications (enhancements to NAT occur with each software release). At the time of this writing, H.323, RealAudio, VDOLive, Vxtreme, CuSeeMe (White Pine), NetBIOS over TCP/IP, NFS, rlogin, rsh, rcp, and FTP are supported. Web (http), Telnet, NTP, and other applications that do not carry addresses in the data payload work fine with NAT—they do not require inspection of the data payload.

More Important Points on NAT

The following are some additional notes on NAT of which you should be aware:

- If no available addresses exist in the NAT pool because all are in use, NAT is not able to support any more translations. In this situation, the router drops all packets it cannot translate and sends an Internet Control Message Protocol (ICMP) "Host Unreachable" message back to the privately addressed host. To remedy this, you can try one or more of the following measures:
  - — Use the overload option.
  - — Increase the size of the NAT pool.
  - — Decrease the NAT timers so addresses are returned to the pool more often.

- Privately addressed hosts and publicly addressed hosts can coexist in your network, and you can configure the router to translate addresses for the privately addressed hosts only.
- NAT is not restricted to translating RFC 1918 private addresses. It can also be used to translate IP addresses that were deployed "illegally"—that is, public addresses that are used within an organization but the organization is not the registered owner of those addresses. This might have been done at a time when the organization had never planned to connect to the Internet and probably before reserved private addresses were defined by RFC 1918 in 1996.
- Your organization has the responsibility to filter privately addressed routes so they don't get advertised to the Internet by your routing protocols. Route filtering is covered in Chapter 3.
- If you are translating many concurrent hosts and find NAT causes too much load on your router, you might investigate using dedicated NAT hardware such as Cisco's PIX firewall.
- NAT hides the identity of the internal hosts for which it is translating; therefore, it enhances security to a degree. This in no way substitutes for the security of a full-featured firewall, but it can be a favorable by-product of NAT.
- RFC 1631 also describes NAT.

Summary

This chapter covered a range of IP addressing information from basic definitions to more sophisticated services such as NAT. Developing a plan that makes efficient use of your address space is an important step in building an IP network that meets today's requirements and scales to the future.

The following are the key concepts of this chapter:

- The subnetting procedure divides a major net into smaller subnets by stealing host bits for a new field called the subnet field.
- Traditionally, use of the top and bottom subnets was forbidden. Legacy hosts and networking devices might not let you configure an address from these edge subnets, so be familiar with your organization's installed base if you plan to deploy them.
- The command **ip subnet-zero** enables you to configure a router with an address from the bottom subnet.
- VLSM gives you more flexibility in how you define subnets and more efficient use of your address space.
- Classless routing protocols, such as OSPF and EIGRP, support VLSM.

- Classless addressing abolishes the idea of traditional class A, class B, and class C major nets and the notion of a subnet field.
- A supernet is a contiguous block of addresses that spans the traditional boundaries of classful addressing.
- VLSM techniques also apply to classless address blocks.
- Address summarization improves the scalability of routing protocols by consolidating multiple prefixes into a single, less specific prefix.
- The IP unnumbered feature conserves addresses by enabling point-to-point router links to operate without a subnet.
- RFC 1918 defines three blocks of private addresses that may be used freely without registration. These addresses, however, are not routed on the public Internet.
- NAT can be used to scale your address space by translating private addresses into legitimate public Internet addresses.