2

Internet Trends

2.1 Introduction

This chapter considers how data networking and the Internet have changed since their inception. The chapter begins with a brief history of the Internet that highlights some of the early motivations. It describes a shift in emphasis from sharing centralized facilities to fully distributed information systems.

Later chapters in this part of the text continue the discussion by examining specific Internet applications. In addition to describing the communication paradigms available on the Internet, the chapters explain the programming interface that Internet applications use to communicate.

2.2 Resource Sharing

Early computer networks were designed when computers were large and expensive, and the main motivation was *resource sharing*. For example, networks were devised to connect multiple users, each with a screen and keyboard, to a large centralized computer. Later networks allowed multiple users to share peripheral devices such as printers. The point is:

> *Early computer networks were designed to permit sharing of expensive, centralized resources.*

In the 1960s, the *Advanced Research Projects Agency* (*ARPA*†), an agency of the U.S. Department of Defense, was especially interested in finding ways to share resources. Researchers needed powerful computers, and computers were incredibly expensive. The ARPA budget was insufficient to fund many computers. Thus, ARPA began investigating data networking — instead of buying a computer for each project, ARPA planned to interconnect all computers with a data network and devise software that would allow a researcher to use whichever computer was best suited to perform a given task.

ARPA gathered some of the best minds available, focused them on networking research, and hired contractors to turn the designs into a working system called the *ARPANET*. The research turned out to be revolutionary. The research team chose to follow an approach known as *packet switching* that became the basis for data networks and the Internet‡. ARPA continued the project by funding the Internet research project. During the 1980s, the Internet expanded as a research effort, and during the 1990s, the Internet became a commercial success.

2.3 Growth Of The Internet

In less than 30 years, the Internet has grown from an early research prototype connecting a handful of sites to a global communication system that extends to all countries of the world. The rate of growth has been phenomenal. Figure 2.1 illustrates the growth with a graph of the number of computers attached to the Internet as a function of the years from 1981 through 2008.

The graph in Figure 2.1 uses a linear scale in which the y-axis represents values from zero through five hundred fifty million. Linear plots can be deceptive because they hide small details. For example, the graph hides details about early Internet growth, making it appear that the Internet did not start to grow until approximately 1994 and that the majority of growth occurred in the last few years. In fact, the average rate of new computers added to the Internet reached more than one per second in 1998, and has accelerated. In 2007, more than two computers were added to the Internet each second. To understand the early growth rate, look at the plot in Figure 2.2, which uses a log scale.

†At various times, the agency has included the word *Defense*, and used the acronym *DARPA*.
‡Chapter 13 discusses packet switching.

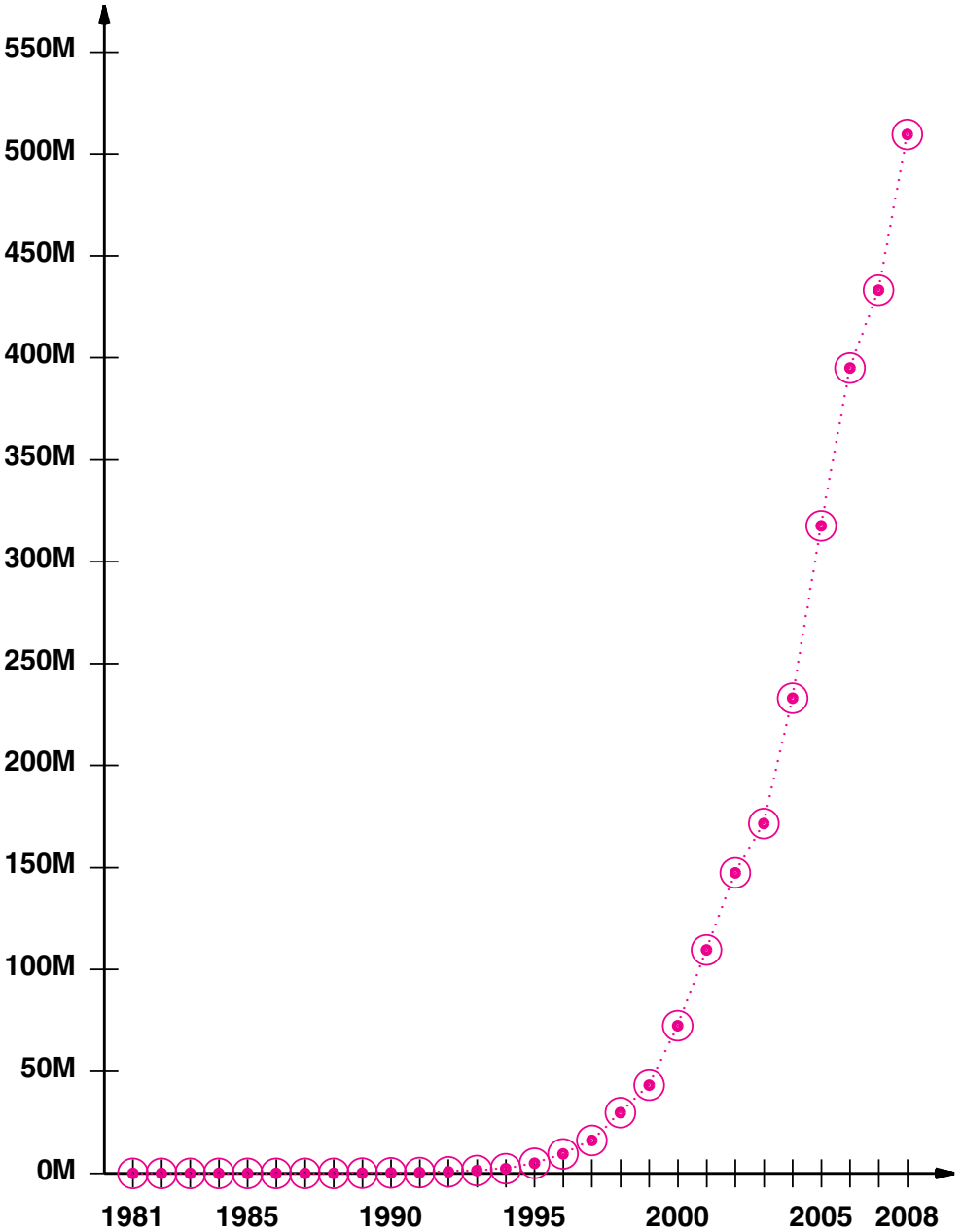


Figure 2.1 Internet growth plotted as the number of computers on the Internet.

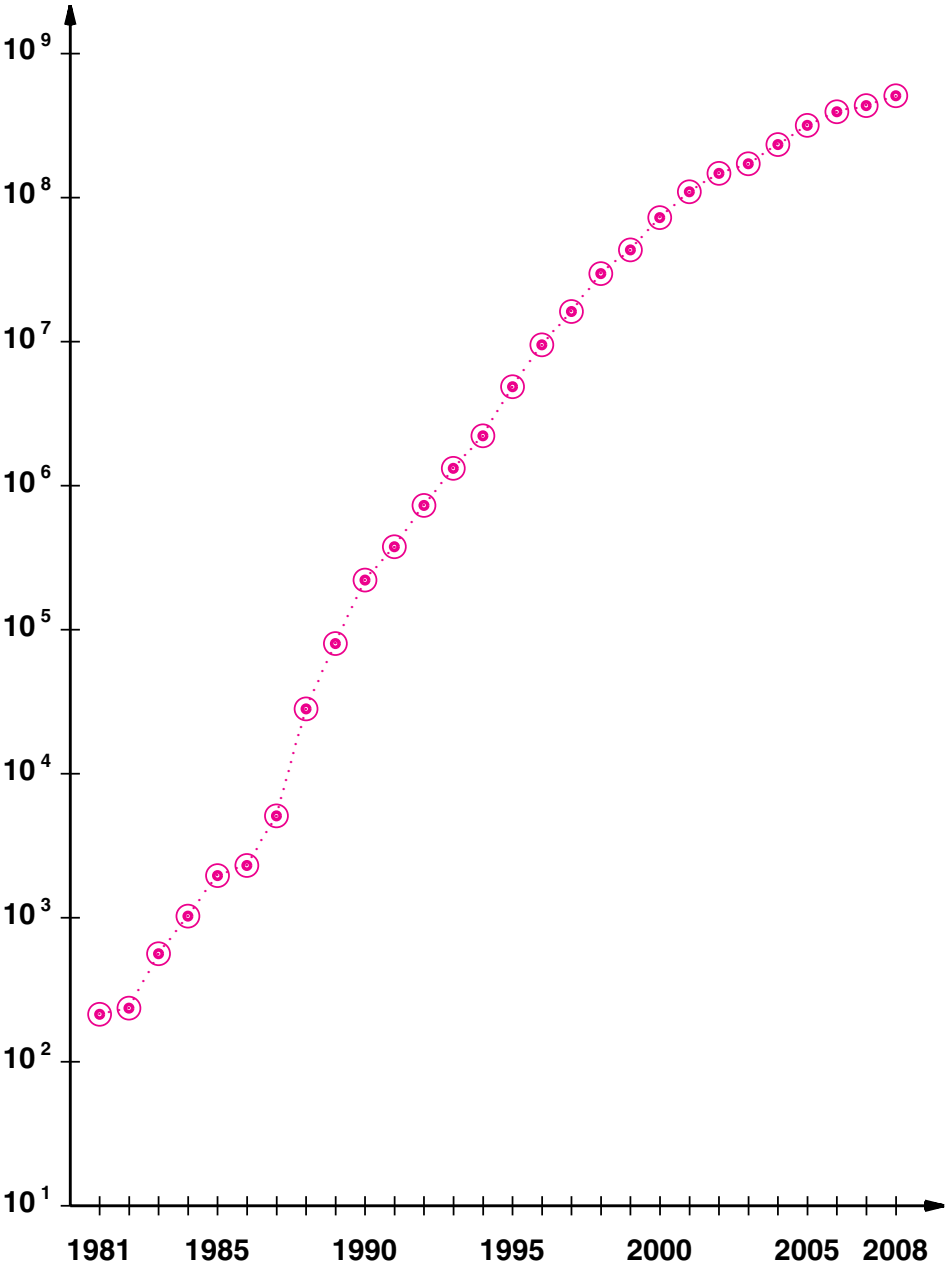


Figure 2.2 Internet growth plotted on a log scale.

21

IP: Internet Addressing

21.1 Introduction

The previous chapter explains the physical architecture of the Internet in which routers interconnect physical networks. This chapter begins a description of protocol software that makes the Internet appear to be a single, seamless communication system. The chapter introduces the addressing scheme used by the *Internet Protocol* (*IPv4*), and discusses the use of address masks for classless and subnet addressing†.

The next chapters expand the description of IP. They each consider one aspect of the protocol in detail. Taken as a group, the chapters define the IP protocol and explain how IP software allows computers to exchange packets across the Internet.

21.2 Addresses For The Virtual Internet

Recall from Chapter 20 that the goal of internetworking is to provide a seamless communication system. To achieve the goal, protocol software must hide the details of physical networks and offer the illusion of a single, large network. From the point of view of an application, the virtual Internet operates like any network, allowing computers to send and receive packets. The chief difference between the Internet and a physical network is that the Internet is an abstraction imagined by its designers and created entirely by protocol software. Thus, the designers chose addresses, packet formats, and delivery techniques independent of the details of the underlying hardware.

Addressing is a critical component of the Internet abstraction. To give the appearance of a single network, all host computers must use a uniform addressing scheme, and each address must be unique. Although each computer has a MAC address, such ad-

†Unless otherwise noted, *Internet Protocol* and *IP* refer to version 4 of IP throughout the text.

dresses do not suffice because the Internet can include multiple network technologies and each technology defines its own MAC addresses.

To guarantee uniform addressing, IP defines an addressing scheme that is independent of the underlying MAC addresses. IP addresses are used as destinations in the Internet analogous to the way MAC addresses are used as destinations on a LAN. To send a packet across the Internet, the sender places the destination's IP address in the packet, and passes the packet to IP protocol software for forwarding. IP protocol software uses the destination IP address when it forwards the packet across the Internet to the destination computer.

The advantage of IP addressing lies in uniformity: an arbitrary pair of application programs can communicate without knowing the type of network hardware or MAC addresses being used. The illusion is so complete that some users are surprised to learn that IP addresses are supplied by protocol software and are not part of the underlying network. Interestingly, we will learn that many layers of protocol software use IP addresses. To summarize:

To provide uniform addressing in the Internet, IP defines an abstract addressing scheme that assigns each host a unique protocol address; applications use IP addresses to communicate.

21.3 The IP Addressing Scheme

The IP standard specifies that each host is assigned a unique 32-bit number known as the host's *Internet Protocol address*, *IP address*, or *Internet address*†. When sending a packet across the Internet, the sender must specify its own 32-bit IP address (the source address) as well as the address of the intended recipient (the destination address).

To summarize:

An Internet address (IP address) is a unique 32-bit binary number assigned to a host and used for all communication with the host.

21.4 The IP Address Hierarchy

Analogous to the hierarchical addressing using with WANs, each 32-bit IP address is divided into two parts: a prefix and a suffix. Instead of identifying a packet switch, an IP prefix identifies the physical network to which the host is attached. An IP suffix identifies a specific computer on the network. That is, each physical network in the Internet is assigned a unique *network number*. The network number appears as a prefix in the IP address of each computer attached to the network, and each computer on a given physical network is assigned a unique suffix.

†The three terms are used as interchangeable synonyms.

To guarantee uniqueness, no two networks in the Internet can be assigned the same network number and no two computers on a given network can be assigned the same suffix. For example, if an internet contains three networks, they might be assigned network numbers *1*, *2*, and *3*. Three computers attached to network *1* can be assigned suffixes *1*, *3*, and *5*, while three computers attached to network *2* can be assigned suffixes *1*, *2*, and *3*. The assigned values do not need to be contiguous.

The important point is that the IP address scheme guarantees two properties:

- Each computer is assigned a unique address (i.e., a single address is never assigned to more than one computer).
- Although network number assignments must be coordinated globally, suffixes can be assigned locally without global coordination.

The first property is guaranteed because an IP address contains both a prefix and a suffix. If two computers are attached to different physical networks, the prefixes assigned to their addresses will differ. If two computers are attached to the same physical network, their addresses have different suffixes. Thus, the address assigned to a computer is unique.

21.5 Original Classes Of IP Addresses

Once they chose a size for IP addresses and decided to divide each address into two parts, the designers of IP had to determine how many bits to place in each part. The prefix needs sufficient bits to allow a unique network number to be assigned to each physical network in the Internet. The suffix needs sufficient bits to permit each computer attached to a network to be assigned a unique suffix. No simple choice was possible because adding bits to one part means subtracting bits from the other. Choosing a large prefix accommodates many networks, but limits the size of each network; choosing a large suffix means each physical network can contain many computers, but limits the total number of networks.

Because the Internet includes arbitrary network technologies, it contains a few large physical networks and many small networks. Consequently, the designers chose an addressing scheme to accommodate a combination of large and small networks. The original scheme, which is known as *classful IP addressing*, divided the IP address space into three primary *classes*, where each class has a different size prefix and suffix.

The first four bits of an address determined the class to which the address belonged, and specified how the remainder of the address was divided into prefix and suffix. Figure 21.1 illustrates the five address classes, the leading bits used to identify each class, and the division into prefix and suffix. The figure follows the convention used in TCP/IP protocols of numbering bits from left to right and using zero for the first bit.

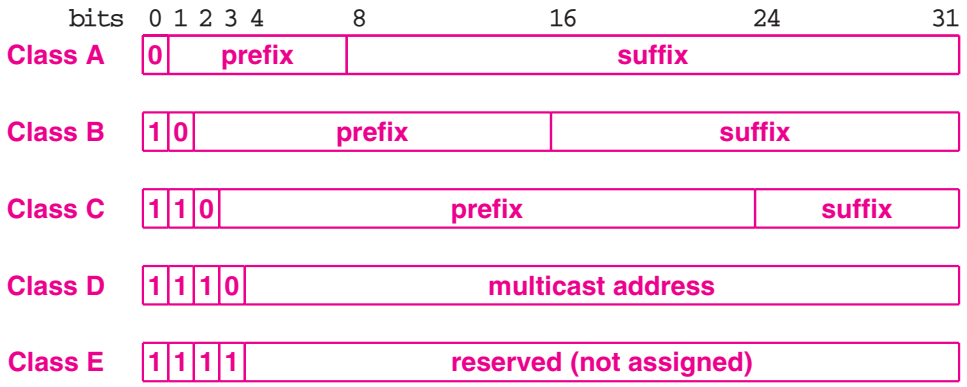


Figure 21.1 The five classes of IP addresses in the original classful scheme.

Although the classful scheme has been superseded, class *D* addresses are still used for multicasting, which allows delivery to a set of computers. Each multicast address corresponds to a group of computers. Once a multicast group has been established, a copy of any packet sent to the multicast address will be delivered to each host in the group. In practice, Internet multicasting has never been available globally, which means that multicasting is restricted to individual sites.

We can summarize:

The original IP addressing scheme divided addresses into classes. Class D addresses are still used for multicasting, but multicasting does not work globally.

21.6 Dotted Decimal Notation

Although IP addresses are 32-bit numbers, users do not enter or read the values in binary. Instead, when interacting with a user, software uses a notation that is more convenient for humans to understand. Called *dotted decimal notation*, the form expresses each 8-bit section of a 32-bit number as a decimal value and uses periods to separate the sections. Figure 21.2 illustrates examples of binary numbers and the equivalent dotted decimal notation.

32-bit Binary Number	Equivalent Dotted Decimal
10000001 00110100 00000110 00000000	129.52.6.0
11000000 00000101 00110000 00000011	192.5.48.3
00001010 00000010 00000000 00100101	10.2.0.37
10000000 00001010 00000010 00000011	128.10.2.3
10000000 10000000 11111111 00000000	128.128.255.0

Figure 21.2 Examples of 32-bit binary numbers and their equivalent in dotted decimal notation.

Dotted decimal treats each *octet* (each 8-bit value) as an unsigned binary integer†. As the final example in the figure shows, the smallest possible value, *0*, occurs when all bits of an octet are zero, and the largest possible value, *255*, occurs when all bits of an octet are one. Thus, dotted decimal addresses range from *0.0.0.0* through *255.255.255.255*. Multicast addresses, class D, occupy the range from 224.0.0.0 through 239.255.255.255.

To summarize:

> *Dotted decimal notation is a syntactic form that IP software uses to express 32-bit binary values when interacting with humans. Dotted decimal represents each octet in decimal and uses a dot to separate octets.*

21.7 Division Of The Address Space

The original classful scheme, which was devised before the PC was invented, before LANs were widely available, and before most companies had a computer network, divided the address space into unequal sizes. The designers chose an unequal division to accommodate a variety of scenarios. For example, although it is limited to 128 networks, class *A* contains half of all addresses. The motivation was to allow major ISPs to each deploy a large network that connected millions of computers. Similarly, the motivation for class C was to allow an organization to have a few computers connected on a LAN. Figure 21.3 summarizes the maximum number of networks available in each class and the maximum number of hosts per network.

†IP uses the term *octet* rather than *byte* because the size of a byte depends on the computer. Thus, although 8-bit bytes have become a de facto standard, octet is unambiguous.

Address Class	Bits In Prefix	Maximum Number of Networks	Bits In Suffix	Maximum Number Of Hosts Per Network
A	7	128	24	16777216
B	14	16384	16	65536
C	21	2097152	8	256

Figure 21.3 The number of networks and hosts per network in each of the original three primary IP address classes.

21.8 Authority For Addresses

Each prefix assigned to an individual network in the Internet must be unique. Therefore a central organization, the *Internet Corporation for Assigned Names and Numbers* (*ICANN*), has been established to handle address assignment and adjudicate disputes. ICANN does not assign individual prefixes. Instead, ICANN authorizes a set of *registrars* to do so. Registrars make blocks of addresses available to ISPs, which provide addresses to subscribers. Thus, to obtain a prefix, a corporation usually contacts an ISP†.

21.9 Subnet And Classless Addressing

As the Internet grew, the original classful addressing scheme became a limitation. Everyone demanded a class A or B address so they would have enough addresses for future growth; many addresses were unused. Although many class C addresses remained, few groups wanted them.

Two new mechanisms were invented to overcome the limitation:

- Subnet addressing
- Classless addressing

The two mechanisms are so closely related that they can be considered to be part of a single abstraction: instead of having three distinct address classes, allow the division between prefix and suffix to occur on an arbitrary bit boundary. Subnet addressing was initially used within large organizations that attached to the global Internet, and classless addressing extended the approach to the entire Internet.

To understand the motivation for using an arbitrary boundary, consider an ISP that hands out prefixes. Suppose a customer of the ISP requests a prefix for a network that contains thirty-five hosts. When classful addressing was used, the ISP would assign a

†Chapter 23 explains how a computer obtains a unique suffix.

class C prefix. In fact, only four bits of host suffix are needed to represent all possible host values, which meant that 219 of the 254 possible suffixes would never be assigned to hosts†. In other words, most of the class C address space is wasted. Classless addressing provides a better solution by allowing the ISP to assign a prefix that is twenty-six bits long. Thus, the suffix is six bits long, meaning that only twenty-seven addresses will be unused.

Another way to look at the situation is to assume the ISP owns a class C prefix. Classful addressing assigns the entire prefix to one organization. With classless addressing, however, the ISP can divided the prefix into several longer prefixes, and assign each to a subscriber. Figure 21.4 illustrates how classless addressing allows an ISP to divide a class C prefix into four longer prefixes that each accommodate a network of up to sixty-two hosts.

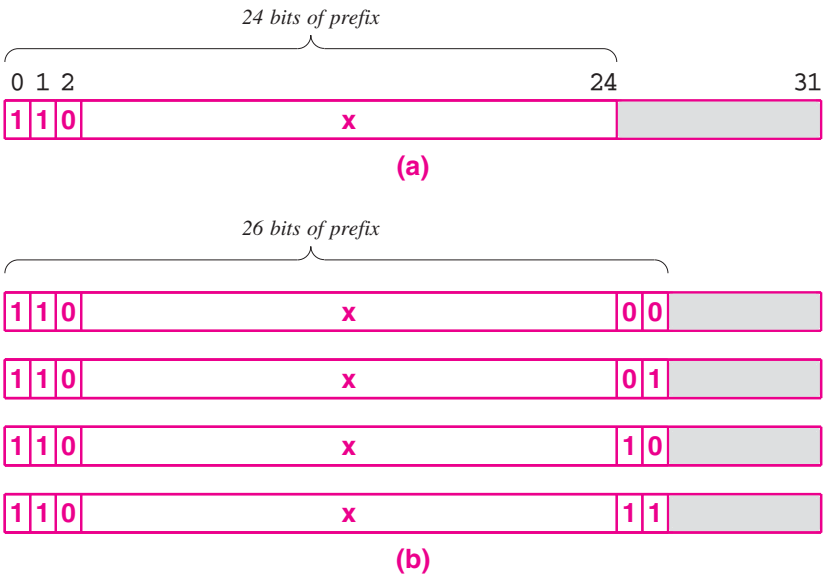


Figure 21.4 (a) A class C prefix, and (b) the same prefix divided into four classless prefixes.

In the figure, the host portion of each prefix is shown in gray. The original class C address has eight bits of suffix, and each of the classless addresses has six bits of suffix. Assuming that the original class C prefix was unique, each of the classless prefixes will also be unique. Thus, instead of wasting addresses, the ISP can assign each of the four classless prefixes to a subscriber.

†The number 254 arises because a class C address has 256 possible suffixes and the all 0s and all 1s suffixes are reserved for subnet broadcast as described later in the chapter.

21.10 Address Masks

How can an IP address be divided at an arbitrary boundary? The classless and subnet addressing schemes require hosts and routers that process addresses to store an additional piece of information: a value that specifies the exact boundary between the network prefix and the host suffix. To mark the boundary, IP uses a 32-bit value known as an *address mask*, which was originally called a *subnet mask*. An address mask has one bits to mark the network prefix and zero bits to mark the host portion.

Why store the boundary size as a bit mask? A mask makes processing efficient. In particular, we will see that when they handle an IP packet, hosts and routers need to compare the network prefix portion of the address to a value in their forwarding tables. The bit-mask representation makes the comparison efficient. To see how, suppose a router is given a destination address, *D*, a network prefix represented as a 32-bit value, *N*, and a 32-bit address mask, *M*. That is, assume the top bits of *N* contain a network prefix, and the remaining bits have been set to zero. To test whether the destination lies on the specified network, the router tests the condition:

$$N == (D \ \& \ M)$$

That is, the router uses the mask with a "logical and" operation to set the host bits of address *D* to zero, and then compares the result with the network prefix *N*.

As an example, consider the following 32-bit network prefix:

10000000 00001010 00000000 00000000

which has the dotted decimal value *128.10.0.0*. Also consider a 32-bit mask that has sixteen one bits followed by 16 zero bits, which can be denoted in dotted decimal as *255.255.0.0*:

11111111 11111111 00000000 00000000

Now consider a 32-bit destination address *128.10.2.3*, which has a binary equivalent of:

10000000 00001010 00000010 00000011

A logical *and* between the destination address and the address mask extracts the high-order sixteen bits, which produces the binary result:

10000000 00001010 00000000 00000000

which is equal to the network prefix *128.10.0.0*.

21.11 CIDR Notation

The classless addressing scheme is formally known as *Classless Inter-Domain Routing* (*CIDR*). The name is unfortunate because CIDR only specifies addressing and forwarding. When the CIDR addressing scheme was created, the designers wanted to make it easy for a human to specify a mask. To understand the difficulty, consider the mask needed for the example in Figure 21.4b, which has twenty-six 1 bits followed by six 0 bits. In dotted decimal, the mask is:

255.255.255.192

To make it easier for humans to specify and interpret mask values, dotted decimal notation was extended. In the extended version, which is known as *CIDR notation*, an address and a mask can be specified by giving a dotted decimal address followed by a slash and a decimal number that specifies the number of contiguous, left-justified one bits in the mask. That is, the general form is:

ddd.ddd.ddd.ddd/m

where *ddd* is the decimal value for an octet of the address, and *m* is the number of one bits in the mask. Thus, one might write the following:

192.5.48.69/26

which specifies a mask of 26 bits. Figure 21.5 lists address masks in CIDR notation along with the dotted decimal equivalent of each. Note that some of the CIDR address masks correspond to the original classful assignments.

21.12 A CIDR Example

As an example of CIDR, assume an ISP has the following address block available to assign:

128.211.0.0/16

Further suppose the ISP has two customers, one customer needs twelve IP addresses and the other customer needs nine. The ISP can assign one customer CIDR prefix:

128.211.0.16/28

and can assign the other customer:

128.211.0.32/28

Length (CIDR)	Address Mask	Notes
/0	0 . 0 . 0 . 0	All 0s (equivalent to no mask)
/1	128 . 0 . 0 . 0	
/2	192 . 0 . 0 . 0	
/3	224 . 0 . 0 . 0	
/4	240 . 0 . 0 . 0	
/5	248 . 0 . 0 . 0	
/6	252 . 0 . 0 . 0	
/7	254 . 0 . 0 . 0	
/8	255 . 0 . 0 . 0	Original Class A mask
/9	255 . 128 . 0 . 0	
/10	255 . 192 . 0 . 0	
/11	255 . 224 . 0 . 0	
/12	255 . 240 . 0 . 0	
/13	255 . 248 . 0 . 0	
/14	255 . 252 . 0 . 0	
/15	255 . 254 . 0 . 0	
/16	255 . 255 . 0 . 0	Original Class B mask
/17	255 . 255 . 128 . 0	
/18	255 . 255 . 192 . 0	
/19	255 . 255 . 224 . 0	
/20	255 . 255 . 240 . 0	
/21	255 . 255 . 248 . 0	
/22	255 . 255 . 252 . 0	
/23	255 . 255 . 254 . 0	
/24	255 . 255 . 255 . 0	Original Class C mask
/25	255 . 255 . 255 . 128	
/26	255 . 255 . 255 . 192	
/27	255 . 255 . 255 . 224	
/28	255 . 255 . 255 . 240	
/29	255 . 255 . 255 . 248	
/30	255 . 255 . 255 . 252	
/31	255 . 255 . 255 . 254	
/32	255 . 255 . 255 . 255	All 1s (host specific mask)

Figure 21.5 A list of address masks in CIDR notation and in dotted decimal.

Although both customers have the same mask size (28 bits), the prefixes differ. The binary value assigned to one customer is:

10000000 11010011 00000000 0001 0000

and the binary value assigned to the other customer is:

10000000 11010011 00000000 0010 0000

Thus, there is no ambiguity — each customer has a unique prefix. More important, the ISP retains most of the original address block, which it can allocate to other customers.

21.13 CIDR Host Addresses

Consider computing the range of addresses in a CIDR block. Once an ISP assigns a customer a CIDR prefix, the customer can assign host addresses. For example, suppose an organization is assigned *128.211.0.16/28* as described above. Figure 21.6 illustrates that the organization will have four bits to use as a host address field, and shows the highest and lowest addresses in both binary and dotted decimal. The example avoids assigning the all 1s and all 0s host addresses.

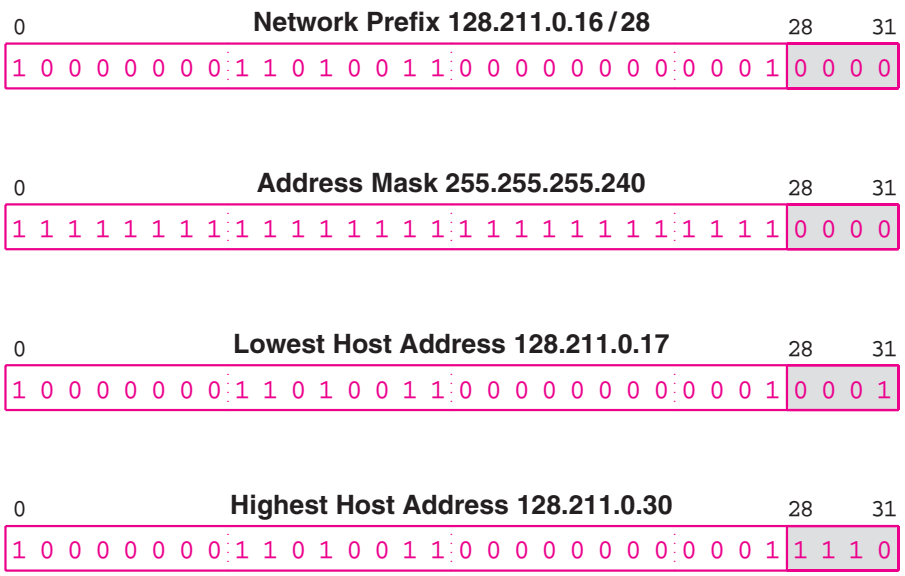


Figure 21.6 Illustration of CIDR addressing for an example /28 prefix.

Figure 21.6 illustrates a disadvantage of classless addressing — because the host suffix can start on an arbitrary boundary, values are not easy to read in dotted decimal. For example, when combined with the network prefix, the fourteen possible host suffixes result in dotted decimal values from *128.211.0.17* through *128.211.0.30*.

21.14 Special IP Addresses

In addition to assigning an address to each computer, it is convenient to have addresses that can be used to denote networks or sets of computers. IP defines a set of special address forms that are *reserved*. That is, special addresses are never assigned to hosts. This section describes both the syntax and semantics of each special address form.

21.14.1 Network Address

One of the motivations for defining special address forms can be seen in Figure 21.6 — it is convenient to have an address that can be used to denote the prefix assigned to a given network. IP reserves host address zero, and uses it to denote a *network*. Thus, the address *128.211.0.16/28* denotes a network because the bits beyond the 28th are zero. A network address should never appear as the destination address in a packet†.

21.14.2 Directed Broadcast Address

Sometimes, it is convenient to send a copy of a packet to all hosts on a physical network. To simplify broadcasting, IP defines a *directed broadcast address* for each physical network. When a packet is sent to a network's directed broadcast address, a single copy of the packet travels across the Internet until it reaches the specified network. The packet is then delivered to all hosts on the network.

The directed broadcast address for a network is formed by adding a suffix that consists of all *1* bits to the network prefix. Thus, the host suffix that consists of all *1* bits is reserved — if an administrator inadvertently assigns the all-ones suffix to a specific computer, software may malfunction.

How does broadcast work? If network hardware supports broadcast, a directed broadcast will be delivered using the hardware broadcast capability. If a particular network does not have hardware support for broadcast, software must send a separate copy of the packet to each host on the network.

†Section 21.16 discusses the Berkeley broadcast address form, which is a nonstandard exception.

21.14.3 Limited Broadcast Address

The term *limited broadcast* refers to a broadcast on a directly-connected network; informally, we say that the broadcast is limited to a "single wire". Limited broadcast is used during system startup by a computer that does not yet know the network number.

IP reserves the address consisting of thirty-two *1* bits to refer to limited broadcast. Thus, IP will broadcast any packet sent to the all-ones address across the local network.

21.14.4 This Computer Address

Because each Internet packet contains the address of the source as well as the destination, a computer needs to know its IP address before it can send or receive Internet packets. In Chapter 23, we will learn that TCP/IP contains protocols a computer can use to obtain its IP address automatically when the computer boots. Interestingly, the startup protocols use IP to communicate. When using such startup protocols, a computer cannot supply a correct IP source address. To handle such cases, IP reserves the address that consists of all zeroes to mean *this computer*.

21.14.5 Loopback Address

IP defines a *loopback address* used to test network applications. Programmers often use loopback for preliminary debugging after a network application has been created. To perform a loopback test, a programmer must have two application programs that are intended to communicate across a network. Each application includes the code needed to interact with TCP/IP protocol software. Instead of executing each program on a separate computer, the programmer runs both programs on a single computer and instructs them to use a loopback address when communicating. When one application sends data to another, data travels down the protocol stack to the IP software, which forwards it back up through the protocol stack to the second program. Thus, the programmer can test the program logic quickly without needing two computers and without sending packets across a network.

IP reserves the network prefix *127/8* for use with loopback. The host address used with *127* is irrelevant — all host addresses are treated the same. By convention, programmers often use host number *1*, making *127.0.0.1* the most popular loopback address.

During loopback testing no packets ever leave a computer — the IP software forwards packets from one application program to another. Consequently, the loopback address never appears in a packet traveling across a network.

21.15 Summary Of Special IP Addresses

The table in Figure 21.7 summarizes the special IP address forms.

Prefix	Suffix	Type Of Address	Purpose
all-0s	all-0s	this computer	used during bootstrap
network	all-0s	network	identifies a network
network	all-1s	directed broadcast	broadcast on specified net
all-1s	all-1s	limited broadcast	broadcast on local net
127/8	any	loopback	testing

Figure 21.7 Summary of the special IP address forms.

We said that special addresses are reserved and should never be assigned to host computers. Furthermore, each special address is restricted to certain uses. For example, a broadcast address must never appear as a source address, and the all-*0*s address must not be used after a host completes the startup procedure and has obtained an IP address.

†BSD stands for the *Berkeley Software Distribution*.

21.17 Routers And The IP Addressing Principle

In addition to assigning an Internet address to each host, the Internet Protocol specifies that routers should be assigned IP addresses as well. In fact, each router is assigned two or more IP addresses, one for each network to which the router attaches. To understand why, recall two facts:

- A router has connections to multiple physical networks.
- Each IP address contains a prefix that specifies a physical network.

Thus, a single IP address does not suffice for a router because each router connects to multiple networks and each network has a unique prefix. The IP scheme can be explained by a fundamental principle:

> *An IP address does not identify a specific computer. Instead, each IP address identifies a connection between a computer and a network. A computer with multiple network connections (e.g., a router) must be assigned one IP address for each connection.*

Figure 21.8 illustrates the idea with an example that shows IP addresses assigned to two routers that connect three networks.

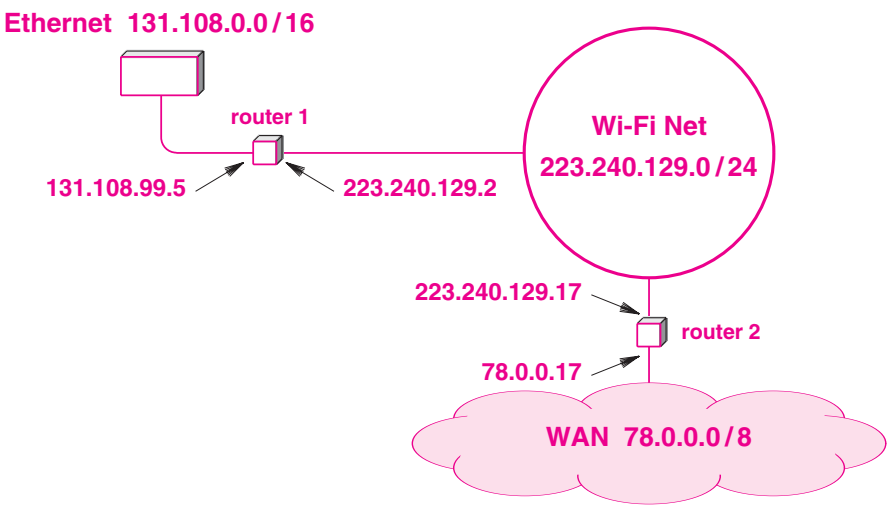


Figure 21.8 An example of IP addresses assigned to two routers.

IP does not require that the same suffix be assigned to all the interfaces of a router. In the figure, for example, the router connecting the Ethernet and Wi-Fi network has suffixes *99.5* (connection to the Ethernet) and *2* (connection to the Wi-Fi network). However, IP does not prevent using the same suffix for all connections. Thus, the example shows that the administrator has chosen to use the same suffix, *17*, for both interfaces of the router that connects the Wi-Fi network to the WAN. As a practical matter, using the same suffix can help humans who manage the networks because a single number is easier to remember.

21.18 Multi-Homed Hosts

Can a host connect to multiple networks? Yes. A host computer with multiple network connections is said to be *multi-homed*. Multi-homing is sometimes used to increase reliability — if one network fails, the host can still reach the Internet through the second connection. Alternatively, multi-homing is used to increase performance — connections to multiple networks can make it possible to send traffic directly and avoid routers, which are sometimes congested. Like a router, a multi-homed host has multiple protocol addresses, one for each network connection.

21.19 Summary

To give the appearance of a large, seamless network, the Internet uses a uniform addressing scheme. Each computer is assigned a unique IP address; all Internet applications use the address when communicating with the computer.

The Internet Protocol specifies addressing. IP divides each Internet address into a two-level hierarchy: a prefix identifies the network to which a computer attaches, and a suffix identifies a specific computer on the network. To ensure that addresses remain unique throughout a given internet, a central authority assigns network prefixes. Once a prefix has been assigned, a local network administrator assigns each host on the network a unique suffix.

An IP address is a 32 bit number. The original addressing scheme divided addresses into classes; the multicast class is still used. Classless and subnet addressing allow the boundary between prefix and suffix to occur on an arbitrary bit boundary. To do so, subnet and classless addressing (CIDR) store a 32-bit mask along with each address. The mask has value *1* for each bit in the prefix, and value *0* for each bit in the suffix.

The IP standard specifies a set of reserved addresses that have special meaning. Special addresses can be used to specify loopback (used for testing), the address of a network, broadcast on the local physical network, and broadcast on a remote network.

Although it is convenient to think of an IP address as specifying a computer, each IP address identifies a connection between a computer and a network.

22

Datagram Forwarding

22.1 Introduction

Previous chapters describe the architecture of the Internet and Internet addressing. This chapter discusses the fundamental communication service in the Internet. It describes the format of packets that are sent across the Internet, and discusses the key concepts of datagram encapsulation, forwarding, and fragmentation and reassembly. Later chapters extend the discussion by considering additional protocols that form a complete service.

22.2 Connectionless Service

The goal of internetworking is to provide a packet communication system that allows a program running on one computer to send data to a program running on another computer. In a well-designed internet, application programs remain unaware of the underlying physical networks — they can send and receive data without knowing the details of the local network to which a computer connects, the remote network to which the destination connects, or the interconnection between the two.

One of the fundamental questions that must be considered when designing an internet concerns the services that will be offered. In particular, designers must decide whether to offer a *connection-oriented* service, a *connectionless* service, or both.

TCP/IP designers chose to include protocols for both connectionless and connection-oriented service. They chose to make the fundamental delivery service connectionless, and to add a reliable connection-oriented service that uses the underlying connectionless service. The design was successful, and forms the basis for all Internet communication.

22.3 Virtual Packets

Connectionless service is a straightforward extension of packet switching — the service allows a sender to transmit individual packets of data across the Internet. Each packet travels independently, and contains information that identifies the intended recipient.

How does a packet pass across the Internet? In general, the answer is that routers handle most of the forwarding. A host creates a packet, places the destination address in the packet header, and then sends the packet to a nearby router. When a router receives a packet, the router uses the destination address to select the next router on the path to the destination, and then forwards the packet. Eventually, the packet reaches a router that can deliver the packet to its final destination.

What format is used for an Internet packet? Because the Internet consists of heterogeneous networks that use incompatible frame formats, the Internet cannot adopt any of the hardware frame formats. More important, a router cannot simply reformat the frame header because the two networks may use incompatible addressing (e.g., the addresses in an incoming frame may make no sense on another network).

To overcome heterogeneity, the Internet Protocol defines a packet format that is independent of the underlying hardware. The result is a *universal*, *virtual* packet that can be transferred across the underlying hardware intact. As the term *virtual* implies, the Internet packet format is not tied directly to any hardware. In fact, the underlying hardware does not understand or recognize an Internet packet. As the term *universal* implies, each host or router in the Internet contains protocol software that recognizes Internet packets. We can summarize:

Because it includes incompatible networks, the Internet cannot adopt a particular hardware packet format. To accommodate heterogeneity, the Internet Protocol defines a hardware-independent packet format.

22.4 The IP Datagram

TCP/IP protocols use the name *IP datagram* to refer to an Internet packet. Surprisingly, an IP datagram has the same general format as a hardware frame: the datagram begins with a header followed by a data (or *payload*) area. Figure 22.1 illustrates the datagram format.

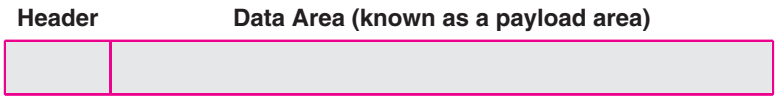


Figure 22.1 The general form of an IP datagram with a header followed by a payload.

To summarize:

A packet sent across a TCP/IP internet is called an IP datagram. Each datagram consists of a header followed by data area, which is known as the payload.

The amount of data carried in a datagram is not fixed. A sender chooses an amount of data that is appropriate to a particular purpose. For example, an application that transmits keystrokes across a network can place each keystroke in a separate datagram, while an application that transfers large files can send large datagrams. The point is:

The size of a datagram is determined by the application that sends data. Allowing the size of datagrams to vary makes IP adaptable to a variety of applications.

In the current version of the Internet Protocol (IP version 4), a datagram can contain as little as a single octet of data or at most 64K octets, including the header. In most datagrams, the header is much smaller than the payload. A header represents overhead — because the size of the datagram header is fixed, sending large datagrams results in more data octets transmitted per unit of time (i.e., higher throughput).

22.5 The IP Datagram Header Format

What does a datagram header contain? Similar to a frame header, a datagram header contains information used to forward the datagram. In particular, the header contains the address of the source (the original sender), the address of the destination (the ultimate recipient), and a field that specifies the type of data being carried in the payload area. Each address in the header is an IP address; MAC addresses for the sender and recipient do not appear in the datagram.

Each field in an IP datagram header has a fixed size, which makes header processing efficient. Figure 22.2 shows the fields of an IP datagram header, and the subsequent text describes each field.

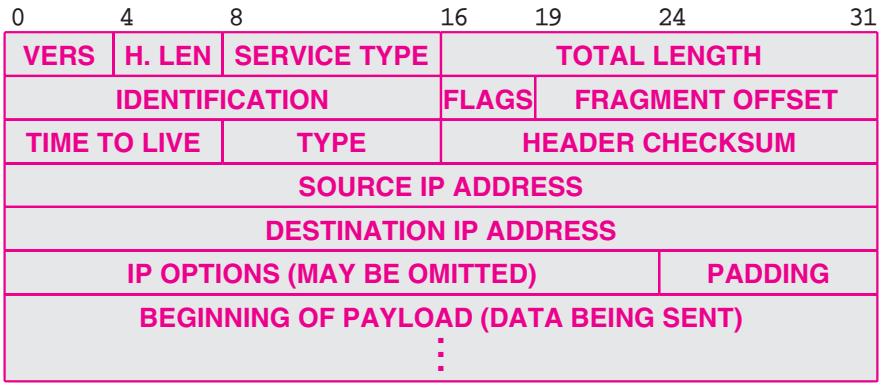


Figure 22.2 Fields in the IP version 4 datagram header.

VERS. Each datagram begins with a 4-bit protocol version number (the figure shows a version *4* header).

H.LEN. The 4-bit header length field specifies the number of 32-bit quantities in the header. If no options are present, the value is 5.

SERVICE TYPE. An 8-bit field that carries a class of service for the datagram (seldom used in practice). Chapter 28 explains the DiffServ interpretation of the service type field.

TOTAL LENGTH. A 16-bit integer that specifies the total number of bytes in the datagram, including both the header and the data.

IDENTIFICATION. A unique 16-bit number (usually sequential) assigned to the datagram that is used to gather all fragments for reassembly.

FLAGS. A 3-bit field with individual bits specifying whether the datagram is a fragment and if so, whether the fragment corresponds to the rightmost piece of the original datagram.

FRAGMENT OFFSET. A 13-bit field that specifies where in the original datagram the data in this fragment belongs. The value of the field is multiplied by eight to obtain an offset.

TIME TO LIVE. An 8-bit integer initialized by the original sender and decremented by each router that processes the datagram. If the value reaches zero, the datagram is discarded and an error message is sent back to the source.

TYPE. An 8-bit field that specifies the type of the payload.

HEADER CHECKSUM. a 16-bit ones-complement checksum of header fields computed according to Algorithm 8.1†.

SOURCE IP ADDRESS. The 32-bit Internet address of the original sender (the addresses of intermediate routers do not appear in the header).

†Algorithm 8.1 can be found on page 144.

DESTINATION IP ADDRESS. The 32-bit Internet address of the ultimate destination. The addresses of intermediate routers do not appear in the header.

IP OPTIONS. Optional header fields used to control routing and datagram processing. Most datagrams do not contain any options, which means the IP OPTIONS field is omitted from the header.

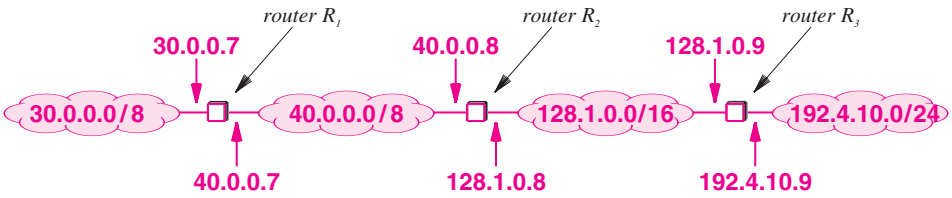
PADDING. If options do not end on a 32-bit boundary, zero bits of padding are added to make the header a multiple of 32 bits.

22.6 Forwarding An IP Datagram

We said that a datagram traverses the Internet by following a path from its initial source through routers to the final destination. The Internet uses next-hop forwarding. Each router along the path receives the datagram, extracts the destination address from the header, and uses the destination address to determine a next hop to which the datagram should be sent. The router then forwards the datagram to the next hop, either the final destination or another router.

To make the selection of a next hop efficient, an IP router uses a *forwarding table*. A forwarding table is initialized when the router boots, and must be updated if the topology changes or hardware fails.

Conceptually, the forwarding table contains a set of entries that each specify a destination and the next hop used to reach that destination. Figure 22.3 shows an example internet and the contents of a forwarding table in one of the three routers that are used to interconnect the networks.



(a)

Destination	Mask	Next Hop
30.0.0.0	255.0.0.0	40.0.0.7
40.0.0.0	255.0.0.0	deliver direct
128.1.0.0	255.255.0.0	deliver direct
192.4.10.0	255.255.255.0	128.1.0.9

(b)

Figure 22.3 (a) An example internet with four networks, and (b) the forwarding table found in router R_2.

In the figure, each router has been assigned two IP addresses, one for each interface. Router R_2, which connects directly to networks *40.0.0.0/8* and *128.1.0.0/16*, has been assigned addresses *40.0.0.8* and *128.1.0.8*. Recall that IP does not require the suffix to be the same on all interfaces — a network administrator has chosen the same suffix for each interface to make it easier for humans who manage the network.

The important point to note is the forwarding table size, which is crucial in the global Internet:

> *Because each destination in a forwarding table corresponds to a network, the number of entries in a forwarding table is proportional to the number of networks in the Internet, not the number of hosts.*

22.7 Network Prefix Extraction And Datagram Forwarding

The process of using a forwarding table to select a next hop for a given datagram is called *forwarding*. Recall from Chapter 21 that the *mask* field in a forwarding table entry is used to extract the network portion of an address during lookup. When a router encounters a datagram with destination IP address *D*, the forwarding function must find an entry in the forwarding table that specifies a next hop for *D*. To do so, the software examines each entry in the table by using the mask in the entry to extract a prefix of address *D* and comparing the resulting prefix to the *Destination* field of the entry. If the two are equal, the datagram will be forwarded to the *Next Hop* in the entry.

The bit mask representation makes extraction efficient — the computation consists of a Boolean *and* between the mask and destination address, *D*. Thus, the computation to examine the i^{th} entry in the table can be expressed as:

if ((Mask[i] & D) == Destination[i]) forward to NextHop[i];

As an example, consider a datagram destined for address *192.4.10.3*, and assume the datagram arrives at the center router, R_2, in Figure 22.3. Further assume the forwarding procedure searches entries of the table in order. The first entry fails because *255.0.0.0 & 192.4.10.3* is not equal to *30.0.0.0*. After rejecting the second and third entries in the table, the routing software eventually chooses next hop *128.1.0.9* because

255.255.255.0 & 192.4.10.3 == 192.4.10.0

22.8 Longest Prefix Match

Figure 22.3 contains a trivial example. In practice, Internet forwarding tables can be extremely large, and the forwarding algorithm is complex. For example, analogous to WAN forwarding described in Chapter 18, Internet forwarding tables can contain a *default* entry that provides a path for all destinations that are not explicitly listed. In addition, Internet forwarding allows a manager to specify a *host-specific route* that directs traffic destined to a specific host along a different path than traffic for other hosts on the same network (i.e., to specify a forwarding table entry with a 32-bit mask, which requires the entire host address to match).

An important feature of Internet forwarding arises because address masks can overlap. For example, suppose a router's forwarding table contains entries for the following two network prefixes:

128.10.0.0/16

128.10.2.0/24

Consider what happens if a datagram arrives destined to 128.10.2.3. Surprisingly, the matching procedure described above succeeds for both of the entries. That is, a Boolean *and* of a 16-bit mask will produce 128.10.0.0, and a Boolean *and* with a 24-bit mask will produce 128.10.2.0. Which entry should be used?

To handle ambiguity that arises from overlapping address masks, Internet forwarding uses a *longest prefix match*. That is, instead of examining the entries in arbitrary order, forwarding software arranges to examine entries with the longest prefix first. In the example above, Internet forwarding will choose the entry that corresponds to 128.10.2.0/24. The point is:

> *To resolve ambiguity that can arise when more than one entry matches a destination, Internet forwarding examines entries with the longest prefix first.*

22.9 Destination Address And Next-Hop Address

What is the relationship between the destination address in a datagram header and the address of the next hop to which the datagram is forwarded? The *DESTINATION IP ADDRESS* field in a datagram contains the address of the ultimate destination; it does not change as the datagram passes through the Internet. When a router receives a datagram, the router uses the ultimate destination, *D*, to compute the address of the next router to which the datagram should be sent, *N*. Although the router forwards a datagram to the next hop, *N*, the header in the datagram retains destination address *D*. In other words:

The destination address in a datagram header always refers to the ultimate destination; at each point, a next hop is computed, but the next hop address does not appear in the datagram header.

22.10 Best-Effort Delivery

In addition to defining the format of Internet datagrams, the Internet Protocol defines the semantics of communication, and uses the term *best-effort* to describe the service it offers. In essence, the standard specifies that although IP makes a best-effort to deliver each datagram, IP does not guarantee that it will handle all problems. Specifically, the IP standard acknowledges that the following problems can occur:

- Datagram duplication
- Delayed or out-of-order delivery
- Corruption of data
- Datagram loss

It may seem strange for IP to specify that errors can occur. However, there is an important reason: IP is designed to run over any type of network. We know from earlier chapters that network equipment can experience interference from noise, which can result in corruption or loss. In a system where routes can change, packets following one path may take longer than those following another path, which can result in out-of-order delivery. The point is

Because IP is designed to operate over all types of network hardware, including hardware that experiences problems, IP datagrams may be lost, duplicated, delayed, delivered out of order, or delivered with corrupted data.

Fortunately, we will see that the TCP/IP protocol suite includes additional protocols that handle many of the problems. We will also learn that some applications prefer to use a best-effort service rather than a service that detects and corrects problems.

22.11 IP Encapsulation

How can a datagram be transmitted across a physical network that does not understand the datagram format? The answer lies in a technique known as *encapsulation*. When an IP datagram is encapsulated in a frame, the entire datagram is placed in the payload area of a frame. The network hardware treats a frame that contains a datagram

exactly like any other frame. In fact, the hardware does not examine or change the contents of the payload. Figure 22.4 illustrates the concept.

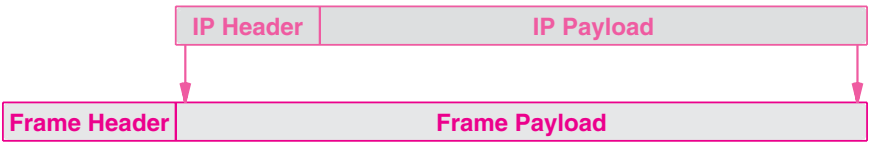


Figure 22.4 Illustration of an IP datagram encapsulated in a frame.

How does a receiver know whether the payload of an incoming frame contains an IP datagram or other data? The sender and receiver must agree on the value used in the frame type field. When it places a datagram in a frame, software on the sending computer assigns the frame type field the special value that is reserved for *IP*. When a frame arrives with the IP value in its type field, the receiver knows that the payload area contains an IP datagram. For example, the Ethernet standard specifies that the type field of an Ethernet frame carrying an IP datagram is assigned *0x0800*.

A frame that carries an IP datagram must have a destination address. Therefore, in addition to placing a datagram in the payload area of a frame, encapsulation requires the sender to supply the MAC address of the next computer to which the datagram should be sent. To compute the appropriate address, IP on the sending computer must bind the next-hop IP address to an equivalent MAC address, which is the destination in the frame header. The next chapter describes the ARP protocol used to perform the binding.

To summarize:

> *A datagram is encapsulated in a frame for transmission across a physical network. The destination address in the frame is the MAC address of the next hop to which the datagram is being sent; the address is obtained by translating the IP address of the next hop to an equivalent MAC address.*

22.12 Transmission Across An Internet

Encapsulation applies to one transmission at a time. After the sender selects a next hop, the sender encapsulates the datagram in a frame and transmits the result across the physical network. When the frame reaches the next hop, the receiving software removes the IP datagram and discards the frame. If the datagram must be forwarded across another network, a new frame is created. Figure 22.5 illustrates how a datagram

is encapsulated and unencapsulated as it travels from a source host to a destination host through three networks and two routers. Each network can use a different hardware technology than the others, meaning that the frame formats and frame header sizes can differ.

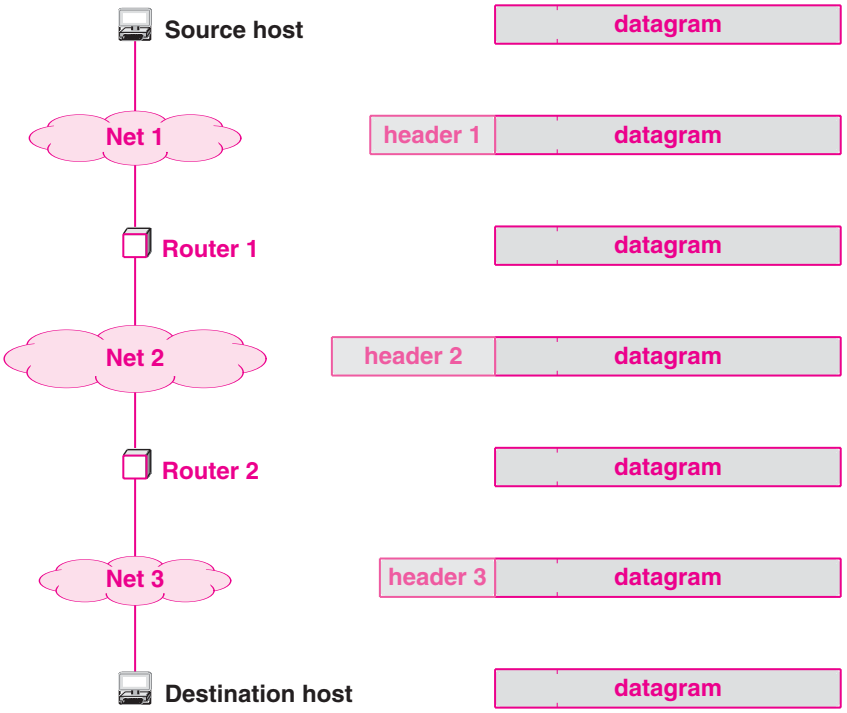


Figure 22.5 An IP datagram as it travels across the Internet.

As the figure shows, hosts and routers store a datagram in memory with no additional header. When the datagram passes across a physical network, the datagram is encapsulated in a frame suitable for the network. The size of the frame header that appears before the datagram depends on the network technology. For example, if *Net 1* represents an Ethernet, the header in frame *1* is an Ethernet header. Similarly, if *Net 2* represents a Wi-Fi network, the header in frame *2* corresponds to a Wi-Fi header.

It is important to observe that frame headers do not accumulate during a trip through the Internet. When a datagram arrives, the datagram is removed from the incoming frame before being encapsulated in an outgoing frame. Thus, when the datagram reaches its final destination, the only frame header on the datagram is the header of the last network over which the datagram arrived. Once the header is removed, the result is the original datagram. The point is:

When a datagram arrives in a network frame, the receiver extracts the datagram from the frame payload area and discards the frame header.

22.13 MTU And Datagram Fragmentation

Each hardware technology specifies the maximum amount of data that a frame can carry. The limit is known as a *Maximum Transmission Unit* (*MTU*). There is no exception to the MTU limit — network hardware is not designed to accept or transfer frames that carry more data than the MTU allows. Thus, a datagram must be smaller or equal to the network MTU, or it cannot be encapsulated for transmission.

In an internet that contains heterogeneous networks, MTU restrictions create a problem. In particular, because a router can connect networks with different MTU values, a datagram that a router receives over one network can be too large to send over another network. For example, Figure 22.6 illustrates a router that interconnects two networks with MTU values of *1500* and *1000*.

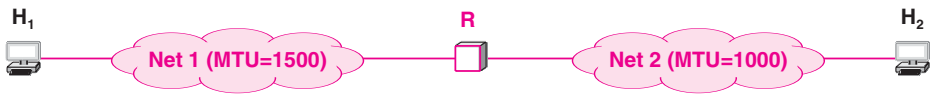


Figure 22.6 Illustration of a router that connects two networks with different MTUs.

In the figure, host H_1 attaches to a network with an MTU of *1500*, and can send a datagram that is up to *1500* octets. Host H_2 attaches to a network that has an MTU of *1000*, which means that it cannot send or receive a datagram larger than *1000* octets. If host H_1 sends a *1500*-octet datagram to host H_2, router *R* will not be able to encapsulate the datagram for transmission across network *2*.

To solve the problem of heterogeneous MTUs, a router uses a technique known as *fragmentation*. When a datagram is larger than the MTU of the network over which it must be sent, the router divides the datagram into smaller pieces called *fragments*, and sends each fragment independently.

Surprisingly, a fragment has the same format as other datagrams — a bit in the *FLAGS* field of the header indicates whether a datagram is a fragment or a complete datagram†. Other fields in the header are assigned information that the ultimate destination uses to *reassemble* fragments to reproduce the original datagram. In particular, the *FRAGMENT OFFSET* field in the header of a fragment specifies where in the original datagram the fragment belongs.

†The datagram header format can be found in Figure 22.2 on page 366.

To fragment a datagram for transmission across a network, a router uses the network MTU and the header size to calculate the maximum amount of data that can be sent in each fragment and the number of fragments that will be needed. The router then creates the fragments. It uses fields from the original header to create a fragment header. For example, the router copies the *IP SOURCE* and *IP DESTINATION* fields from the datagram into the fragment header. Finally, the router copies the appropriate data from the original datagram into the fragment, and transmits the result. Figure 22.7 illustrates the division.

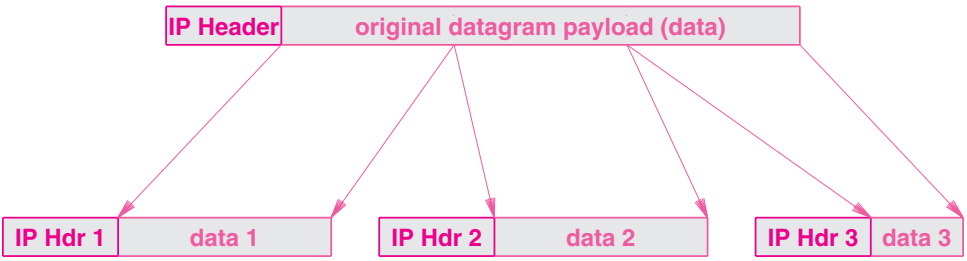


Figure 22.7 An IP datagram divided into three fragments, with the final fragment smaller than the others.

To summarize:

> *Each network has an MTU that specifies the maximum amount of data a frame can carry. When a router receives a datagram that is larger than the MTU of the network over which it is to be sent, the router divides the datagram into smaller pieces called* fragments. *Each fragment uses the IP datagram format, but carries only part of the original payload.*

22.14 Reassembly Of A Datagram From Fragments

The process of recreating a copy of the original datagram from fragments is called *reassembly*. Because each fragment begins with a copy of the original datagram header†, all fragments have the same destination address as the original datagram from which they were derived. The fragment that carries the final piece of data has an additional bit set in the header. Thus, a host performing reassembly can tell whether all fragments have arrived successfully.

Interestingly, IP specifies that the ultimate destination should reassemble fragments. For example, consider the configuration in Figure 22.8.

†The only header fields that differ between the original datagram and a fragment are fields that specify fragmentation.

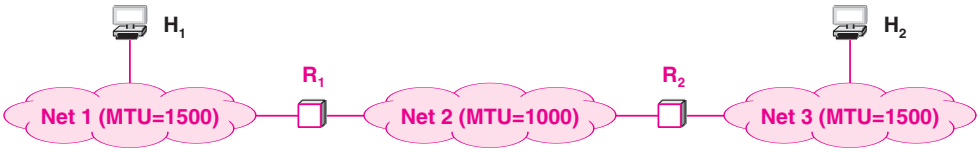


Figure 22.8 Illustration of three networks connected by two routers.

In the figure, if host H_1 sends a *1500*-octet datagram to host H_2, router R_1 will divide the datagram into two fragments, which it will forward to R_2. Router R_2 does not reassemble the fragments. Instead R_2 uses the destination address in a fragment to forward the fragment as usual. The ultimate destination host, H_2, collects the fragments, and reassembles them to produce the original datagram.

Requiring the ultimate destination to reassemble fragments has two advantages.

- First, it reduces the amount of state information in routers. When forwarding a datagram, a router does not need to know whether the datagram is a fragment.
- Second, it allows routes to change dynamically. If an intermediate router were to reassemble fragments, all fragments would need to reach the router.

By postponing reassembly until the ultimate destination, IP is free to pass some fragments from a datagram along differents routes than other fragments. That is, the Internet can change routes at any time (e.g., to route around a hardware failure).

22.15 Collecting The Fragments Of A Datagram

Recall that IP does not guarantee delivery. Thus, individual fragments, which are forwarded exactly like other datagrams, can be lost or arrive out of order. More important, if a given source sends multiple datagrams to the same destination, fragments from multiple datagrams can arrive in arbitrary order.

How does IP software reassemble fragments that arrive out of order? A sender places a unique identification number in the *IDENTIFICATION* field of each outgoing datagram. When a router fragments a datagram, the router copies the identification number into each fragment. A receiver uses the identification number and IP source address in an incoming fragment to determine the datagram to which the fragment belongs. In addition, the *FRAGMENT OFFSET* field tells a receiver where data in the fragment belongs in the original datagram.

22.16 The Consequence Of Fragment Loss

We said that IP does not guarantee fragment delivery — if an underlying network drops packets, either an encapsulated datagram or fragment can be lost. A datagram cannot be reassembled until all fragments arrive. Thus, a problem arises when one or more fragments from a datagram arrive, and other fragments are delayed or lost. Although the datagram cannot be reassembled, the receiver must save the fragments that have arrived in case missing fragments are only delayed.

A receiver cannot hold fragments an arbitrarily long time because fragments occupy space in memory. To avoid exhausting memory, IP specifies a maximum time to hold fragments. When the first fragment arrives from a given datagram, the receiver starts a *reassembly timer*. If all fragments of a datagram arrive before the timer expires, the receiver cancels the timer and reassembles the datagram. However, if the timer expires before all fragments arrive, the receiver discards the fragments that have arrived.

The result of IP's reassembly timer is all-or-nothing: either all fragments arrive and IP reassembles the datagram, or IP discards the incomplete datagram. In particular, there is no mechanism for a receiver to tell the sender which fragments have arrived. The design makes sense because the sender does not know about fragmentation. Furthermore, if the sender did retransmit the datagram, routes may be different, which means a retransmission would not necessarily traverse the same routers. Hence, there is no guarantee that a retransmitted datagram would be fragmented in the same way as the original.

22.17 Fragmenting A Fragment

After performing fragmentation, a router forwards each fragment on to its destination. What happens if a fragment eventually reaches a network that has a smaller MTU? The fragmentation scheme has been planned carefully to make it possible to fragment a fragment. A router along the path divides the fragment into smaller fragments. If networks are arranged in a sequence of decreasing MTUs, each router along the path must further fragment each fragment. Of course, designers work carefully to insure that such situations do not occur in the Internet.

In any case, IP does not distinguish between original fragments and subfragments. In particular, a receiver cannot know whether an incoming fragment is the result of one router fragmenting a datagram or multiple routers fragmenting fragments. The advantage of making all fragments the same is that a receiver can perform reassembly of the original datagram without first reassembling subfragments. Doing so saves CPU time, and reduces the amount of information needed in the header of each fragment.

24

The Future IP (IPv6)

24.1 Introduction

Previous chapters discuss the current version of the Internet Protocol, IPv4. The chapters describe an IP datagram as a header followed by data. The header contains information such as a destination address that IP software uses to deliver the datagram; each header field has a fixed size to make processing efficient. Chapter 22 describes how an IP datagram is encapsulated in a network frame as it travels across a physical network.

This chapter concentrates on the future of the Internet Protocol. It begins by assessing the strengths and limitations of the current version of IP, and then considers a new version of IP that the IETF has developed. The chapter explains features of the new version, and shows how they overcome some of the limitations of the current version.

24.2 The Success Of IP

The current version of IP has been extremely successful. IP has made it possible for the Internet to handle heterogeneous networks, dramatic changes in hardware technology, and extreme increases in scale. Internet protocols provide a set of abstractions that allow applications to communicate without knowledge of the Internet architecture or underlying hardware. To accommodate heterogeneous hardware, IP defines a network-independent addressing scheme, datagram format, encapsulations, and a fragmentation strategy.

The versatility and scalability of IP are evident from the applications that use IP and from the size of the global Internet. More important, IP has accommodated dramatic changes in hardware. Although it was defined before local area network technologies became popular, the original protocol design has continued to work well through several generations of technologies. In addition to higher data rates, IP has accommodated increases in the size of frames.

To summarize:

> *The success of the current version of IP is incredible — the protocol has accommodated changes in hardware technologies, heterogeneous networks, and extremely large scale.*

24.3 The Motivation For Change

If IP works so well, why change? When IP was defined, only a few computer networks existed. The designers decided to use 32 bits for an IP address because doing so allowed the Internet to include over a million networks. However, the global Internet is growing exponentially, with the size doubling in less than a year. At the current growth rate, each of the possible network prefixes will eventually be assigned, and no further growth will be possible. Thus, the primary motivation for defining a new version of IP arose from the address space limitation — larger addresses are necessary to accommodate continued growth of the Internet.

Secondary motivations for changes in IP have arisen from the perception that special facilities are needed for some applications. Consequently, various groups argue that when IP is replaced, the new version should have more features. For example, consider applications that send real-time audio and video. Such applications deliver data at regular intervals, and need a guarantee of low jitter. Unfortunately, a change in routes usually changes end-to-end latency, which means an increase in jitter. Although the current IP datagram header includes a field that can be used to request a type of service, the protocol does not define a real-time service. Thus, is has been argued that a new version of IP should provide a mechanism that allows datagrams carrying real-time traffic to avoid route changes.

Another group argues that a new version of IP should accommodate more complex addressing and routing capabilities. In particular, it should be possible to configure IP addressing and routing to handle replicated services. For example, Google maintains many data centers around the world. When a user enter *google.com* into a browser, the groups argue, it would be beneficial if IP passed datagrams to the nearest Google data center. In addition, many current applications allow a set of users to *collaborate*. To make collaboration efficient, the Internet needs a mechanism that allows groups to be created or changed and a way to send a copy of a packet to each participant in a given group.

24.4 The Hourglass Model And Difficulty Of Change

Although the apparent scarcity of remaining addresses was considered crucial when work began on a new version of IP in 1993, no emergency occurred, and IP has not been changed. To understand why, think of the importance of IP and the cost to change. In terms of importance, IP lies at the center of Internet communication — all applications use IP, and IP runs over all underlying network technologies. Networking professionals say that Internet communication follows an *hourglass model*, and that IP lies at the position where the hourglass is thin. Figure 24.1 illustrates the concept.

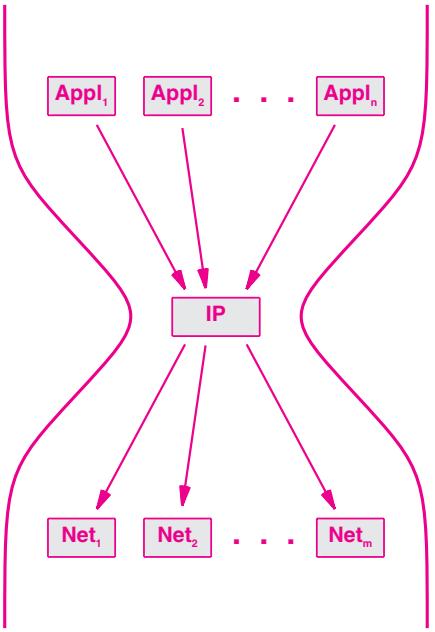


Figure 24.1 The hourglass model of Internet communication with IP at the center.

An important point arises from dependency on IP and the consequent inertia IP introduces:

> *Because IP is central to all Internet communication, changing IP requires a change to the entire Internet.*

24.5 A Name And A Version Number

When researchers began working on a new version of IP, they needed a name for the project. Borrowing from a popular television show, they selected *IP — The Next Generation*, and early reports referred to the new protocol as *IPng*. Unfortunately, many competing proposals were made for IPng, and the name became ambiguous.

When a specific protocol was defined, the designers needed to distinguish the protocol from all other proposals. They decided to use an official version number in the header of the final standardized protocol. The version number that was selected was a surprise. Because the current IP version number is *4*, the networking community expected the next official version to be *5*. However, version *5* had been assigned to an experimental protocol known as *ST*. Consequently, the new version of IP received *6* as its official version number, and the protocol became known as *IPv6*. To distinguish it from IPv6, the current version of IP became known as *IPv4*.

24.6 IPv6 Features

IPv6 retains many of the design features that have made IPv4 so successful. Like IPv4, IPv6 is connectionless — each datagram contains a destination address, and each datagram is routed independently. Like IPv4, the header in a datagram contains a maximum number of hops the datagram can take before being discarded. More important, IPv6 retains most of the general facilities provided by the IPv4 options.

Despite retaining the basic concepts from the current version, IPv6 changes all the details. For example, IPv6 uses larger addresses and an entirely new datagram header format. Furthermore, IPv6 divides header information into a series of fixed-length headers. Thus, unlike IPv4, which places key information in fixed fields of the header and only appends variable-length options for less important information, the IPv6 header is always a variable size.

The new features in IPv6 can be grouped into five broad categories:

- *Address Size*. Instead of 32 bits, each IPv6 address contains 128 bits. The resulting address space is large enough to accommodate continued growth of the world-wide Internet for many decades.
- *Header Format*. The IPv6 datagram header is completely different than the IPv4 header. Almost every field in the header has been changed; some have been replaced.
- *Extension Headers*. Unlike IPv4, which uses a single header format for all datagrams, IPv6 encodes information into separate headers. A datagram consists of the base IPv6 header followed by zero or more extension headers, followed by data.

- *Support For Real-Time Traffic*. IPv6 includes a mechanism that allows a sender and receiver to establish a high-quality path through the underlying network and to associate datagrams with that path. Although the mechanism is intended for use with audio and video applications that require high performance guarantees, the mechanism can also be used to associate datagrams with low-cost paths.

- *Extensible Protocol*. Unlike IPv4, IPv6 does not specify all possible protocol features. Instead, the designers have provided a scheme that allows a sender to add additional information to a datagram. The extension scheme makes IPv6 more flexible than IPv4, and means that new features can be added to the design as needed.

The next sections explain how the new features are implemented by showing the organization of an IPv6 datagram and the addressing structure.

24.7 IPv6 Datagram Format

An IPv6 datagram contains a series of headers. As Figure 24.2 illustrates, each datagram begins with a *base header*, which is followed by zero or more *extension headers* followed by the payload.

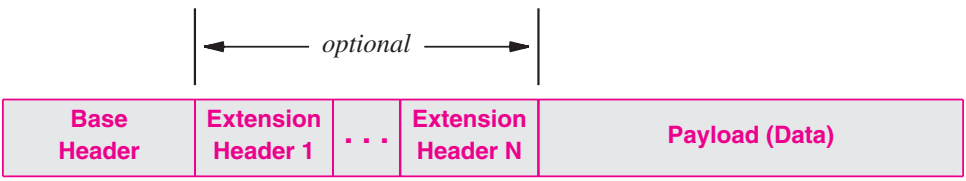


Figure 24.2 The general form of an IPv6 datagram.

Although the figure illustrates the general structure, fields are not drawn to scale. In particular, some extension headers are larger than the base header, and others are smaller. In many datagrams, the size of the payload is much larger than the size of the headers.

24.8 IPv6 Base Header Format

Although it is twice as large as an IPv4 header, the IPv6 base header contains less information. Figure 24.3 illustrates the format.

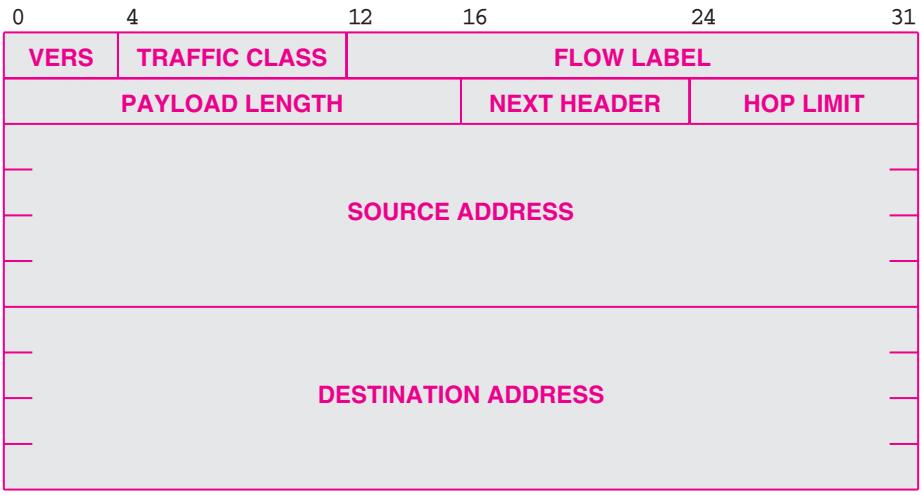


Figure 24.3 The format of the base header in an IPv6 datagram.

As the figure shows, most of the space in the header is devoted to the *SOURCE ADDRESS* and *DESTINATION ADDRESS* fields, each of which occupies sixteen octets, four times more than an IPv4 address. As in IPv4, a source address identifies the original source, and the destination address identifies the ultimate recipient.

In addition to the source and destination addresses, the base header contains six fields. The *VERS* field identifies the protocol as version *6*. The *TRAFFIC CLASS* field specifies the *traffic class* using a definition of traffic types known as *differentiated services* to specify general characteristics that the datagram needs. For example, to send interactive traffic (e.g., keystrokes and mouse movements), one might specify a class that has low latency. To send real-time audio across the Internet, however, a sender might request a path with low jitter. The *PAYLOAD LENGTH* field corresponds to IPv4's datagram length field. Unlike IPv4, the *PAYLOAD LENGTH* specifies only the size of the data being carried (i.e., the payload); the size of the header is excluded. The *HOP LIMIT* corresponds to the IPv4 *TIME-TO-LIVE* field. IPv6 interprets the *HOP LIMIT* strictly — the datagram will be discarded if the *HOP LIMIT* counts down to zero before the datagram arrives at its destination. Field *FLOW LABEL* was originally intended to associate a datagram with a particular underlying network path. Since IPv6 was defined, the use of end-to-end flow labels has fallen out of favor, and the *FLOW LABEL* has become less important.

The MIME standard inserts extra header lines to allow non-text attachments to be sent within an email message. An attachment is encoded as printable letters, and a separator line appears before each attachment.

4.17 Domain Name System (DNS)

The *Domain Name System* (*DNS*) provides a service that maps human-readable symbolic names to computer addresses. Browsers, mail software, and most other Internet applications use the DNS. The system provides an interesting example of client-server interaction because the mapping is not performed by a single server. Instead, the naming information is distributed among a large set of servers located at sites across the Internet. Whenever an application program needs to translate a name, the application becomes a client of the naming system. The client sends a request message to a name server, which finds the corresponding address and sends a reply message. If it cannot answer a request, a name server temporarily becomes the client of another name server, until a server is found that can answer the request.

Syntactically, each name consists of a sequence of alpha-numeric segments separated by periods. For example, a computer in the Computer Science Department at Purdue University has the domain name:

mordred.cs.purdue.edu

and a computer at Cisco, Incorporated has the domain name:

anakin.cisco.com

Domain names are hierarchical, with the most significant part of the name on the right. The left-most segment of a name (*mordred* and *anakin* in the examples) is the name of an individual computer. Other segments in a domain name identify the group that owns the name. For example, the segment *purdue* gives the name of a university, and *cisco* gives the name of a company. DNS does not specify the number of segments in a name. Instead, each organization can choose how many segments to use for computers inside the organization and what the segments represent.

The Domain Name System does specify values for the most significant segment, which is called a *top-level domain* (*TLD*). Top-level domains are controlled by the *Internet Corporation for Assigned Names and Numbers* (*ICANN*), which designates one or more *domain registrars* to administer a given top-level domain and approve specific names. Some TLDs are *generic*, which means they are generally available. Other TLDs are restricted to specific groups or government agencies. Figure 4.16 lists example top-level DNS domains.

Domain Name	Assigned To
aero	Air transport industry
arpa	Infrastructure domain
asia	For or about Asia
biz	Businesses
com	Commercial organizations
coop	Cooperative associations
edu	Educational institutions
gov	United States Government
info	Information
int	International treaty organizations
jobs	Human resource managers
mil	United States military
mobi	Mobile content providers
museum	Museums
name	Individuals
net	Major network support centers
org	Non-commercial organizations
pro	Credentialed professionals
travel	Travel and tourism
country code	A sovereign nation

Figure 4.16 Example top-level domains and the group to which each is assigned.

An organization applies for a name under one of the existing top-level domains. For example, most U.S. corporations choose to register under the *com* domain. Thus, a corporation named *Foobar* might request to be assigned domain *foobar* under the top-level domain *com*. Once the request is approved, Foobar Corporation will be assigned the domain:

foobar.com

Once the name has been assigned another organization named Foobar can apply for *foobar.biz* or *foobar.org*, but not *foobar.com*. Furthermore, once *foobar.com* has

been assigned, the Foobar Corporation can choose how many additional levels to add and the meaning of each. Thus, if Foobar has locations on the East and West coast, one might find names such as:

computer1.east-coast.foobar.com

or Foobar may choose a relatively flat naming hierarchy with all computers identified by name and the company's domain name:

computer1.foobar.com

In addition to the familiar organizational structure, the DNS allows organizations to use a geographic registration. For example, the Corporation For National Research Initiatives registered the domain:

cnri.reston.va.us

because the corporation is located in the town of Reston, Virginia in the United States. Thus, names of computers at the corporation end in *.us* instead of *.com*.

Some foreign countries have adopted a combination of geographic and organizational domain names. For example, universities in the United Kingdom register under the domain:

ac.uk

where *ac* is an abbreviation for *academic*, and *uk* is the official country code for the United Kingdom.

4.18 Domain Names That Begin With www

Many organizations assign domain names that reflect the service a computer provides. For example, a computer that runs a server for the File Transfer Protocol might be named:

ftp.foobar.com

Similarly, a computer that runs a web server, might be named:

www.foobar.com

Such names are mnemonic, but are not required. In particular, the use of *www* to name computers that run a web server is merely a convention — an arbitrary computer can run a web server, even if the computer's domain name does not contain *www*. Furthermore, a computer that has a domain name beginning with *www* is not required to run a web server. The point is:

Using the first label in a domain name to denote a service (e.g., www) is merely a convention to help humans.

4.19 The DNS Hierarchy And Server Model

One of the main features of the Domain Name System is autonomy — the system is designed to allow each organization to assign names to computers or to change those names without informing a central authority. To achieve autonomy, each organization is permitted to operate DNS servers for its part of the hierarchy. Thus, Purdue University operates a server for names ending in *purdue.edu*, and IBM Corporation operates a server for names ending in *ibm.com*. Each DNS server contains information that links the server to other domain name servers up and down the hierarchy. Furthermore, a given server can be *replicated*, such that multiple physical copies of the server exist. Replication is especially useful for heavily used servers, such as the *root servers* that provide information about top-level domains. In such cases, administrators must guarantee that all copies are coordinated so they provide exactly the same information.

Each organization is free to choose the details of its servers. For example, a small organization that only has a few computers can contract with an ISP to run a DNS server. A large organization that runs its own server can choose to place all names for the organization in a single physical server, or can choose to divide its names among multiple servers. For example, Figure 4.17 illustrates how the hypothetical Foobar Corporation might choose to structure servers if the corporation had a candy division and a soap division.

4.20 Name Resolution

The translation of a domain name into an address is called *name resolution*, and the name is said to be *resolved* to an address. Software to perform the translation is known as a *name resolver* (or simply *resolver*). In the socket API, for example, the resolver is invoked by calling function *gethostbyname*. The resolver becomes a client, contacts a DNS server, and returns an answer to the caller.

Each resolver is configured with the address of one or more *local* domain name servers†. The resolver forms a *DNS request* message, sends the message to the local server, and waits for the server to send a *DNS reply* message that contains the answer. A resolver can choose to use either the stream or message paradigm when communicating with a DNS server; most resolvers are configured to use a message paradigm because it imposes less overhead for a small request.

As an example of name resolution, consider the server hierarchy that Figure 4.17a illustrates, and assume a computer in the soap division generates a request for name *chocolate.candy.foobar.com*. The resolver will be configured to send the request to the local DNS server (i.e., the server for foobar.com). Although it cannot answer the request, the server knows to contact the server for *candy.foobar.com*, which can generate an answer.

†The significance of contacting a local server first will become apparent when we discuss caching.

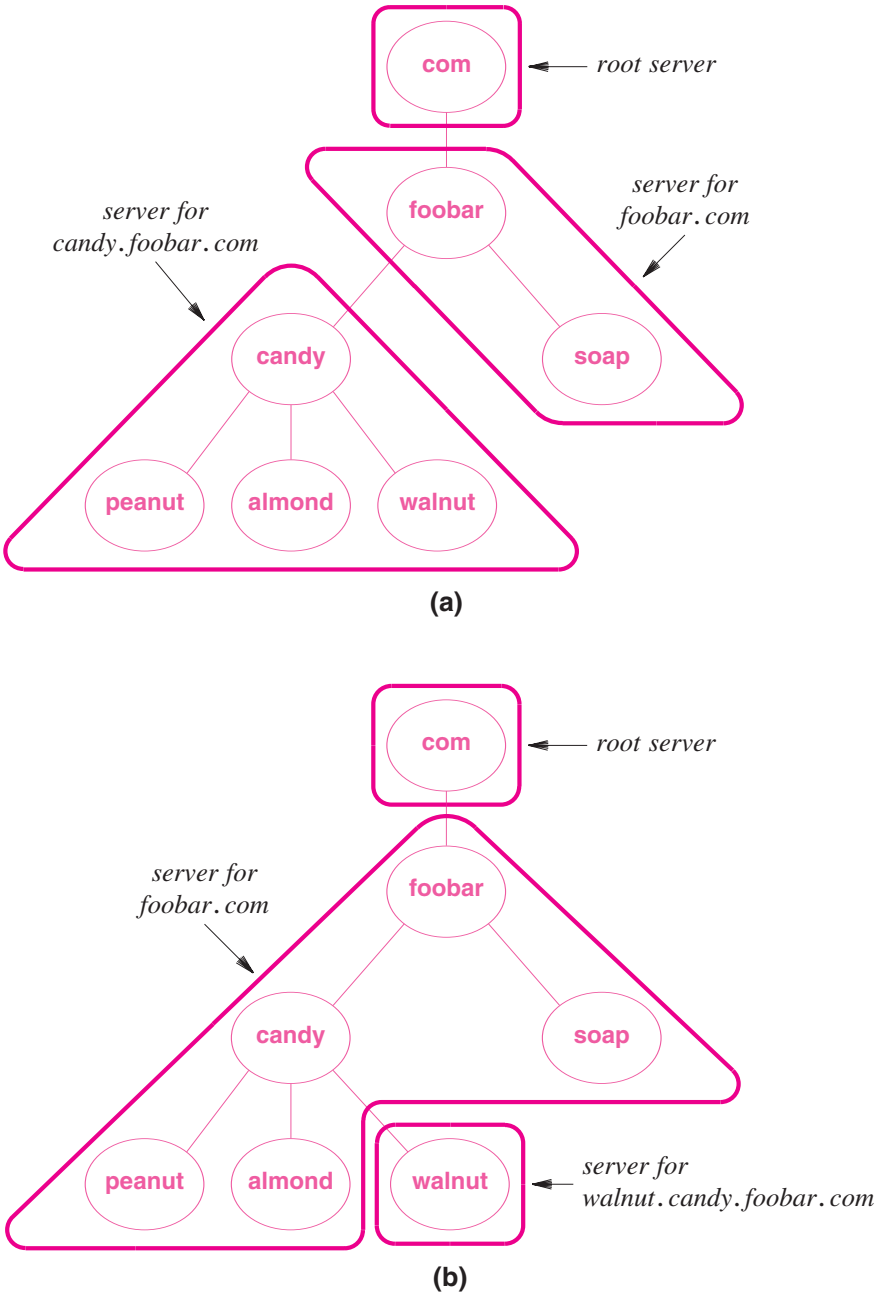


Figure 4.17 A hypothetical DNS hierarchy and two possible assignments of names to servers.

4.21 Caching In DNS Servers

The *locality of reference* principle that forms the basis for caching applies to the Domain Name System in two ways:

- Spatial: A user tends to look up the names of local computers more often than the names of remote computers
- Temporal: A user tends to look up the same set of domain names repeatedly

We have already seen how DNS exploits spatial locality: a name resolver contacts a local server first. To exploit temporal locality, a DNS server caches all lookups. Algorithm 4.4 summarizes the process.

Algorithm 4.4

```
Given:
   A request message from a DNS name resolver
Provide:
   A response message that contains the address
Method:
   Extract the name, N, from the request
   if ( server is an authority for N ) {
        Form and send a response to the requester;
   else if ( answer for N is in the cache ) {
        Form and send a response to the requester;
   else {   /* Need to look up an answer */
        if ( authority server for N is known ) {
             Send request to authority server;
        } else {
             Send request to root server;
        }
        Receive response and place in cache;
        Form and send a response to the requester;
   }
```

Algorithm 4.4 Steps a DNS server takes to resolve a name.

According to the algorithm, when a request arrives for a name outside the set for which the server is an authority, further client-server interaction results. The server temporarily becomes a client of another name server. When the other server returns an answer, the original server caches the answer and sends a copy of the answer back to the resolver from which the request arrived. Thus, in addition to knowing the address of all servers down the hierarchy, each DNS server must know the address of a root server.

The fundamental question in all caching relates to the length of time items should be cached — if an item is cached too long, the item will become *stale*. DNS solves the problem by arranging for an authoritative server to specify a cache timeout for each item. Thus, when a local server looks up a name, the response consists of a *Resource Record* that specifies a cache timeout as well as an answer. Whenever a server caches an answer, the server honors the timeout specified in the Resource Record. The point is:

Because each DNS Resource Record generated by an authoritative server specifies a cache timeout, items can be removed from a DNS cache when they become stale.

DNS caching does not stop with servers: a resolver can cache items as well. In fact, the resolver software in most computer systems caches the answers from DNS lookups, which means that successive requests for the same name do not need to use the network because the resolver can satisfy the request from the cache on the local disk.

4.22 Types Of DNS Entries

Each entry in a DNS database consists of three items: a domain name, a record type, and a value. The record type specifies how the value is to be interpreted (e.g., that the value is an IP address). More important, a query sent to a DNS server specifies both a domain name and a type; the server only returns a binding that matches the type of the query.

The principal type maps a domain name to an IP address. DNS classifies such bindings as type *A*, and type *A* lookup is used by applications such as *FTP*, *ping*, or a browser. DNS supports several other types, including type *MX* that specifies a *Mail eXchanger*. When it looks up the name in an email address, SMTP uses type *MX*. The answer that the server returns matches the requested type. Thus, an email system will receive an answer that matches type *MX*. The important point is:

Each entry in a DNS server has a type. When a resolver looks up a name, the resolver specifies the type that is desired, and the DNS server returns only entries that match the specified type.