PART 2

Network Layer

CHAPTER

4

Introduction to Network Layer

To solve the problem of delivery through several links, the network layer (or the internetwork layer, as it is sometimes called) was designed. The network layer is responsible for host-to-host delivery and for routing the packets through the routers. In this chapter, we give an introduction to the network layer to prepare readers for a more thorough coverage in Chapters 5 through 12. In this chapter we give the rationale for the need of the network layers and the issues involved. However, we need the next eight chapters to fully understand how these issues are answered. We may even need to cover the whole book before we get satisfactory answers for them.

OBJECTIVES

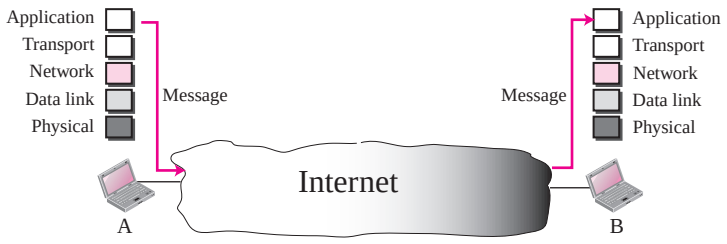
This chapter has several objectives:

- To introduce switching and in particular packet switching as the mechanism of data delivery in the network layer.
- To discuss two distinct types of services a packet-switch network can provide: connectionless service and connection-oriented service.
- To discuss how routers forward packets in a connectionless packet-switch network using the destination address of the packet and a routing table.
- To discuss how routers forward packets in a connection-oriented packet-switch network using the label on the packet and a routing table.
- To discuss services already provided in the network layer such as logical addressing and delivery at the source, at each router, and at the destination.
- To discuss issues or services that are not directly provided in the network layer protocol, but are sometimes provided by some auxiliary protocols or some protocols added later to the Internet.

4.1 INTRODUCTION

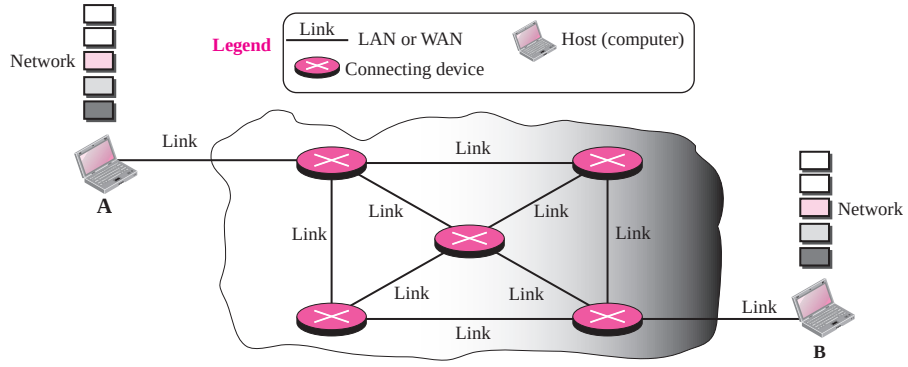
At the conceptual level, we can think of the global Internet as a black box network that connects millions (if not billions) of computers in the world together. At this level, we are only concerned that a message from the application layer in one computer reaches the application layer in another computer. In this conceptual level, we can think of communication between A and B as shown in Figure 4.1.

Figure 4.1 *Internet as a black box*



The Internet, however, is not one single network; it is made of many networks (or links) connected together through the connecting devices. In other words, the Internet is an internetwork, a combination of LANs and WANs. To better understand the role of the network layer (or the internetwork layer), we need to move from our conceptual level and think about all of these LANs and WANs that make the Internet. Since it is impossible to show all of these LANs and WANs, we show only an imaginary small internet with a few networks and a few connecting devices, as shown in Figure 4.2.

Figure 4.2 *Internet as a combination of LAN and WANs connected together*



An internet as a combination of links and connecting devices

In this model, a connecting device such as a router acts as a switch. When a packet arrives from one of its ports (interface), the packet is forwarded through another port to the next switch (or final destination). In other words, a process called **switching** occurs at the connecting device.

4.2 SWITCHING

From the previous discussion, it is clear that the passage of a message from a source to a destination involves many decisions. When a message reaches a connecting device, a decision needs to be made to select one of the output ports through which the packet needs to be send out. In other words, the connecting device acts as a switch that connects one port to another port.

Circuit Switching

One solution to the switching is referred to as **circuit switching**, in which a physical circuit (or channel) is established between the source and destination of the message before the delivery of the message. After the circuit is established, the entire message, is transformed from the source to the destination. The source can then inform the network that the transmission is complete, which allows the network to open all switches and use the links and connecting devices for another connection. The circuit switching was never implemented at the network layer; it is mostly used at the physical layer.

In circuit switching, the whole message is sent from the source to the destination without being divided into packets.

Example 4.1

A good example of a circuit-switched network is the early telephone systems in which the path was established between a caller and a callee when the telephone number of the callee was dialed by the caller. When the callee responded to the call, the circuit was established. The voice message could now flow between the two parties, in both directions, while all of the connecting devices maintained the circuit. When the caller or callee hung up, the circuit was disconnected. The telephone network is not totally a circuit-switched network today.

Packet Switching

The second solution to switching is called **packet switching**. The network layer in the Internet today is a packet-switched network. In this type of network, a message from the upper layer is divided into manageable packets and each packet is sent through the network. The source of the message sends the packets one by one; the destination of the message receives the packets one by one. The destination waits for all packets belonging to the same message to arrive before delivering the message to the upper layer. The connecting devices in a packet-switching network still need to decide how to route the packets to the final destination. Today, a packet-switched network can use two different

approaches to route the packets: the datagram approach and the virtual circuit approach. We discuss both approaches in the next section.

In packet switching, the message is first divided into manageable packets at the source before being transmitted. The packets are assembled at the destination.

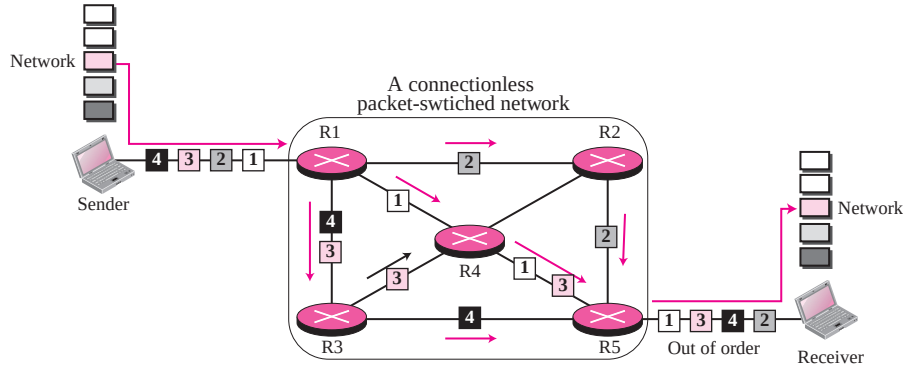
4.3 PACKET SWITCHING AT NETWORK LAYER

The network layer is designed as a packet-switched network. This means that the packet at the source is divided into manageable packets, normally called **datagrams**. Individual datagrams are then transferred from the source to the destination. The received datagrams are assembled at the destination before recreating the original message. The packet-switched network layer of the Internet was originally designed as a *connectionless service*, but recently there is a tendency to change this *to a connection-oriented service*. We first discuss the dominant trend and then briefly discuss the new one.

Connectionless Service

When the Internet started, the network layer was designed to provide a **connectionless service**, in which the network layer protocol treats each packet independently, with each packet having no relationship to any other packet. The packets in a message may or may not travel the same path to their destination. When the Internet started, it was decided to make the network layer a connectionless service to make it simple. The idea was that the network layer is only responsible for delivery of packets from the source to the destination. Figure 4.3 shows the idea.

Figure 4.3 *A connectionless packet-switched network*

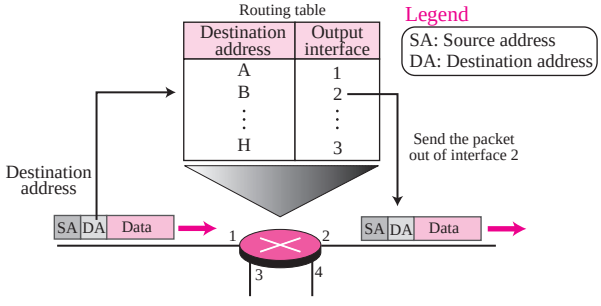


When the network layer provides a connectionless service, each packet traveling in the Internet is an independent entity; there is no relationship between packets belonging to the same message. The switches in this type of network are called *routers*. A packet

belonging to a message may be followed by a packet belonging to the same message or a different message. A packet may be followed by a packet coming from the same or from a different source.

Each packet is routed based on the information contained in its header: source and destination address. The destination address defines where it should go; the source address defines where it comes from. The router in this case routes the packet based only on the destination address. The source address may be used to send an error message to the source if the packet is discarded. Figure 4.4 shows the forwarding process in a router in this case. We have used symbolic addresses such as A and B.

Figure 4.4 *Forwarding process in a router when used in a connectionless network*

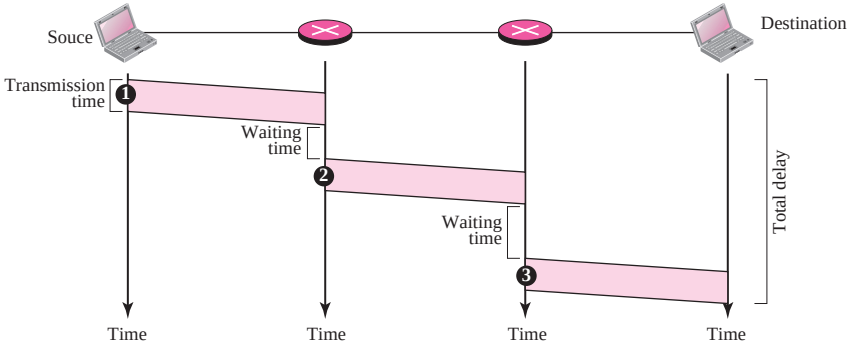


In a connectionless packet-switched network, the forwarding decision is based on the destination address of the packet.

Delay In Connectionless Network

If we ignore the fact that the packet may be lost and resent and also the fact that the destination may be needed to wait to receive all packets, we can model the delay as shown in Figure 4.5.

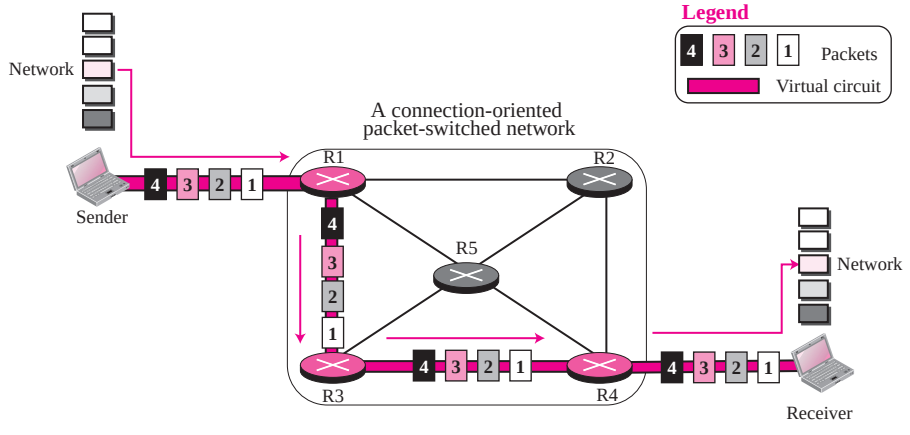
Figure 4.5 *Delay in a connectionless network*



Connection-Oriented Service

In a **connection-oriented service,** there is a relation between all packets belonging to a message. Before all datagrams in a message can be sent, a virtual connection should be set up to define the path for the datagrams. After connection setup, the datagrams can follow the same path. In this type of service, not only must the packet contain the source and destination addresses, it must also contain a *flow label*, a *virtual circuit identifier* that defines the virtual path the packet should follow. We will shortly show how this flow label is determined, but for the moment, we assume that the packet carries this *label*. Although it looks as though the use of the label may make the source and destination addresses useless, the parts of the Internet that use connectionless service at the network layer still keep these addresses for several reasons. One reason is that part of the packet path may still be using the connectionless service. Another reason is that the protocol at the network layer is designed with these addresses and it may take a while before they can be changed. Figure 4.6 shows the concept of connection-oriented service.

Figure 4.6 *A connection-oriented packet switched network*



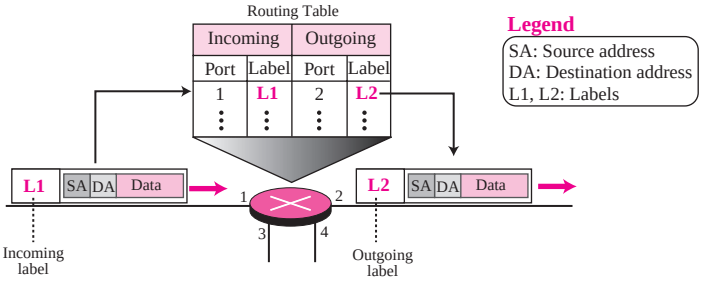
Each packet is forwarded based on the label in the packet. To follow the idea of connection-oriented design to be used in the Internet, we assume that the packet has a label when it reaches the router. Figure 4.7 shows the idea.

In this case, the forwarding decision is based on the value of the label, or virtual circuit identifier as it is sometimes called.

To create a connection-oriented service, a three-phase process is used: *setup*, *data transfer*, and *teardown*. In the setup phase, the source and destination addresses of the sender and receiver is used to make table entries for the connection-oriented service. In the teardown phase, the source and destination inform the router to delete the corresponding entries. Data transfer occurs between these two phases.

In a connection-oriented packet switched network, the forwarding decision is based on the label of the packet.

Figure 4.7 *Forwarding process in a router when used in a connection-oriented network*



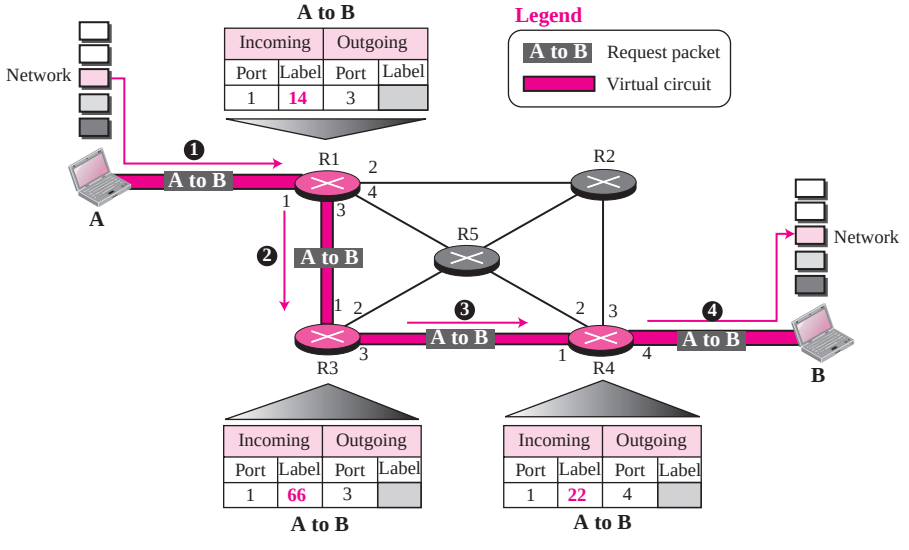
Setup Phase

In the **setup phase**, a router creates an entry for a virtual circuit. For example, suppose source A needs to create a virtual circuit to destination B. Two auxiliary packets need to be exchanged between the sender and the receiver: the request packet and the acknowledgment packet.

Request packet A request packet is sent from the source to the destination. This auxiliary packet carries the source and destination addresses. Figure 4.8 shows the process.

1. Source A sends a request packet to router R1.
2. Router R1 receives the request packet. It knows that a packet going from A to B goes out through port 3. How the router has obtained this information is a point

Figure 4.8 *Sending request packet in a virtual-circuit network*

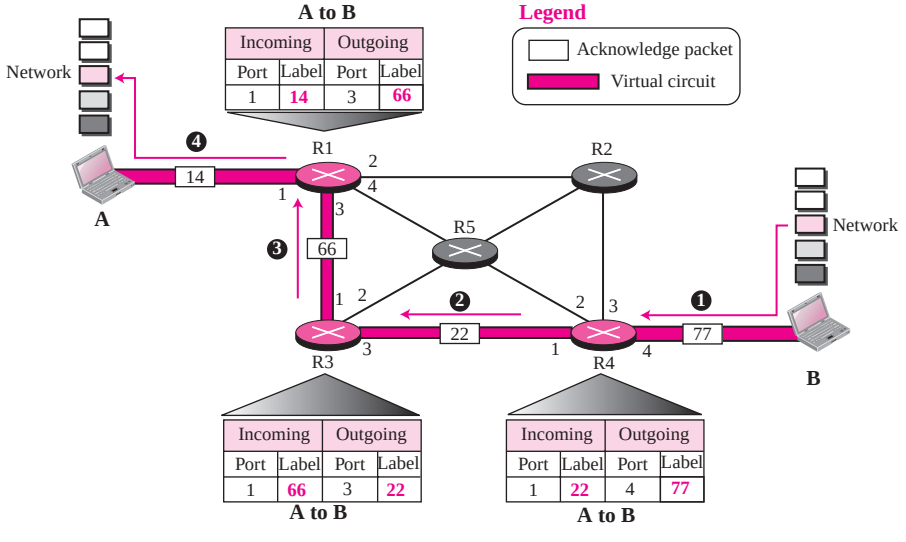


covered in future chapters. For the moment, assume that it knows the output port. The router creates an entry in its table for this virtual circuit, but it is only able to fill three of the four columns. The router assigns the incoming port (1) and chooses an available incoming label (14) and the outgoing port (3). It does not yet know the outgoing label, which will be found during the acknowledgment step. The router then forwards the packet through port 3 to router R3.

3. Router R3 receives the setup request packet. The same events happen here as at router R1; three columns of the table are completed: in this case, incoming port (1), incoming label (66), and outgoing port (2).
4. Router R4 receives the setup request packet. Again, three columns are completed: incoming port (2), incoming label (22), and outgoing port (3).
5. Destination B receives the setup packet, and if it is ready to receive packets from A, it assigns a label to the incoming packets that come from A, in this case 77. This label lets the destination know that the packets come from A, and not other sources.

Acknowledgment Packet A special packet, called the acknowledgment packet, completes the entries in the switching tables. Figure 4.9 shows the process.

Figure 4.9 *Setup acknowledgment in a virtual-circuit network*



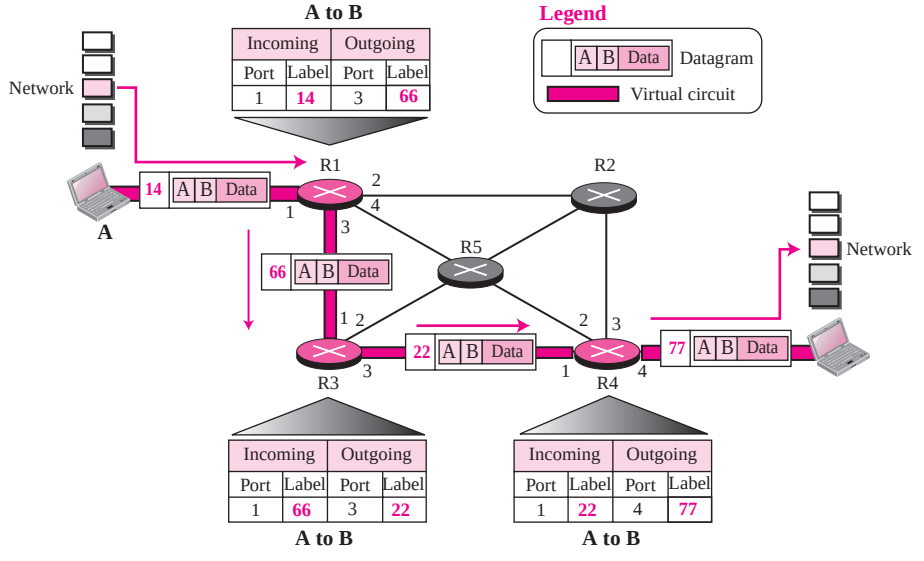
1. The destination sends an acknowledgment to router R4. The acknowledgment carries the global source and destination addresses so the router knows which entry in the table is to be completed. The packet also carries label 77, chosen by the destination as the incoming label for packets from A. Router R4 uses this label to complete the outgoing label column for this entry. Note that 77 is the incoming label for destination B, but the outgoing label for router R4.

2. Router R4 sends an acknowledgment to router R3 that contains its incoming label in the table, chosen in the setup phase. Router R3 uses this as the outgoing label in the table.
3. Router R3 sends an acknowledgment to router R1 that contains its incoming label in the table, chosen in the setup phase. Router R1 uses this as the outgoing label in the table.
4. Finally router R1 sends an acknowledgment to source A that contains its incoming label in the table, chosen in the setup phase.
5. The source uses this as the outgoing label for the data packets to be sent to destination B.

Data Transfer Phase

The second phase is called the **data transfer phase**. After all routers have created their routing table for a specific virtual circuit, then the network-layer packets belonging to one message can be sent one after another. In Figure 4.10, we show the flow of one single packet, but the process is the same for 1, 2, or 100 packets. The source computer uses the label 14, which it has received from router R1 in the setup phase. Router R1 forwards the packet to router R3, but changes the label to 66. Router R3 forwards the packet to router R4, but changes the label to 22. Finally, router R4 delivers the packet to its final destination with the label 77. All the packets in the message follow the same sequence of labels to reach their destination. The packet arrives in order at the destination.

Figure 4.10 *Flow of one packet in an established virtual circuit.*



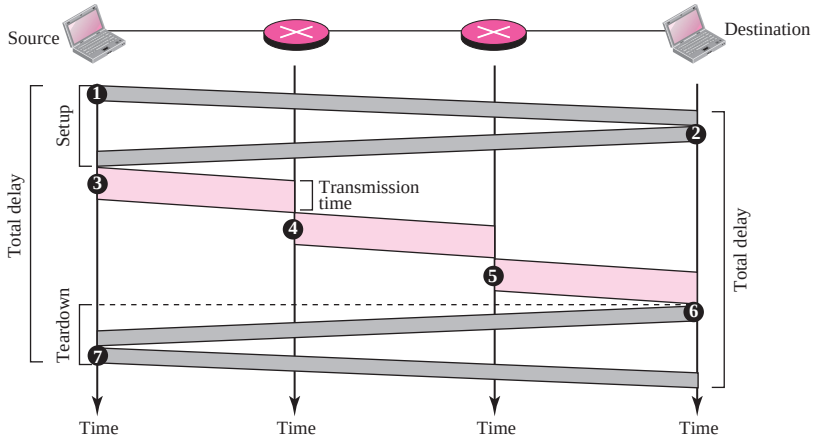
Teardown Phase

In the **teardown phase,** source A, after sending all packets to B, sends a special packet called a *teardown packet*. Destination B responds with a *confirmation* packet. All routers delete the corresponding entry from their tables.

Delay In Connection-Oriented Network

If we ignore the fact that the packet may be lost and resent, we can model the delay as shown in Figure 4.11.

Figure 4.11 *Delay in a connection-oriented network*



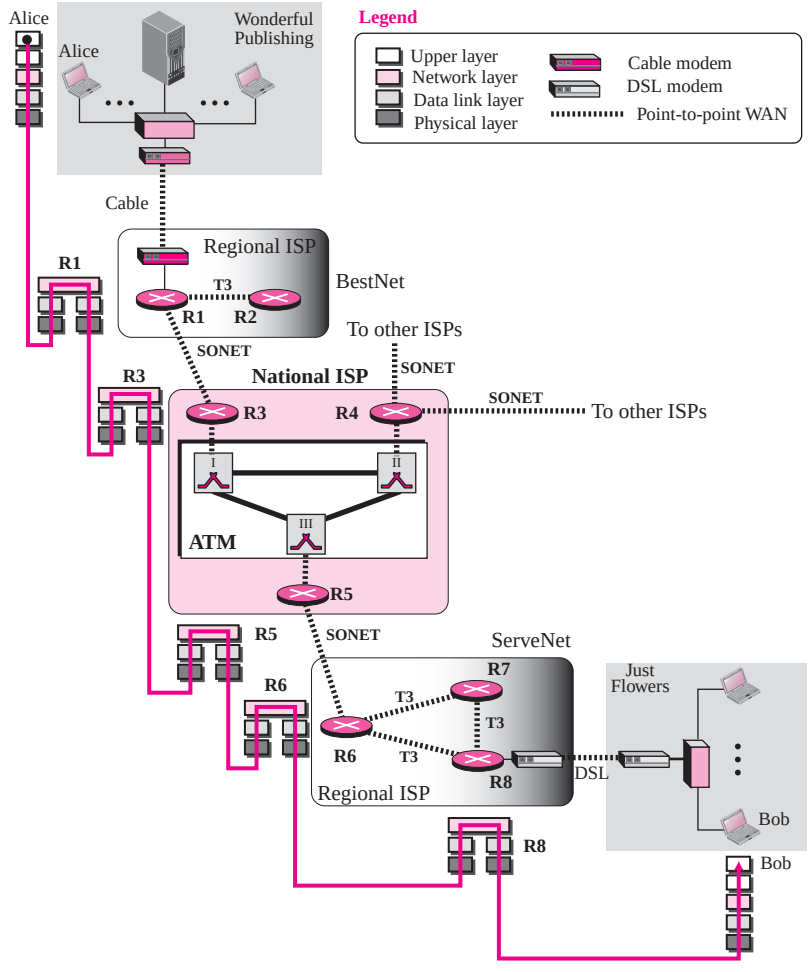
4.4 NETWORK LAYER SERVICES

In this section, we briefly discuss services provided by the network layer. Our discussion is mostly based on the connectionless service, the dominant service in today's Internet.

An Example

To better understand the issues to be discussed, we give an example. In Figure 4.12, we assume Alice, who is working in a publishing company, Wonderful Publishing, needs to send a message to Bob, the manager of a flower shop, Just Flowers, to inform him that the advertising brochure for the shop has been printed and is ready to be shipped. Alice sends the message using an e-mail. Let us follow the imaginary path Alice's message takes to reach Bob. The Wonderful Publishing company uses a LAN, which is connected via a cable WAN to a regional ISP called BestNet; the Just Flowers company also uses a LAN, which is connected via a DSL WAN to another regional ISP called ServeNet. The two regional ISPs are connected through high-speed SONET WANs to a national ISP. The message that Alice sends to Bob may be split into several network-layer packets. As we will see shortly, packets may or may not follow the same path. For the sake of discussion, we follow the path of one single packet from Alice's computer to Bob's computer. We also assume that the packet passes through routers R1, R3, R5, R6, and R8 before reaching its destination. The two computers are involved in five layers; the routers are involved in three layers of the TCP/IP protocol suite.

Figure 4.12 *An imaginary part of the Internet*



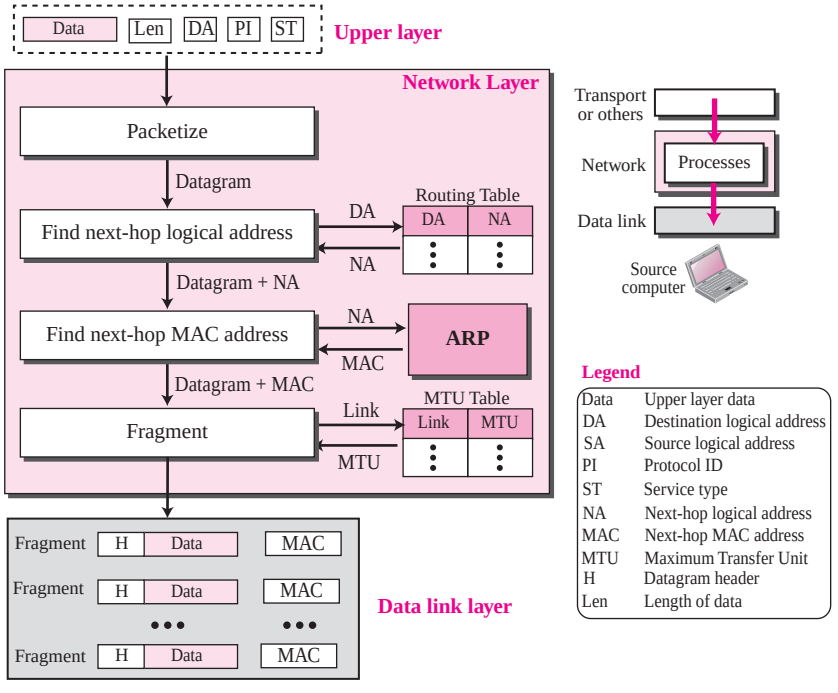
Logical Addressing

Since the network layer provides end-to-end communication, the two computers that need to communicate with each other each need a universal identification system, referred to as network-layer address or logical address. This type of identification is provided in the network layer through a uniform and global addressing mechanism. The Internet uses an address space. Each entity that needs to use the Internet needs to be assigned a unique address from this pool. In Chapter 5 we discuss this addressing space in version 4 of the Internet; In Chapter 26, we discuss the new addressing system in version 6 (version 5 was never implemented). In Figure 4.12, Alice and Bob need two network-layer addresses to be able to communicate.

Services Provided at the Source Computer

The network layer at the source computer provides four services: packetizing, finding the logical address of the next hop, finding the physical (MAC) address of the next hop, and fragmenting the datagram if necessary. Figure 4.13 shows these services.

Figure 4.13 *Services provided at the source computer*



The network layer receives several pieces of information from the upper layer: data, length of data, logical destination address, protocol ID (the identifier of the protocol using the network layer), and service type (discussed later). The network layer processes these pieces of information to create a set of fragmented datagrams and the next-hop MAC address and delivered them to the data link layer. We briefly discuss each service here, but the future chapters in this part of the book explain more.

Packetizing

The first duty of the network layer is to encapsulate the data coming from the upper layer in a datagram. This is done by adding a header to the data that contains the logical source and destination address of the packet, information about fragmentation, the protocol ID of the protocol that has requested the service, the data length, and possibly some options. The network layer also includes a checksum that is calculated only over the datagram header. We discuss the format of the datagram and the checksum calculation in Chapter 7. Note that the upper layer protocol only provides the logical destination

address; the logical source address comes from the network layer itself (any host needs to know its own logical address).

Finding Logical Address of Next Hop

The prepared datagram contains the source and destination addresses of the packet. The datagram, as we saw before, may have to pass through many networks to reach its final destination. If the destination computer is not connected to the same network as the source, the datagram should be delivered to the next router. The source and destination address in the datagram does not tell anything about the logical address of the next hop. The network layer at the source computer needs to consult a routing table to find the logical address of the next hop.

Finding MAC Address of Next Hop

The network layer does not actually deliver the datagram to the next hop; it is the duty of the data link layer to do the delivery. The data link layer needs the MAC address of the next hop to do the delivery. To find the MAC address of the next hop, the network layer could use another table to map the next-hop logical address to the MAC address. However, for the reason we discuss in Chapter 8, this task has been assigned to another auxiliary protocol called Address Resolution Protocol (ARP) that finds the MAC address of the next hop given the logical address.

Fragmentation

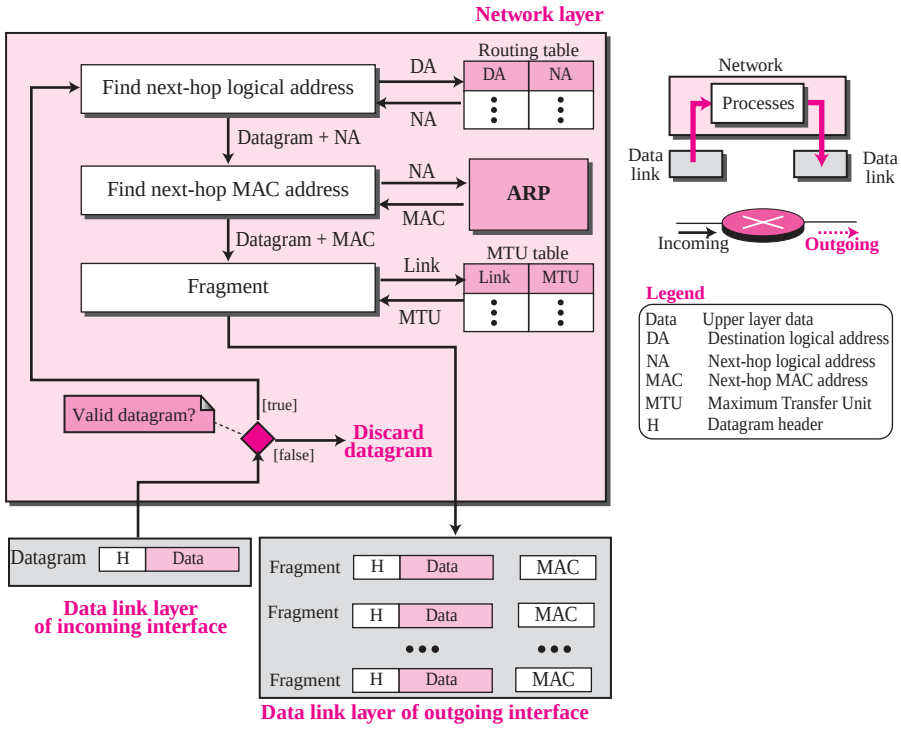
The datagram at this step may not be ready to be passed to the data link layer. As we saw in Chapter 3, most LANs and WANs have a limit on the size of the data to be carried in a frame (MTU). The datagram prepared at the network layer, may be larger than that limit. The datagram needs to be fragmented to smaller units before being passed to the data link layer. Fragmentation needs to preserve the information at the header of the datagram. In other words, although the data can be fragmented, the header needs to be repeated. In addition, some more information needs to be added to the header to define the position of the fragment in the whole datagram. We discuss fragmentation in more detail in Chapter 7.

Services Provided at Each Router

As we have mentioned before, a router is involved with two interfaces with respect to a single datagram: the incoming interface and the outgoing interface. The network layer at the router, therefore, needs to interact with two data link layers: the data link of the incoming interface and the data link layer of the outgoing interface. The network layer is responsible to receive a datagram from the data link layer of the incoming interface, fragment it if necessary, and deliver the fragments to the data link of the outgoing interface. The router normally does not involve upper layers (with some exceptions discussed in future chapters). Figure 4.14 shows the services provides by a router at the network layer.

The three processes (finding next-hop logical address, finding next-hop MAC address, and fragmentation) here are the same as the last three processes mentioned for

Figure 4.14 *Processing at each router*

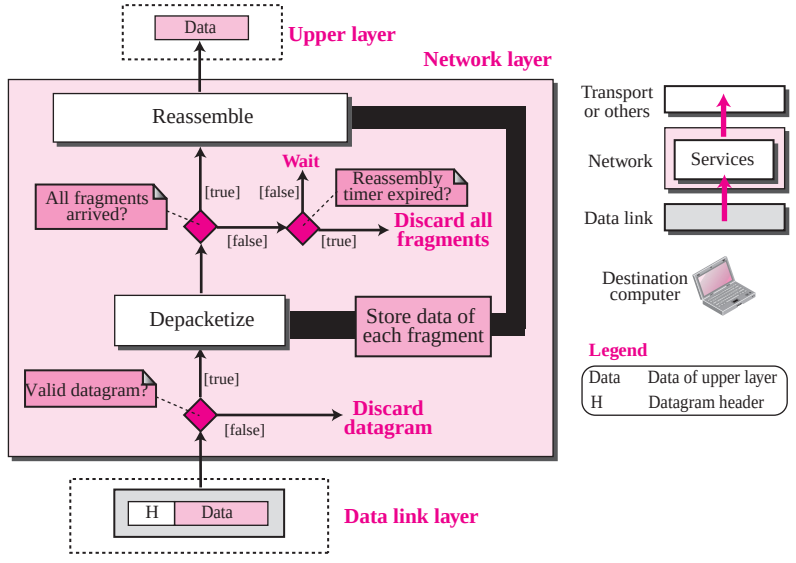


a source. Before applying these processes, however, the router needs to check the validity of the datagram using the checksum (see Chapter 7). Validation here means that the datagram header is not corrupted and the datagram is delivered to the correct router.

Services Provided at the Destination Computer

The network layer at the destination computer is simpler. No forwarding is needed. However, the destination computer needs to assemble the fragments before delivering the data to the destination. After validating each datagram, the data is extracted from each fragment and stored. When all fragments have arrived, the data are reassembled and delivered to the upper layer. The network layer also sets a reassembly timer. If the timer is expired, all data fragments are destroyed and an error message is sent that all the fragmented datagram need to be resent. Figure 4.15 shows the process. Note that the process of fragmentation is transparent to the upper layer because the network layer does not deliver any piece of data to the upper layer until all pieces have arrived and assembled. Since a datagram may have been fragmented in the source computer as well as in any router (multilevel) fragmentation, the reassembly procedure is very delicate and complicated. We discuss the procedure in more detail in Chapter 7.

Figure 4.15 *Processing at the destination computer*



4.5 OTHER NETWORK LAYER ISSUES

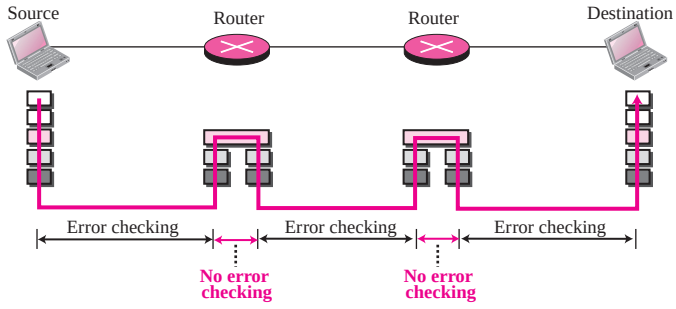
In this section we introduce some issues related to the network layer. These issues actually represent services that are normally discussed for the network layer, but they are either partially implemented at the network layer or not implemented at all. Some services are provided by some auxiliary protocols or by protocols added to the Internet later. Most of these issues resurface in future chapters.

Error Control

Error control means including a mechanism for detecting corrupted, lost, or duplicate datagrams. Error control also includes a mechanism for correcting errors after they have been detected. The network layer in the Internet does not provide a real error control mechanism. At the surface level, it looks as though there is no need for error control at the network layer because each datagram passes through several networks before reaching its final destination. The data link layer that controls the behavior of these networks (LANs or WANs) use error control. In other words, if a hop-to-hop error control is already implemented at the data link layer, why do we need error control at the network layer? Although hop-to-hop error control may protect a datagram to some extent, it does not provide full protection. Figure 4.16 shows that there are some areas in the path of the datagram that some errors may occur, but never checked; the error control at the data link layer can miss any error that occurs when the datagram is being processed by the router.

The designers of the network layer wanted to make this layer operate simply and fast. They thought if there is a need for more rigorous error checking, it can be done at

Figure 4.16 *Error checking at the data link layer*



the upper layer protocol that uses the service of the network layer. Another rationale for omitting the error checking at this layer can be related to fragmentation. Since the data is possibly fragmented at some routers and part of the network layer may be changed because of fragmentation, if we use error control, it must be checked at each router. This makes error checking at this layer very inefficient.

The designers of the network layer, however, have added a checksum field (see Chapter 7) to the datagram to control any corruption in the header, but not the whole datagram. This checksum may prevent any changes or corruptions in the header of the datagram between two hops and from end to end. For example, it prevents the delivery of the datagram to a wrong destination if the destination address has been corrupted. However, since the header may be changed in each router, the checksum needs to be calculated at the source and recalculated at each router.

We need to mention that although the network layer at the Internet does not directly provide error control, the Internet uses another protocol, ICMP, that provides some kind of error control if the datagram is discarded or has some unknown information in the header. We discuss ICMP in detail in Chapter 9.

Flow Control

Flow control regulates the amount of data a source can send without overwhelming the receiver. If the upper layer at the source computer produces data faster than the upper layer at the destination computer can consume it, the receiver will be overwhelmed with data. To control the flow of data, the receiver needs to send some feedback to the sender to inform the latter it is overwhelmed with data.

The network layer in the Internet, however, does not directly provide any flow control. The datagrams are sent by the sender when they are ready without any attention to the readiness of the receiver.

No flow control is provided for the current version of Internet network layer.

Probably a few reasons for the lack of flow control in the design of the network layer can be mentioned. First, since there is no error control in this layer, the job of the

network layer at the receiver is so simple that it may rarely be overwhelmed. Second, the upper layers that use the service of the network layer can implement buffers to receive data from the network layer as soon as they are ready and does not have to consume the data as fast as received. Second, the flow control is provided for most of the upper layer protocols that use the services of the network layer, so another level of flow control makes the network layer more complicated and the whole system less proficient.

Congestion Control

Another issue in a network layer protocol is **congestion control.** *Congestion* in the network layer is a situation in which too many datagrams are present in an area of the Internet. Congestion may occur if the number of datagrams sent by source computers are beyond the capacity of the network or routers. In this situation, some routers may drop some of the datagrams. However, as more datagrams are dropped, the situation may become worse because, due to the error control mechanism at the upper layers, the sender may send duplicates of the lost packets. If the congestion continues, sometimes a situation may reach a point that collapses the system and no datagram is delivered.

Congestion Control in a Connectionless Network

There are several ways to control congestion in a connectionless network. One solution is referred to as signaling. In backward signaling a bit can set in the datagram moving in the direction opposite to the congested direction to inform the sender that congestion has occurred and the sender needs to slow down the sending of packets. In this case, the bit can be set in the response to a packet or in a packet that acknowledges the packet. If no feedback (acknowledgment) is used at the network layer, but the upper layer uses feedback, forward signaling can be used in which a bit is set in the packet traveling in the direction of the congestion to warn the receiver of the packet about congestion. The receiver then may inform the upper layer protocol, which in turn may inform the source. No forward or backward signaling is used in the Internet network layer.

Congestion in a connectionless network can also be implemented using a **choke packet,** a special packet that can be sent from a router to the sender when it encounters congestion. This mechanism, in fact, is implemented in the Internet network layer. The network layer uses an auxiliary protocol, ICMP, which we discuss in Chapter 9. When a router is congested, it can send an ICMP packet to the source to slow down.

Another way to ameliorate the congestion is to rank the packets by their *importance* in the whole message. A field can be used in the header of a packet to define the rank of a datagram as more important or less important, for example. If a router is congested and needs to drop some packets, the packets marked as less important can be dropped. For example, if a message represents an image, it may be divided into many packets. Some packets, the one in the corner, may be less important than the ones representing the middle part of the image. If the router is congested, it can drop these less important packets without tremendously changing the quality of the image. We talk about these issues in Chapter 25 when we discuss multimedia communication.

Congestion Control in a Connection-Oriented Network

It is sometimes easier to control congestion in a connection-oriented network than in a connectionless network. One method simply creates an extra virtual circuit when there

is a congestion in an area. This, however, may create more problems for some routers. A better solution is *advanced negotiation* during the setup phase. The sender and the receiver may agree to a level of traffic when they setup the virtual circuit. The traffic level can be dictated by the routers that allow the establishment of the virtual circuits. In other words, a router can look at the exiting traffic and compare it with its maximum capacity, which allows a new virtual circuit to be created.

Quality of Service

As the Internet has allowed new applications such as multimedia communication (in particular real-time communication of audio and video), the **quality of service (QoS)** of the communication has become more and more important. The Internet has thrived to provide better quality of service to support these applications. However, to keep the network layer untouched, these provisions are mostly implemented in the upper layer. Since QoS manifests itself more when we use multimedia communication, we discuss this issue in Chapter 25 when we discuss multimedia.

Routing

A very important issue in the network layer is **routing;** how a router creates its routing table to help in forwarding a datagram in a connectionless service or helps in creating a virtual circuit, during setup phase, in a connection-oriented service. This can be done by *routing protocols*, that help hosts and routers make their routing table, maintain them, and update them. These are separate protocols that sometimes use the service of the network layer and sometimes the service of some transport layer protocols to help the network layer do its job. They can be grouped into two separate categories: unicast and multicast. We devote Chapter 11 to unicast routing and Chapter 12 to multicast routing. We need to assume that the routers already have created their routing protocols until we discuss these protocols in Chapters 11 and 12.

Security

Another issue related to the communication at the network layer is security. Security was not a concern when the Internet was originally designed because it was used by a small number of users at the universities to do research activities; other people had no access to the Internet. The network layer was designed with no security provision. Today, however, security is a big concern. To provide security for a connectionless network layer, we need to have another virtual level that changes the connectionless service to a connection-oriented service. This virtual layer, called IPSec, is discussed in Chapter 30 after we discuss the general principles of cryptography and security in Chapter 29.

4.6 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books: [Ste 94], [Tan 03], [Com 06], [Gar & Vid 04], and [Kur & Ros 08]. The items enclosed in brackets refer to the reference list at the end of the book.

4.7 KEY TERMS

choke packet
circuit switching
congestion control
connectionless service
connection-oriented service
data transfer phase
error control
flow control
packet switching
quality of service (QoS)
routing
setup phase
switching
teardown phase

4.8 SUMMARY

- At the conceptual level, we can think of the global Internet as a black box network. The Internet, however, is not one single network; it is made of many networks (or links) connected together through the connecting devices.
- In this Internet, a connecting device such as a router acts as a switch. Two types of switching are traditionally used in networking: circuit switching and packet switching.
- The network layer is designed as a packet-switched network. Packet-switched network can provide either a connectionless service or a connection-oriented service. When the network layer provides a connectionless service, each packet traveling in the Internet is an independent entity; there is no relationship between packets belonging to the same message. In a connection-oriented service, there is a virtual connection between all packets belonging to a message.
- In a connectionless service, the packets are forwarded to the next hop using the destination address in the packet. In a connection-oriented service, the packets are forwarded to the next hop using a label in the packet.
- In a connection-oriented network, communication occurs in three phases: setup, data transfer, and teardown. After connection setup, a virtual circuit is established between the sender and the receiver in which all packets belonging to the same message are sent through that circuit.
- We discussed existing services at the network layer in the Internet including addressing, services provided at the source computer, services provided at the destination computer, and services provided at each router.
- We also discussed some issues related to the network layer, services that are normally discussed for the network layer, but they are either partially implemented at the network layer or not implemented at all. Some of these services, such as routing and security are provided by other protocols in the Internet.

4.9 PRACTICE SET

Exercises

1. Give some advantages and disadvantages of the connectionless service.
2. Give some advantages and disadvantages of the connection-oriented service.

3. If a label in a connection-oriented service is *n* bits, how many virtual circuits can be established at the same time?
4. Assume a destination computer receives messages from several computers. How can it be sure that the fragments from one source is not mixed with the fragments from another source.
5. Assume a destination computer receives several packets from a source. How can it be sure that the fragments belonging to a datagram are not mixed from the fragments belonging to another datagram.
6. Why do you think that the packets in Figure 4.7 need both addresses and labels?
7. Compare and contrast the delays in connectionless and connection-oriented services. Which service creates less delay if the message is large? Which service creates less delay if the message is small?
8. In Figure 4.13, why should the fragmentation be the last service?
9. Discuss why we need fragmentation at each router.
10. Discuss why we need to do reassembly at the final destination, not at each router.
11. In Figure 4.15, why do we need to set a timer and destroy all fragments if the timer expires? What criteria do you use in selecting the expiration duration of such a timer?

CHAPTER

5

IPv4 Addresses

At the network layer, we need to uniquely identify each device on the Internet to allow global communication between all devices. In this chapter, we discuss the addressing mechanism related to the prevalent IPv4 protocol called IPv4 addressing. It is believed that IPv6 protocol will eventually supersede the current protocol, and we need to become aware of IPv6 addressing as well. We discuss IPv6 protocol and its addressing mechanism in Chapters 26 to 28.

OBJECTIVES

This chapter has several objectives:

- To introduce the concept of an address space in general and the address space of IPv4 in particular.
- To discuss the classful architecture, classes in this model, and the blocks of addresses available in each class.
- To discuss the idea of hierarchical addressing and how it has been implemented in classful addressing.
- To explain subnetting and supernetting for classful architecture and show how they were used to overcome the deficiency of classful addressing.
- To discuss the new architecture, classless addressing, that has been devised to solve the problems in classful addressing such as address depletion.
- To show how some ideas borrowed from classful addressing such as subnetting can be easily implemented in classless addressing.
- To discuss some special blocks and some special addresses in each block.
- To discuss NAT technology and show how it can be used to alleviate the shortage in number of addresses in IPv4.

5.1 INTRODUCTION

The identifier used in the IP layer of the TCP/IP protocol suite to identify each device connected to the Internet is called the Internet address or **IP address.** An IPv4 address is a 32-bit address that *uniquely* and *universally* defines the connection of a host or a router to the Internet; an IP address is the address of the interface.

An IPv4 address is 32 bits long.

IPv4 addresses are *unique*. They are unique in the sense that each address defines one, and only one, connection to the Internet. Two devices on the Internet can never have the same address at the same time. However, if a device has two connections to the Internet, via two networks, it has two IPv4 addresses. The IPv4 addresses are *universal* in the sense that the addressing system must be accepted by any host that wants to be connected to the Internet.

The IPv4 addresses are unique and universal.

Address Space

A protocol like IPv4 that defines addresses has an **address space.** An address space is the total number of addresses used by the protocol. If a protocol uses *b* bits to define an address, the address space is 2^b because each bit can have two different values (0 or 1). IPv4 uses 32-bit addresses, which means that the address space is 2^{32} or 4,294,967,296 (more than four billion). Theoretically, if there were no restrictions, more than 4 billion devices could be connected to the Internet.

The address space of IPv4 is 2^{32} or 4,294,967,296.

Notation

There are three common notations to show an IPv4 address: binary notation (base 2), dotted-decimal notation (base 256), and hexadecimal notation (base 16). The most prevalent, however, is base 256. These bases are defined in Appendix B. We also show how to convert a number from one base to another in that appendix. We recommend a review of this appendix before continuing with this chapter.

Numbers in base 2, 16, and 256 are discussed in Appendix B.

Binary Notation: Base 2

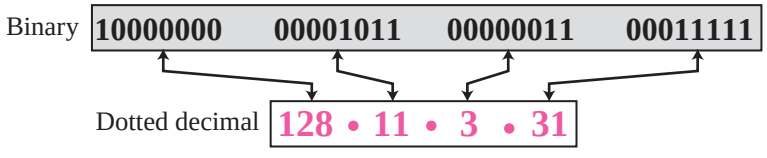
In **binary notation,** an IPv4 address is displayed as 32 bits. To make the address more readable, one or more spaces is usually inserted between each octet (8 bits). Each octet is often referred to as a byte. So it is common to hear an IPv4 address referred to as a 32-bit address, a 4-octet address, or a 4-byte address. The following is an example of an IPv4 address in binary notation:

```
01110101 10010101 00011101 11101010
```

Dotted-Decimal Notation: Base 256

To make the IPv4 address more compact and easier to read, an IPv4 address is usually written in decimal form with a decimal point (dot) separating the bytes. This format is referred to as **dotted-decimal notation.** Figure 5.1 shows an IPv4 address in dotted-decimal notation. Note that because each byte (octet) is only 8 bits, each number in the dotted-decimal notation is between 0 and 255.

Figure 5.1 *Dotted-decimal notation*



Example 5.1

Change the following IPv4 addresses from binary notation to dotted-decimal notation.

a. `10000001 00001011 00001011 11101111`
b. `11000001 10000011 00011011 11111111`
c. `11100111 11011011 10001011 01101111`
d. `11111001 10011011 11111011 00001111`

Solution

We replace each group of 8 bits with its equivalent decimal number (see Appendix B) and add dots for separation:

a. 129.11.11.239
b. 193.131.27.255
c. 231.219.139.111
d. 249.155.251.15

Example 5.2

Change the following IPv4 addresses from dotted-decimal notation to binary notation.

a. 111.56.45.78
b. 221.34.7.82

c. 241.8.56.12
d. 75.45.34.78

Solution
We replace each decimal number with its binary equivalent (see Appendix B):

a. **01101111 00111000 00101101 01001110**
b. **11011101 00100010 00000111 01010010**
c. **11110001 00001000 00111000 00001100**
d. **01001011 00101101 00100010 01001110**

Example 5.3

Find the error, if any, in the following IPv4 addresses:

a. 111.56.045.78
b. 221.34.7.8.20
c. 75.45.301.14
d. 11100010.23.14.67

Solution

a. There should be no leading zeroes in dotted-decimal notation (045).
b. We may not have more than 4 bytes in an IPv4 address.
c. Each byte should be less than or equal to 255; 301 is outside this range.
d. A mixture of binary notation and dotted-decimal notation is not allowed.

Hexadecimal Notation: Base 16

We sometimes see an IPv4 address in **hexadecimal notation.** Each hexadecimal digit is equivalent to four bits. This means that a 32-bit address has 8 hexadecimal digits. This notation is often used in network programming.

Example 5.4

Change the following IPv4 addresses from binary notation to hexadecimal notation.

a. **10000001 00001011 00001011 11101111**
b. **11000001 10000011 00011011 11111111**

Solution
We replace each group of 4 bits with its hexadecimal equivalent (see Appendix B). Note that hexadecimal notation normally has no added spaces or dots; however, 0X (or 0x) is added at the beginning or the subscript 16 at the end to show that the number is in hexadecimal.

a. 0X810B0BEF or $810B0BEF_{16}$
b. 0XC1831BFF or $C1831BFF_{16}$

Range of Addresses

We often need to deal with a range of addresses instead of one single address. We sometimes need to find the number of addresses in a range if the first and last address is given. Other times, we need to find the last address if the first address and the number of addresses in the range are given. In this case, we can perform subtraction or addition

operations in the corresponding base (2, 256, or 16). Alternatively, we can covert the addresses to decimal values (base 10) and perform operations in this base.

Example 5.5

Find the number of addresses in a range if the first address is 146.102.29.0 and the last address is 146.102.32.255.

Solution

We can subtract the first address from the last address in base 256 (see Appendix B). The result is 0.0.3.255 in this base. To find the number of addresses in the range (in decimal), we convert this number to base 10 and add 1 to the result.

$$\textbf{Number of addresses} = (0 \times 256^3 + 0 \times 256^2 + 3 \times 256^1 + 255 \times 256^0) + 1 = 1024$$

Example 5.6

The first address in a range of addresses is 14.11.45.96. If the number of addresses in the range is 32, what is the last address?

Solution

We convert the number of addresses minus 1 to base 256, which is 0.0.0.31. We then add it to the first address to get the last address. Addition is in base 256.

$$\textbf{Last address} = (14.11.45.96 + 0.0.0.31)_{256} = 14.11.45.127$$

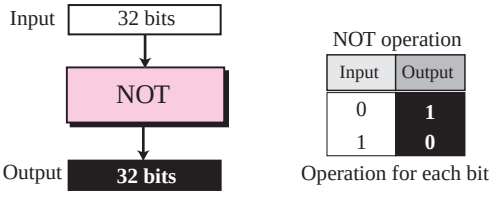
Operations

We often need to apply some operations on 32-bit numbers in binary or dotted-decimal notation. These numbers either represent IPv4 addresses or some entities related to IPv4 addresses (such as a *mask*, which is discussed later). In this section, we introduce three operations that are used later in the chapter: NOT, AND, and OR.

Bitwise NOT Operation

The bitwise NOT operation is a unary operation; it takes one input. When we apply the NOT operation on a number, it is often said that the number is complemented. The NOT operation, when applied to a 32-bit number in binary format, inverts each bit. Every 0 bit is changed to a 1 bit; every 1 bit is changed to a 0 bit. Figure 5.2 shows the NOT operation.

Figure 5.2 *Bitwise NOT operation*



Although we can directly use the NOT operation on a 32-bit number, when the number is represented as a four-byte dotted-decimal notation, we can use a short cut; we can subtract each byte from 255.

Example 5.7

The following shows how we can apply the NOT operation on a 32-bit number in binary.

Original number:	00010001	01111001	00001110	00100011
Complement:	11101110	10000110	11110001	11011100

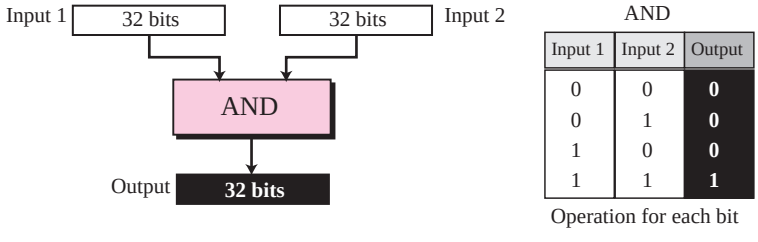
We can use the same operation using the dotted-decimal representation and the short cut.

Original number:	17	.	121	.	14	.	35
Complement:	238	.	134	.	241	.	220

Bitwise AND Operation

The bitwise AND operation is a binary operation; it takes two inputs. The AND operation compares the two corresponding bits in two inputs and selects the smaller bit from the two (or select one of them if the bits are equal). Figure 5.3 shows the AND operation.

Figure 5.3 *Bitwise AND operation*



Although we can directly use the AND operation on the 32-bit binary representation of two numbers, when the numbers are represented in dotted-decimal notation, we can use two short cuts.

1. When at least one of the numbers is 0 or 255, the AND operation selects the smaller byte (or one of them if equal).
2. When none of the two bytes is either 0 or 255, we can write each byte as the sum of eight terms, where each term is a power of 2. We then select the smaller term in each pair (or one of them if equal) and add them to get the result.

Example 5.8

The following shows how we can apply the AND operation on two 32-bit numbers in binary.

First number:	00010001	01111001	00001110	00100011
Second number:	11111111	11111111	10001100	00000000
Result	00010001	01111001	00001100	00000000

We can use the same operation using the dotted-decimal representation and the short cut.

First number:	17	.	121	.	14	.	35
Second number:	255	.	255	.	140	.	0
Result:	17	.	121	.	12	.	0

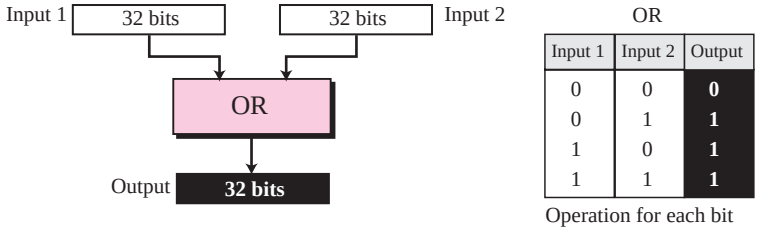
We have applied the first short cut on the first, second, and the fourth byte; we have applied the second short cut on the third byte. We have written 14 and 140 as the sum of terms and selected the smaller term in each pair as shown below.

Powers	2^7		2^6		2^5		2^4		2^3		2^2		2^1		2^0
Byte (14)	0	+	0	+	0	+	0	+	8	+	4	+	2	+	0
Byte (140)	128	+	0	+	0	+	0	+	8	+	4	+	0	+	0
Result (12)	0	+	0	+	0	+	0	+	8	+	4	+	0	+	0

Bitwise OR Operation

The bitwise OR operation is a binary operation; it takes two inputs. The OR operation compares the corresponding bits in the two numbers and selects the larger bit from the two (or one of them if equal). Figure 5.4 shows the OR operation.

Figure 5.4 *Bitwise OR operation*



OR

Input 1	Input 2	Output
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Operation for each bit

Although we can directly use the OR operation on the 32-bit binary representation of the two numbers, when the numbers are represented in dotted-decimal notation, we can use two short cuts.

1. When at least one of the two bytes is 0 or 255, the OR operation selects the larger byte (or one of them if equal).
2. When none of the two bytes is 0 or 255, we can write each byte as the sum of eight terms, where each term is a power of 2. We then select the larger term in each pair (or one of them if equal) and add them to get the result of OR operation.

Example 5.9

The following shows how we can apply the OR operation on two 32-bit numbers in binary.

First number:	00010001	01111001	00001110	00100011
Second number:	11111111	11111111	10001100	00000000
Result	11111111	11111111	10001110	00100011

We can use the same operation using the dotted-decimal representation and the short cut.

First number:	17	.	121	.	14	.	35
Second number:	255	.	255	.	140	.	0
Result:	255	.	255	.	142	.	35

We have used the first short cut for the first and second bytes and the second short cut for the third byte.

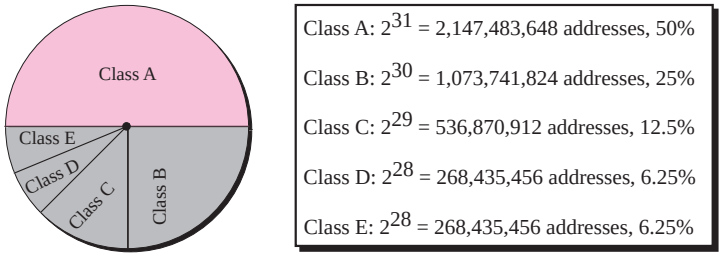
5.2 CLASSFUL ADDRESSING

IP addresses, when started a few decades ago, used the concept of *classes*. This architecture is called **classful addressing.** In the mid-1990s, a new architecture, called **classless addressing,** was introduced that supersedes the original architecture. In this section, we introduce classful addressing because it paves the way for understanding classless addressing and justifies the rationale for moving to the new architecture. Classless addressing is discussed in the next section.

Classes

In classful addressing, the IP address space is divided into five **classes: A, B, C, D,** and **E.** Each class occupies some part of the whole address space. Figure 5.5 shows the class occupation of the address space.

Figure 5.5 *Occupation of the address space*

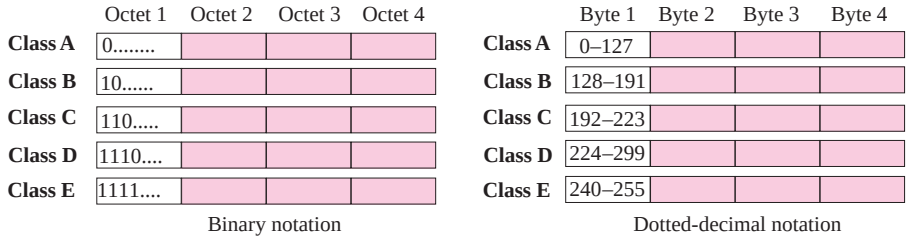


In classful addressing, the address space is divided into five classes: A, B, C, D, and E.

Recognizing Classes

We can find the class of an address when the address is given either in binary or dotted-decimal notation. In the binary notation, the first few bits can immediately tell us the class of the address; in the dotted-decimal notation, the value of the first byte can give the class of an address (Figure 5.6).

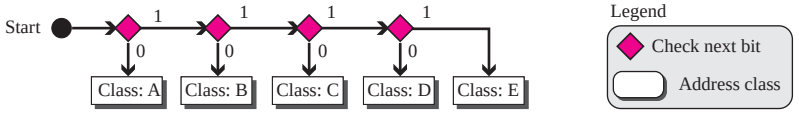
Figure 5.6 *Finding the class of an address*



Note that some special addresses fall in class A or E. We emphasize that these special addresses are exceptions to the classification; they are discussed later in the chapter.

Computers often store IPv4 addresses in binary notation. In this case, it is very convenient to write an algorithm to use a continuous checking process for finding the address as shown in Figure 5.7.

Figure 5.7 *Finding the address class using continuous checking*



Example 5.10

Find the class of each address:

a. **00000001 00001011 00001011 11101111**
b. **11000001 10000011 00011011 11111111**
c. **10100111 11011011 10001011 01101111**
d. **11110011 10011011 11111011 00001111**

Solution

See the procedure in Figure 5.7.

a. The first bit is 0. This is a class A address.
b. The first 2 bits are 1; the third bit is 0. This is a class C address.
c. The first bit is 1; the second bit is 0. This is a class B address.
d. The first 4 bits are 1s. This is a class E address.

Example 5.11

Find the class of each address:

a. 227.12.14.87
b. 193.14.56.22
c. 14.23.120.8
d. 252.5.15.111

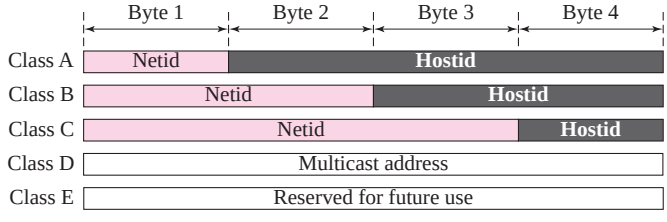
Solution

a. The first byte is 227 (between 224 and 239); the class is D.
b. The first byte is 193 (between 192 and 223); the class is C.
c. The first byte is 14 (between 0 and 127); the class is A.
d. The first byte is 252 (between 240 and 255); the class is E.

Netid and Hostid

In classful addressing, an IP address in classes A, B, and C is divided into **netid** and **hostid.** These parts are of varying lengths, depending on the class of the address. Figure 5.8 shows the netid and hostid bytes. Note that classes D and E are not divided into netid and hostid, for reasons that we will discuss later.

Figure 5.8 *Netid and hostid*



In class A, 1 byte defines the netid and 3 bytes define the hostid. In class B, 2 bytes define the netid and 2 bytes define the hostid. In class C, 3 bytes define the netid and 1 byte defines the hostid.

Classes and Blocks

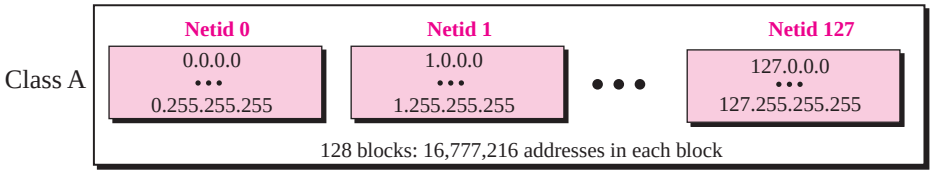
One problem with classful addressing is that each class is divided into a fixed number of blocks with each block having a fixed size. Let us look at each class.

Class A

Since only 1 byte in class A defines the netid and the leftmost bit should be 0, the next 7 bits can be changed to find the number of blocks in this class. Therefore, class A is divided into $2^7 = 128$ blocks that can be assigned to 128 organizations (the number is less because some blocks were reserved as special blocks). However, each block in this class contains 16,777,216 addresses, which means the organization should be a really

large one to use all these addresses. Many addresses are wasted in this class. Figure 5.9 shows the block in class A.

Figure 5.9 *Blocks in class A*

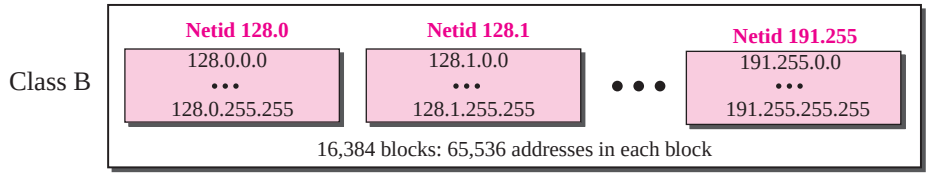


Millions of class A addresses are wasted.

Class B

Since 2 bytes in class B define the class and the two leftmost bit should be 10 (fixed), the next 14 bits can be changed to find the number of blocks in this class. Therefore, class B is divided into $2^{14} = 16{,}384$ blocks that can be assigned to 16,384 organizations (the number is less because some blocks were reserved as special blocks). However, each block in this class contains 65,536 addresses. Not so many organizations can use so many addresses. Many addresses are wasted in this class. Figure 5.10 shows the blocks in class B.

Figure 5.10 *Blocks in class B*

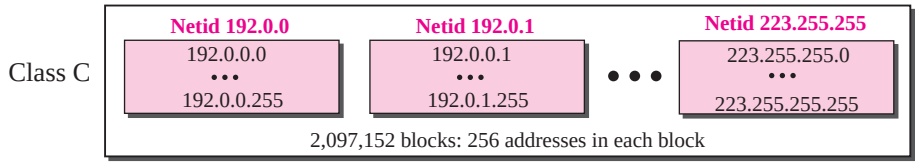


Many class B addresses are wasted.

Class C

Since 3 bytes in class C define the class and the three leftmost bits should be 110 (fixed), the next 21 bits can be changed to find the number of blocks in this class. Therefore, class C is divided into $2^{21} = 2{,}097{,}152$ blocks, in which each block contains 256 addresses, that can be assigned to 2,097,152 organizations (the number is less because some blocks were reserved as special blocks). Each block contains 256 addresses. However, not so many organizations were so small as to be satisfied with a class C block. Figure 5.11 shows the blocks in class C.

Figure 5.11 *Blocks in class C*

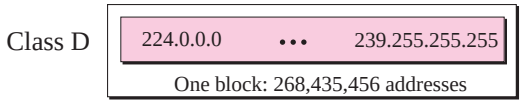


Not so many organizations are so small to have a class C block.

Class D

There is just one block of class D addresses. It is designed for multicasting, as we will see in a later section. Each address in this class is used to define one group of hosts on the Internet. When a group is assigned an address in this class, every host that is a member of this group will have a multicast address in addition to its normal (unicast) address. Figure 5.12 shows the block.

Figure 5.12 *The single block in Class D*

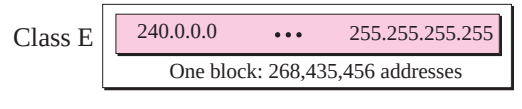


Class D addresses are made of one block, used for multicasting.

Class E

There is just one block of class E addresses. It was designed for use as reserved addresses, as shown in Figure 5.13.

Figure 5.13 *The single block in Class E*



The only block of class E addresses was reserved for future purposes.

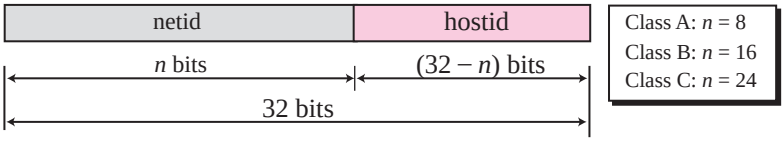
Two-Level Addressing

The whole purpose of IPv4 addressing is to define a destination for an Internet packet (at the network layer). When classful addressing was designed, it was assumed that the whole Internet is divided into many networks and each network connects many hosts. In other words, the Internet was seen as a network of networks. A network was normally created by an organization that wanted to be connected to the Internet. The Internet authorities allocated a block of addresses to the organization (in class A, B, or C).

The range of addresses allocated to an organization in classful addressing was a block of addresses in Class A, B, or C.

Since all addresses in a network belonged to a single block, each address in classful addressing contains two parts: netid and hostid. The netid defines the network; the hostid defines a particular host connected to that network. Figure 5.14 shows an IPv4 address in classful addressing. If *n* bits in the class defines the net, then 32 – *n* bits defines the host. However, the value of *n* depends on the class the block belongs to. The value of *n* can be 8, 16 or 24 corresponding to classes A, B, and C respectively.

Figure 5.14 *Two-level addressing in classful addressing*



Example 5.12

Two-level addressing can be found in other communication systems. For example, a telephone system inside the United States can be thought of as two parts: area code and local part. The area code defines the area, the local part defines a particular telephone subscriber in that area.

(626) 3581301

The area code, 626, can be compared with the netid, the local part, 3581301, can be compared to the hostid.

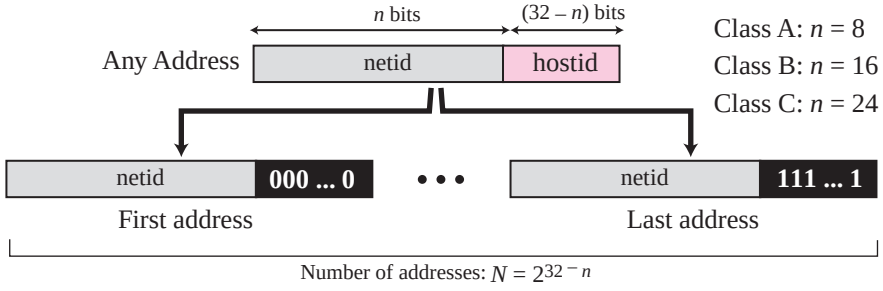
Extracting Information in a Block

A block is a range of addresses. Given any address in the block, we normally like to know three pieces of information about the block: the number of addresses, the first address, and the last address. Before we can extract these pieces of information, we need to know the class of the address, which we showed how to find in the previous section. After the class of the block is found, we know the value of *n*, the length of netid in bits. We can now find these three pieces of information as shown in Figure 5.15.

1. The number of addresses in the block, *N*, can be found using $N = 2^{32-n}$.

2. To find the first address, we keep the n leftmost bits and set the $(32 - n)$ rightmost bits all to 0s.
3. To find the last address, we keep the n leftmost bits and set the $(32 - n)$ rightmost bits all to 1s.

Figure 5.15 *Information extraction in classful addressing*



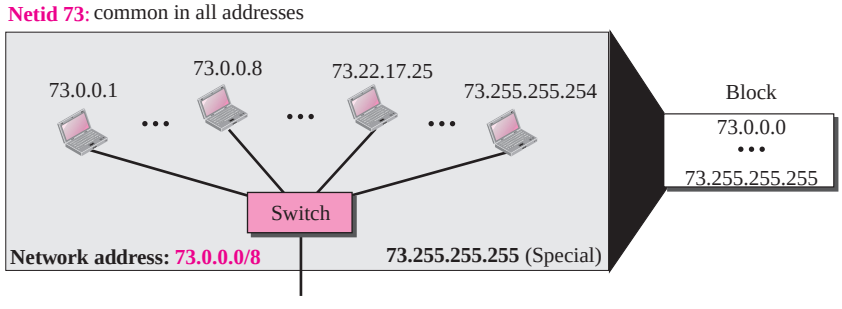
Example 5.13

An address in a block is given as 73.22.17.25. Find the number of addresses in the block, the first address, and the last address.

Solution

Since 73 is between 0 and 127, the class of the address is A. The value of n for class A is 8. Figure 5.16 shows a possible configuration of the network that uses this block. Note that we show the value of n in the network address after a slash. Although this was not a common practice in classful addressing, it helps to make it a habit in classless addressing in the next section.

Figure 5.16 *Solution to Example 5.13*



1. The number of addresses in this block is $N = 2^{32-n} = 2^{24} = 16{,}777{,}216$.
2. To find the first address, we keep the leftmost 8 bits and set the rightmost 24 bits all to 0s. The first address is 73.0.0.0/8 in which 8 is the value of n. The first address is called the *network address* and is not assigned to any host. It is used to define the network.

3. To find the last address, we keep the leftmost 8 bits and set the rightmost 24 bits all to 1s. The last address is 73.255.255.255. The last address is normally used for a special purpose, as discussed later in the chapter.

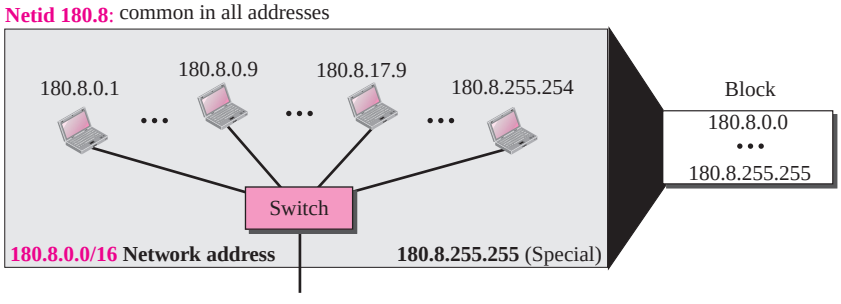
Example 5.14

An address in a block is given as 180.8.17.9. Find the number of addresses in the block, the first address, and the last address.

Solution

Since 180 is between 128 and 191, the class of the address is B. The value of *n* for class B is 16. Figure 5.17 shows a possible configuration of the network that uses this block.

Figure 5.17 *Solution to Example 5.14*



1. The number of addresses in this block is $N = 2^{32-n} = 2^{16} = 65{,}536$.
2. To find the first address, we keep the leftmost 16 bits and set the rightmost 16 bits all to 0s. The first address (network address) is 18.8.0.0/16, in which 16 is the value of *n*.
3. To find the last address, we keep the leftmost 16 bits and set the rightmost 16 bits all to 1s. The last address is 18.8.255.255.

Example 5.15

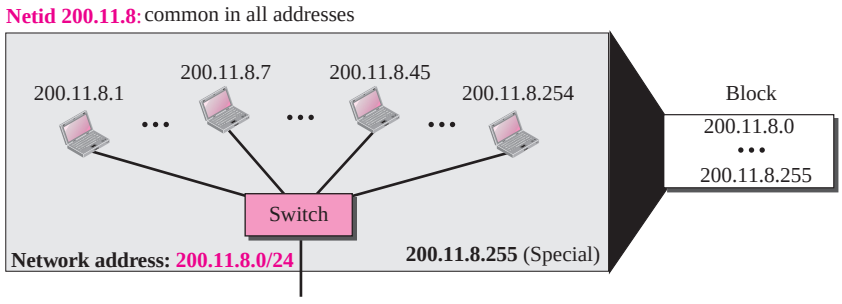
An address in a block is given as 200.11.8.45. Find the number of addresses in the block, the first address, and the last address.

Solution

Since 200 is between 192 and 223, the class of the address is C. The value of *n* for class C is 24. Figure 5.18 shows a possible configuration of the network that uses this block.

1. The number of addresses in this block is $N = 2^{32-n} = 2^{8} = 256$.
2. To find the first address, we keep the leftmost 24 bits and set the rightmost 8 bits all to 0s. The first address is 200.11.8.0/24. The first address is called the network address.
3. To find the last address, we keep the leftmost 24 bits and set the rightmost 8 bits all to 1s. The last address is 200.11.8.255.

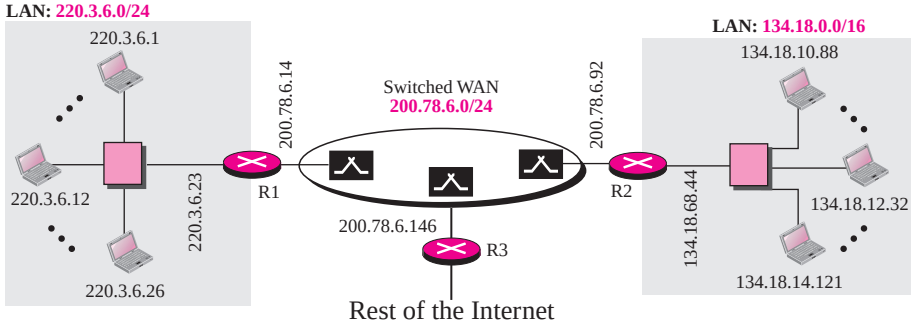
Figure 5.18 *Solution to Example 5.15*



An Example

Figure 5.19 shows a hypothetical part of an internet with three networks.

Figure 5.19 *Sample internet*



We have

1. A LAN with the network address 220.3.6.0 (class C).
2. A LAN with the network address 134.18.0.0 (class B).
3. A switched WAN (class C), such as Frame Relay or ATM, that can be connected to many routers. We have shown three. One router connects the WAN to the left LAN, one connects the WAN to the right LAN, and one connects the WAN to the rest of the internet.

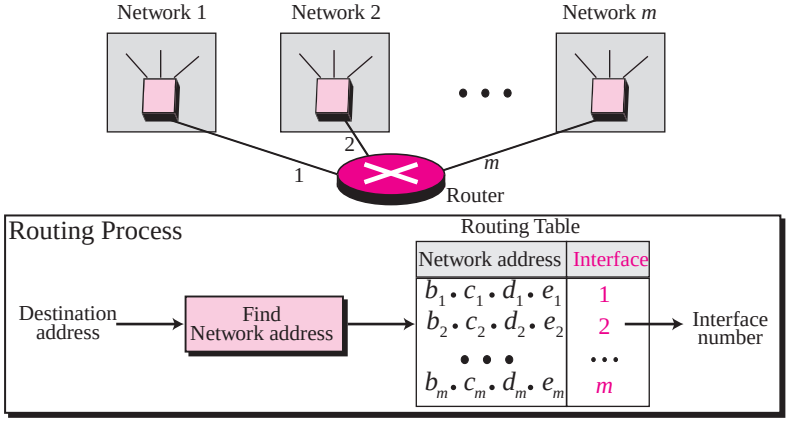
Network Address

The above three examples show that, given any address, we can find all information about the block. The first address, **network address,** is particularly important because it is used in routing a packet to its destination network. For the moment, let us assume that an internet is made of *m* networks and a router with *m* interfaces. When a packet arrives at the router from any source host, the router needs to know to which network the packet

should be sent; the router needs to know from which interface the packet should be sent out. When the packet arrives at the network, it reaches its destination host using another strategy that we discuss in later chapters. Figure 5.20 shows the idea. After the network address has been found, the router consults its routing table to find the corresponding interface from which the packet should be sent out. The network address is actually the identifier of the network; each network is identified by its network address.

The network address is the identifier of a network.

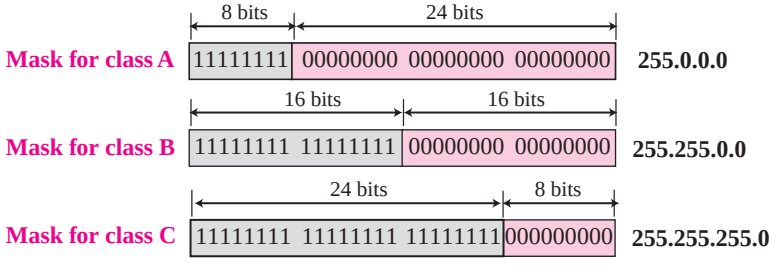
Figure 5.20 *Network address*



Network Mask

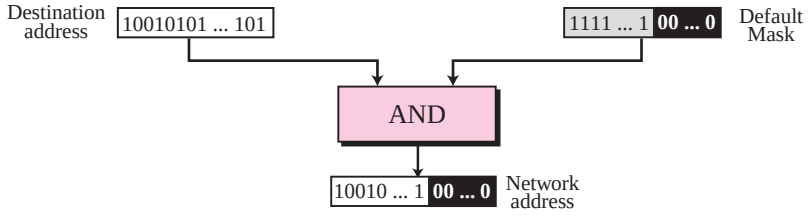
The methods we described previously for extracting the network address are mostly used to show the concept. The routers in the Internet normally use an algorithm to extract the network address from the destination address of a packet. To do this, we need a network mask. A **network mask** or a **default mask** in classful addressing is a 32-bit number with n leftmost bits all set to 1s and $(32 - n)$ rightmost bits all set to 0s. Since n is different for each class in classful addressing, we have three default masks in classful addressing as shown in Figure 5.21.

Figure 5.21 *Network mask*



To extract the network address from the destination address of a packet, a router uses the AND operation described in the previous section. When the destination address (or any address in the block) is ANDed with the default mask, the result is the network address (Figure 5.22). The router applies the AND operation on the binary (or hexadecimal representation) of the address and the mask, but when we show an example, we use the short cut discussed before and apply the mask on the dotted-decimal notation. The default mask can also be used to find the number of addresses in the block and the last address in the block, but we discuss these applications in classless addressing.

Figure 5.22 *Finding a network address using the default mask*



Example 5.16

A router receives a packet with the destination address 201.24.67.32. Show how the router finds the network address of the packet.

Solution

We assume that the router first finds the class of the address and then uses the corresponding default mask on the destination address, but we need to know that a router uses another strategy as we will discuss in the next chapter. Since the class of the address is B, we assume that the router applies the default mask for class B, 255.255.0.0 to find the network address.

Destination address	→	201	.	24	.	67	.	32
Default mask	→	255	.	255	.	0	.	0
Network address	→	201	.	24	.	0	.	0

We have used the first short cut as described in the previous section. The network address is 201.24.0.0 as expected.

Three-Level Addressing: Subnetting

As we discussed before, the IP addresses were originally designed with two levels of addressing. To reach a host on the Internet, we must first reach the network and then the host. It soon became clear that we need more than two hierarchical levels, for two reasons. First, an organization that was granted a block in class A or B needed to divide its large network into several subnetworks for better security and management. Second, since the blocks in class A and B were almost depleted and the blocks in class C were smaller than the needs of most organizations, an organization that has been granted a block in class A or B could divide the block into smaller subblocks and share them with

other organizations. The idea of splitting a block to smaller blocks is referred to as subnetting. In **subnetting,** a network is divided into several smaller subnetworks (subnets) with each subnetwork having its own subnetwork address.

Example 5.17

Three-level addressing can be found in the telephone system if we think about the local part of a telephone number as an exchange and a subscriber connection:

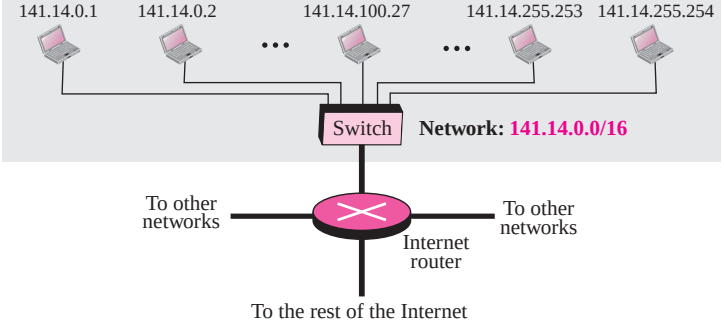
(626) 358 - 1301

in which 626 is the area code, 358 is the exchange, and 1301 is the subscriber connection.

Example 5.18

Figure 5.23 shows a network using class B addresses before subnetting. We have just one network with almost 2^{16} hosts. The whole network is connected, through one single connection, to one of the routers in the Internet. Note that we have shown /16 to show the length of the netid (class B).

Figure 5.23 *Example 5.18*



Example 5.19

Figure 5.24 shows the same network in Figure 5.23 after subnetting. The whole network is still connected to the Internet through the same router. However, the network has used a private router to divide the network into four subnetworks. The rest of the Internet still sees only one network; internally the network is made of four subnetworks. Each subnetwork can now have almost 2^{14} hosts. The network can belong to a university campus with four different schools (buildings). After subnetting, each school has its own subnetworks, but still the whole campus is one network for the rest of the Internet. Note that /16 and /18 show the length of the netid and subnetids.

Subnet Mask

We discussed the network mask (default mask) before. The network mask is used when a network is not subnetted. When we divide a network to several subnetworks, we need to create a subnetwork mask (or subnet mask) for each subnetwork. A subnetwork has subnetid and hostid as shown in Figure 5.25.

Figure 5.24 *Example 5.19*

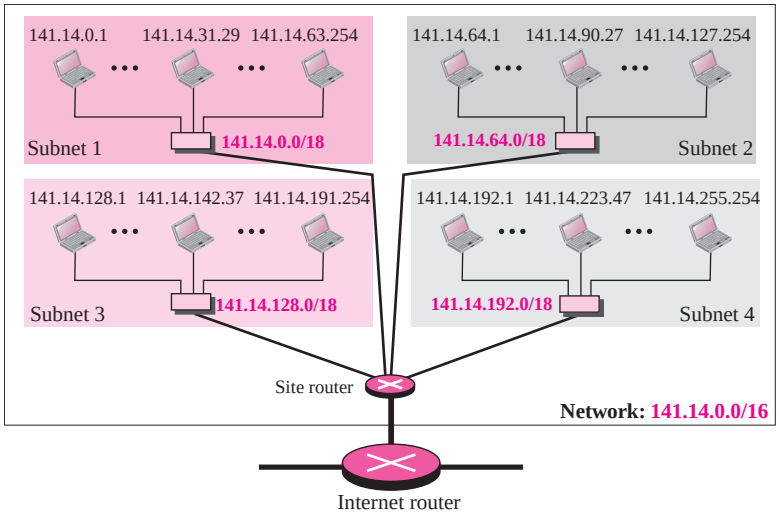
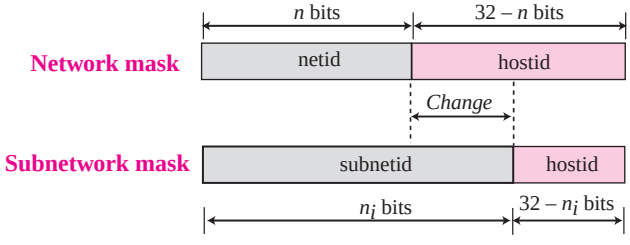


Figure 5.25 *Network mask and subnetwork mask*



Subnetting increases the length of the netid and decreases the length of hostid. When we divide a network to s number of subnetworks, each of equal numbers of hosts, we can calculate the subnetid for each subnetwork as

$$n_{sub} = n + \log_2 s$$

in which n is the length of netid, n_{sub} is the length of each subnetid, and s is the number of subnets which must be a power of 2.

Example 5.20

In Example 5.19, we divided a class B network into four subnetworks. The value of $n = 16$ and the value of $n_1 = n_2 = n_3 = n_4 = 16 + \log_2 4 = 18$. This means that the subnet mask has eighteen 1s and fourteen 0s. In other words, the subnet mask is 255.255.192.0 which is different from the network mask for class B (255.255.0.0).

Subnet Address

When a network is subnetted, the first address in the subnet is the identifier of the subnet and is used by the router to route the packets destined for that subnetwork. Given any address in the subnet, the router can find the subnet mask using the same procedure we discussed to find the network mask: ANDing the given address with the subnet mask. The short cuts we discussed in the previous section can be used to find the subnet address.

Example 5.21

In Example 5.19, we show that a network is divided into four subnets. Since one of the addresses in subnet 2 is 141.14.120.77, we can find the subnet address as:

Address	→	141	.	14	.	120	.	77
Mask	→	255	.	255	.	192	.	0
Subnet Address	→	141	.	14	.	64	.	0

The values of the first, second, and fourth bytes are calculated using the first short cut for AND operation. The value of the third byte is calculated using the second short cut for the AND operation.

Address (120)	0	+	64	+	32	+	16	+	8	+	0	+	0	+	0
Mask (192)	128	+	64	+	0	+	0	+	0	+	0	+	0	+	0
Result (64)	0	+	64	+	0	+	0	+	0	+	0	+	0	+	0

Designing Subnets

We show how to design a subnet when we discuss classless addressing. Since classful addressing is a special case of classless addressing, what is discussed later can also be applied to classful addressing.

Supernetting

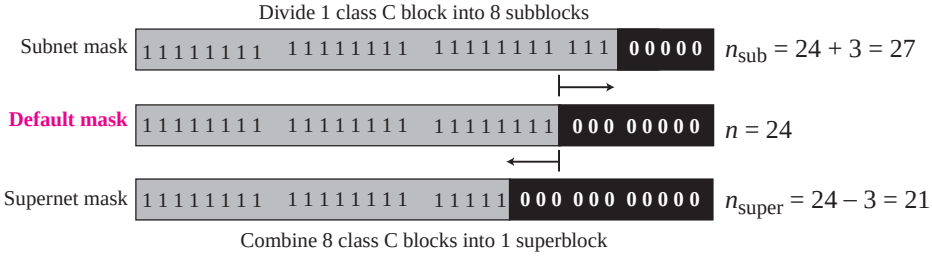
Subnetting could not completely solve address depletion problems in classful addressing because most organizations did not want to share their granted blocks with others. Since class C blocks were still available but the size of the block did not meet the requirement of new organizations that wanted to join the Internet, one solution was **supernetting.** In supernetting, an organization can combine several class C blocks to create a larger range of addresses. In other words, several networks are combined to create a supernetwork. By doing this, an organization can apply for several class C blocks instead of just one. For example, an organization that needs 1000 addresses can be granted four class C blocks.

Supernet Mask

A **supernet mask** is the reverse of a subnet mask. A subnet mask for class C has more 1s than the default mask for this class. A supernet mask for class C has less 1s than the default mask for this class.

Figure 5.26 shows the difference between a subnet mask and a supernet mask. A subnet mask that divides a block into eight subblocks has three more 1s ($2^3 = 8$) than the default mask; a supernet mask that combines eight blocks into one superblock has three less 1s than the default mask.

Figure 5.26 *Comparison of subnet, default, and supernet masks*



In supernetting, the number of class C addresses that can be combined to make a supernet needs to be a power of 2. The length of the supernetid can be found using the formula

$$n_{super} = n - \log_2 c$$

in which n_{super} defines the length of the supernetid in bits and c defines the number of class C blocks that are combined.

Unfortunately, supernetting provided two new problems: First, the number of blocks to combine needs to be a power of 2, which means an organization that needed seven blocks should be granted at least eight blocks (address wasting). Second, supernetting and subnetting really complicated the routing of packets in the Internet.

5.3 CLASSLESS ADDRESSING

Subnetting and supernetting in classful addressing did not really solve the address depletion problem and made the distribution of addresses and the routing process more difficult. With the growth of the Internet, it was clear that a larger address space was needed as a long-term solution. The larger address space, however, requires that the length of IP addresses to be increased, which means the format of the IP packets needs to be changed. Although the long-range solution has already been devised and is called IPv6 (see Chapters 26 to 28), a short-term solution was also devised to use the same address space but to change the distribution of addresses to provide a fair share to each organization. The short-term solution still uses IPv4 addresses, but it is called *classless* addressing. In other words, the class privilege was removed from the distribution to compensate for the address depletion.

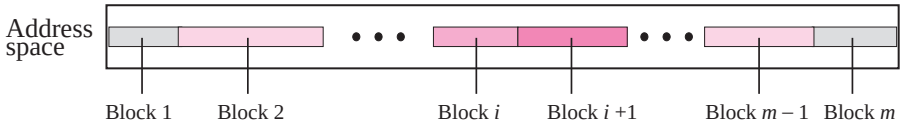
There was another motivation for classless addressing. During the 1990s, Internet service providers (ISPs) came into prominence. An ISP is an organization that provides Internet access for individuals, small businesses, and midsize organizations that do not want to create an Internet site and become involved in providing Internet services (such as e-mail services) for their employees. An ISP can provide these services. An ISP is granted a large range of addresses and then subdivides the addresses (in groups of 1, 2, 4, 8, 16, and so on), giving a range of addresses to a household or a small business. The customers are connected via a dial-up modem, DSL, or cable modem to the ISP. However, each customer needs some IPv4 addresses.

In 1996, the Internet authorities announced a new architecture called classless addressing. In classless addressing, variable-length blocks are used that belong to no classes. We can have a block of 1 address, 2 addresses, 4 addresses, 128 addresses, and so on.

Variable-Length Blocks

In classful addressing the whole address space was divided into five classes. Although each organization was granted one block in class A, B, or C, the size of the blocks was predefined; the organization needed to choose one of the three block sizes. The only block in class D and the only block in class E were reserved for a special purpose. In classless addressing, the whole address space is divided into variable length blocks. Theoretically, we can have a block of $2^0, 2^1, 2^2, \ldots, 2^{32}$ addresses. The only restriction, as we discuss later, is that the number of addresses in a block needs to be a power of 2. An organization can be granted one block of addresses. Figure 5.27 shows the division of the whole address space into nonoverlapping blocks.

Figure 5.27 *Variable-length blocks in classless addressing*

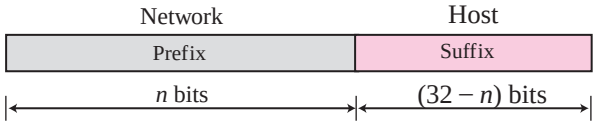


Two-Level Addressing

In classful addressing, two-level addressing was provided by dividing an address into *netid* and *hostid*. The netid defined the network; the hostid defined the host in the network. The same idea can be applied in classless addressing. When an organization is granted a block of addresses, the block is actually divided into two parts, the **prefix** and the **suffix.** The prefix plays the same role as the netid; the suffix plays the same role as the hostid. All addresses in the block have the same prefix; each address has a different suffix. Figure 5.28 shows the prefix and suffix in a classless block.

In classless addressing, the prefix defines the network and the suffix defines the host.

Figure 5.28 *Prefix and suffix*



In classful addressing, the length of the netid, *n*, depends on the class of the address; it can be only 8, 16, or 24. In classless addressing, the length of the prefix, *n*, depends on the size of the block; it can be 0, 1, 2, 3, . . . , 32. In classless addressing, the value of *n* is referred to as **prefix length;** the value of 32 – *n* is referred to as **suffix length.**

The prefix length in classless addressing can be 1 to 32.

Example 5.22

What is the prefix length and suffix length if the whole Internet is considered as one single block with 4,294,967,296 addresses?

Solution

In this case, the prefix length is 0 and the suffix length is 32. All 32 bits vary to define 2^{32} = 4,294,967,296 hosts in this single block.

Example 5.23

What is the prefix length and suffix length if the Internet is divided into 4,294,967,296 blocks and each block has one single address?

Solution

In this case, the prefix length for each block is 32 and the suffix length is 0. All 32 bits are needed to define 2^{32} = 4,294,967,296 blocks. The only address in each block is defined by the block itself.

Example 5.24

The number of addresses in a block is inversely related to the value of the prefix length, *n*. A small *n* means a larger block; a large *n* means a small block.

Slash Notation

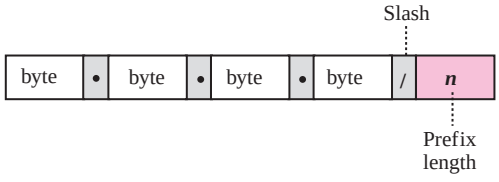
The netid length in classful addressing or the prefix length in classless addressing play a very important role when we need to extract the information about the block from a given address in the block. However, there is a difference here in classful and classless addressing.

- In classful addressing, the netid length is inherent in the address. Given an address, we know the class of the address that allows us to find the netid length (8, 16, or 24).

- In classless addressing, the prefix length cannot be found if we are given only an address in the block. The given address can belong to a block with any prefix length.

In classless addressing, we need to include the prefix length to each address if we need to find the block of the address. In this case, the prefix length, *n*, is added to the address separated by a slash. The notation is informally referred to as **slash notation.** An address in classless addressing can then be represented as shown in Figure 5.29.

Figure 5.29 *Slash notation*



The slash notation is formally referred to as **classless interdomain routing** or **CIDR** (pronounced cider) notation.

In classless addressing, we need to know one of the addresses in the block and the prefix length to define the block.

Example 5.25

In classless addressing, an address cannot per se define the block the address belongs to. For example, the address 230.8.24.56 can belong to many blocks some of them are shown below with the value of the prefix associated with that block:

Prefix length:16	→	Block:	230.8.0.0	to	230.8.255.255
Prefix length:20	→	Block:	230.8.16.0	to	230.8.31.255
Prefix length:26	→	Block:	230.8.24.0	to	230.8.24.63
Prefix length:27	→	Block:	230.8.24.32	to	230.8.24.63
Prefix length:29	→	Block:	230.8.24.56	to	230.8.24.63
Prefix length:31	→	Block:	230.8.24.56	to	230.8.24.57

Network Mask

The idea of network mask in classless addressing is the same as the one in classful addressing. A network mask is a 32-bit number with the *n* leftmost bits all set to 0s and the rest of the bits all set to 1s.

Example 5.26

The following addresses are defined using slash notations.

a. In the address 12.23.24.78/8, the network mask is 255.0.0.0. The mask has eight 1s and twenty-four 0s. The prefix length is 8; the suffix length is 24.

b. In the address 130.11.232.156/16, the network mask is 255.255.0.0. The mask has sixteen 1s and sixteen 0s.The prefix length is 16; the suffix length is 16.

c. In the address 167.199.170.82/27, the network mask is 255.255.255.224. The mask has twenty-seven 1s and five 0s. The prefix length is 27; the suffix length is 5.

Extracting Block Information

An address in slash notation (CIDR) contains all information we need about the block: the first address (network address), the number of addresses, and the last address. These three pieces of information can be found as follows:

- The number of addresses in the block can be found as:

$$N = 2^{32-n}$$

in which n is the prefix length and N is the number of addresses in the block.

- The first address (network address) in the block can be found by ANDing the address with the network mask:

First address = (any address) AND (network mask)

Alternatively, we can keep the n leftmost bits of any address in the block and set the 32– n bits to 0s to find the first address.

- The last address in the block can be found by either adding the first address with the number of addresses or, directly, by ORing the address with the complement (NOTing) of the network mask:

Last address = (any address) OR [NOT (network mask)]

Alternatively, we can keep the n leftmost bits of any address in the block and set the 32 – n bits to 1s to find the last address.

Example 5.27

One of the addresses in a block is 167.199.170.82/27. Find the number of addresses in the network, the first address, and the last address.

Solution

The value of n is 27. The network mask has twenty-seven 1s and five 0s. It is 255.255.255.240.

a. The number of addresses in the network is $2^{32-n} = 2^{32-n} = 2^5 = 32$.

b. We use the AND operation to find the first address (network address).The first address is 167.199.170.64/27.

Address in binary:	**10100111 11000111 10101010 01010010**
Network mask:	**11111111 11111111 11111111 11100000**
First address:	**10100111 11000111 10101010 01000000**

c. To find the last address, we first find the complement of the network mask and then OR it with the given address: The last address is 167.199.170.95/27.

Address in binary:	10100111	11000111	10101010	01010010
Complement of network mask:	00000000	00000000	00000000	00011111
Last address:	10100111	11000111	10101010	01011111

Example 5.28

One of the addresses in a block is 17.63.110.114/24. Find the number of addresses, the first address, and the last address in the block.

Solution

The network mask is 255.255.255.0.

a. The number of addresses in the network is $2^{32-24} = 256$.

b. To find the first address, we use the short cut methods discussed early in the chapter.

Address:	17	63	110	114
Network mask:	255	255	255	0
First address (AND):	17	63	110	0

The first address is 17.63.110.0/24.

c. To find the last address, we use the complement of the network mask and the first short cut method we discussed before. The last address is 17.63.110.255/24.

Address:	17	63	110	114
Complement of the mask (NOT):	0	0	0	255
Last address (OR):	17	63	110	255

Example 5.29

One of the addresses in a block is 110.23.120.14/20. Find the number of addresses, the first address, and the last address in the block.

Solution

The network mask is 255.255.240.0.

a. The number of addresses in the network is $2^{32-20} = 4096$.

b. To find the first address, we apply the first short cut to bytes 1, 2, and 4 and the second short cut to byte 3. The first address is 110.23.112.0/20.

Address:	110	23	120	14
Network mask:	255	255	240	0
First address (AND):	110	23	112	0

c. To find the last address, we apply the first short cut to bytes 1, 2, and 4 and the second short cut to byte 3. The OR operation is applied to the complement of the mask. The last address is 110.23.127.255/20.

Address:	110	.	23	.	120	.	14
Network mask:	0	.	0	.	15	.	255
Last address (OR):	110	.	23	.	127	.	255

Block Allocation

The next issue in classless addressing is block allocation. How are the blocks allocated? The ultimate responsibility of block allocation is given to a global authority called the Internet Corporation for Assigned Names and Addresses (ICANN). However, ICANN does not normally allocate addresses to individual Internet users. It assigns a large block of addresses to an ISP (or a larger organization that is considered an ISP in this case). For the proper operation of the CIDR, three restrictions need to be applied to the allocated block.

1. The number of requested addresses, N, needs to be a power of 2. This is needed to provide an integer value for the prefix length, n (see the second restriction). The number of addresses can be 1, 2, 4, 8, 16, and so on.
2. The value of prefix length can be found from the number of addresses in the block. Since $N = 2^{32 - n}$, then $n = \log_2 (2^{32}/N) = 32 - \log_2 N$. That is the reason why N needs to be a power of 2.
3. The requested block needs to be allocated where there are a contiguous number of unallocated addresses in the address space. However, there is a restriction on choosing the beginning addresses of the block. The beginning address needs to be divisible by the number of addresses in the block. To see this restriction, we can show that the beginning address can be calculated as $X \times 2^{n - 32}$ in which X is the decimal value of the prefix. In other words, the beginning address is $X \times N$.

Example 5.30

An ISP has requested a block of 1000 addresses. The following block is granted.

a. Since 1000 is not a power of 2, 1024 addresses are granted ($1024 = 2^{10}$).
b. The prefix length for the block is calculated as $n = 32 - \log_2 1024 = 22$.
c. The beginning address is chosen as 18.14.12.0 (which is divisible by 1024).

The granted block is 18.14.12.0/22. The first address is 18.14.12.0/22 and the last address is 18.14.15.255/22.

Relation to Classful Addressing

All issues discussed for classless addressing can be applied to classful addressing. As a matter of fact, classful addressing is a special case of the classless addressing in which the blocks in class A, B, and C have the prefix length $n_A = 8$, $n_B = 16$, and $n_C = 24$. A block in classful addressing can be easily changed to a block in class addressing if we use the prefix length defined in Table 5.1.

Table 5.1 *Prefix length for classful addressing*

Class	*Prefix length*	*Class*	*Prefix length*
A	/8	D	/4
B	/16	E	/4
C	/24		

Example 5.31

Assume an organization has given a class A block as 73.0.0.0 in the past. If the block is not revoked by the authority, the classless architecture assumes that the organization has a block 73.0.0.0/8 in classless addressing.

Subnetting

Three levels of hierarchy can be created using subnetting. An organization (or an ISP) that is granted a range of addresses may divide the range into several subranges and assign each subrange to a **subnetwork** (or **subnet**). The concept is the same as we discussed for classful addressing. Note that nothing stops the organization from creating more levels. A subnetwork can be divided into several sub-subnetworks. A sub-subnetwork can be divided into several sub-sub-subnetworks. And so on.

Designing Subnets

The subnetworks in a network should be carefully designed to enable the routing of packets. We assume the total number of addresses granted to the organization is N, the prefix length is n, the assigned number of addresses to each subnetwork is N_{sub}, the prefix length for each subnetwork is n_{sub}, and the total number of subnetworks is s. Then, the following steps need to be carefully followed to guarantee the proper operation of the subnetworks.

1. The number of addresses in each subnetwork should be a power of 2.
2. The prefix length for each subnetwork should be found using the following formula:

$$n_{sub} = n + \log_2 (N/N_{sub})$$

3. The starting address in each subnetwork should be divisible by the number of addresses in that subnetwork. This can be achieved if we first assign addresses to larger networks.

The restrictions applied in allocating addresses for a subnetwork are parallel to the ones used to allocate addresses for a network.

Finding Information about Each Subnetwork

After designing the subnetworks, the information about each subnetwork, such as first and last address, can be found using the process we described to find the information about each network in the Internet.

Example 5.32

An organization is granted the block 130.34.12.64/26. The organization needs four subnetworks, each with an equal number of hosts. Design the subnetworks and find the information about each network.

Solution

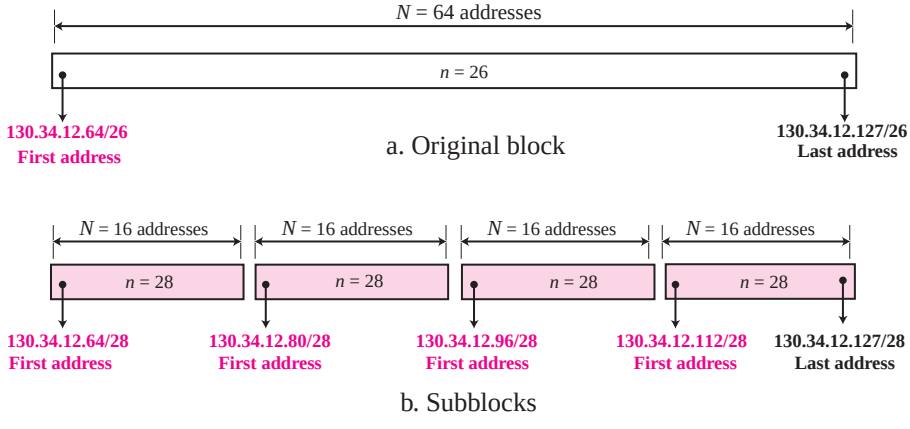
The number of addresses for the whole network can be found as $N = 2^{32 - 26} = 64$. Using the process described in the previous section, the first address in the network is 130.34.12.64/26 and the last address is 130.34.12.127/26. We now design the subnetworks:

1. We grant 16 addresses for each subnetwork to meet the first requirement (64/16 is a power of 2).
2. The **subnetwork** mask for each subnetwork is:

$$n_1 = n_2 = n_3 = n_4 = n + \log_2 (N/N_i) = 26 + \log_2 4 = 28$$

3. We grant 16 addresses to each subnet starting from the first available address. Figure 5.30 shows the subblock for each subnet. Note that the starting address in each subnetwork is divisible by the number of addresses in that subnetwork.

Figure 5.30 *Solution to Example 5.32*



Example 5.33

An organization is granted a block of addresses with the beginning address 14.24.74.0/24. The organization needs to have 3 subblocks of addresses to use in its three subnets as shown below:

- One subblock of 120 addresses.
- One subblock of 60 addresses.
- One subblock of 10 addresses.

Solution

There are $2^{32 - 24} = 256$ addresses in this block. The first address is 14.24.74.0/24; the last address is 14.24.74.255/24.

a. The number of addresses in the first subblock is not a power of 2. We allocate 128 addresses. The first can be used as network address and the last as the special address. There are still 126 addresses available. The subnet mask for this subnet can be found as $n_1 = 24 + \log_2 (256/128) = 25$. The first address in this block is 14.24.74.0/25; the last address is 14.24.74.127/25.

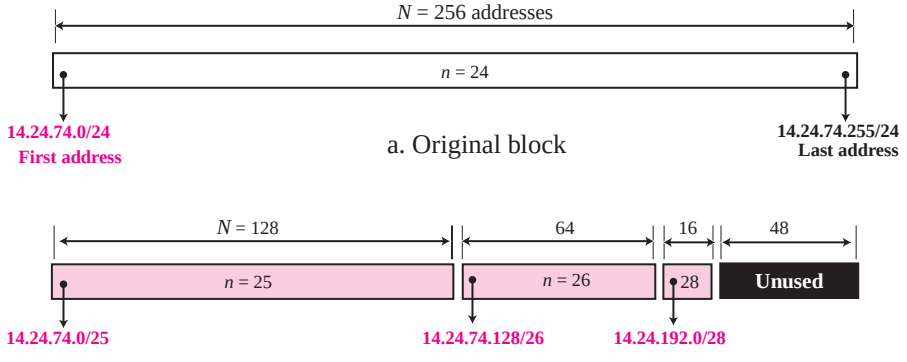
b. The number of addresses in the second subblock is not a power of 2 either. We allocate 64 addresses. The first can be used as network address and the last as the special address. There are still 62 addresses available. The subnet mask for this subnet can be found as $n_1 = 24 + \log_2 (256/64) = 26$. The first address in this block is 14.24.74.128/26; the last address is 14.24.74.191/26.

c. The number of addresses in the third subblock is not a power of 2 either. We allocate 16 addresses. The first can be used as network address and the last as the special address. There are still 14 addresses available. The subnet mask for this subnet can be found as $n_1 = 24 + \log_2 (256/16) = 28$. The first address in this block is 14.24.74.192/28; the last address is 14.24.74.207/28.

d. If we add all addresses in the previous subblocks, the result is 208 addresses, which means 48 addresses are left in reserve. The first address in this range is 14.24.74.209. The last address is 14.24.74.255. We don't know about the prefix length yet.

e. Figure 5.31 shows the configuration of blocks. We have shown the first address in each block.

Figure 5.31 *Solution to Example 5.33*



a. Original block

b. Subblocks

Example 5.34

Assume a company has three offices: Central, East, and West. The Central office is connected to the East and West offices via private, point-to-point WAN lines. The company is granted a block of 64 addresses with the beginning address 70.12.100.128/26. The management has decided to allocate 32 addresses for the Central office and divides the rest of addresses between the two other offices.

1. The number of addresses are assigned as follows:

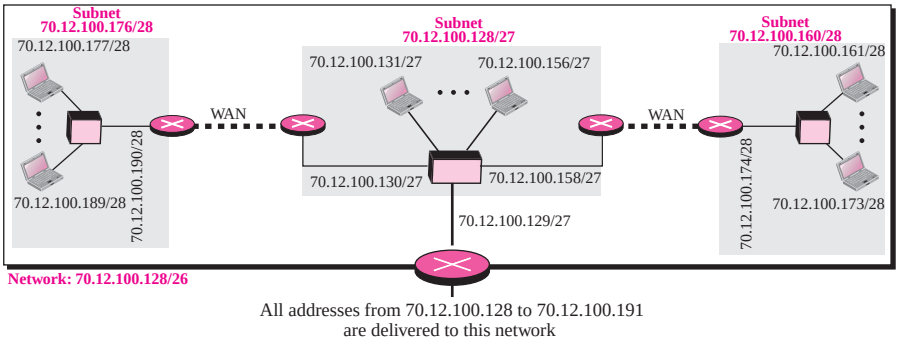
Central office $N_c = 32$	East office $N_e = 16$	West office $N_w = 16$

2. We can find the prefix length for each subnetwork:

$n_c = n + \log_2(64/32) = 27$	$n_e = n + \log_2(64/16) = 28$	$n_w = n + \log_2(64/16) = 28$

3. Figure 5.32 shows the configuration designed by the management. The Central office uses addresses 70.12.100.128/27 to 70.12.100.159/27. The company has used three of these addresses for the routers and has reserved the last address in the subblock. The East office uses the addresses 70.12.100.160/28 to 70.12.100.175/28. One of these addresses is used for the router and the company has reserved the last address in the subblock. The West office uses the addresses 70.12.100.160/28 to 70.12.100.175/28. One of these addresses is used for the router and the company has reserved the last address in the subblock. The company uses no address for the point-to-point connections in WANs.

Figure 5.32 *Example 14*



All addresses from 70.12.100.128 to 70.12.100.191 are delivered to this network

Address Aggregation

One of the advantages of CIDR architecture is **address aggregation.** ICANN assigns a large **block of addresses** to an ISP. Each ISP in turn divides its assigned block into smaller subblocks and grants the subblocks to its customers; many blocks of addresses are aggregated in one block and granted to one ISP.

Example 5.35

An ISP is granted a block of addresses starting with 190.100.0.0/16 (65,536 addresses). The ISP needs to distribute these addresses to three groups of customers as follows:

- The first group has 64 customers; each needs approximately 256 addresses.
- The second group has 128 customers; each needs approximately 128 addresses.
- The third group has 128 customers; each needs approximately 64 addresses.

We design the subblocks and find out how many addresses are still available after these allocations.

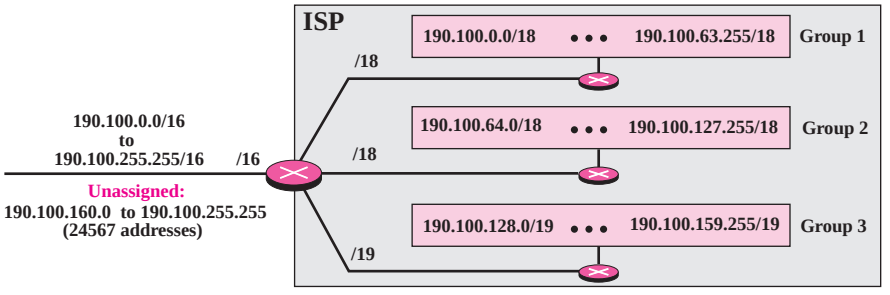
Solution

Let us solve the problem in two steps. In the first step, we allocate a subblock of addresses to each group. The total number of addresses allocated to each group and the prefix length for each subblock can found as

Group 1: $64 \times 256 = 16{,}384$	$n_1 = 16 + \log_2 (65536/16384) = 18$
Group 2: $128 \times 128 = 16{,}384$	$n_2 = 16 + \log_2 (65536/16384) = 18$
Group 3: $128 \times 64 = 8192$	$n_3 = 16 + \log_2 (65536/8192) = 19$

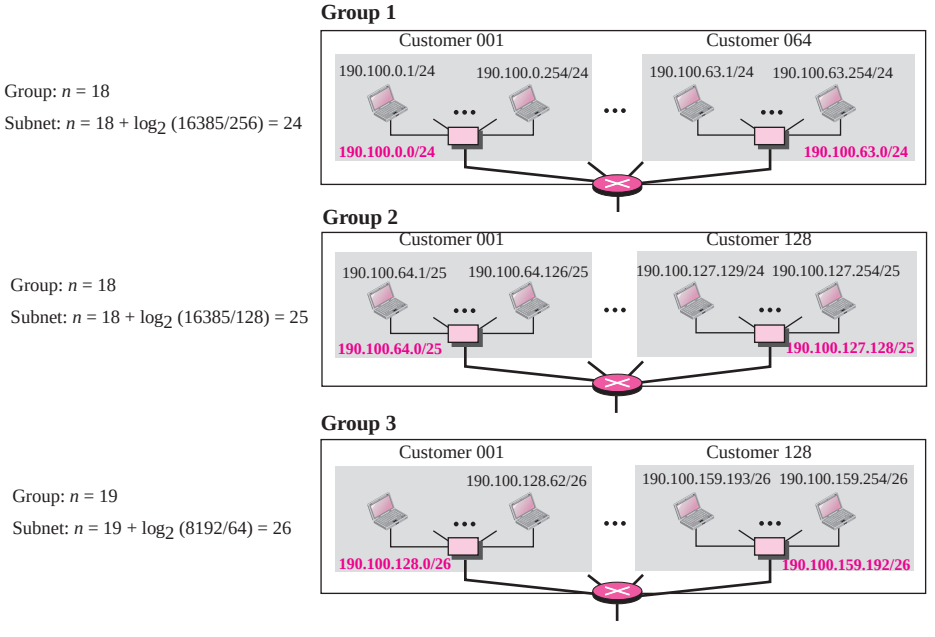
Figure 5.33 shows the design for the first hierarchical level.

Figure 5.33 *Solution to Example 5.35: first step*



Now we can think about each group. The prefix length changes for the networks in each group depending on the number of addresses used in each network. Figure 5.34 shows the second level of the hierarchy. Note that we have used the first address for each customer as the subnet address and have reserved the last address as a special address.

Figure 5.34 *Solution to Example 5.35: second step*



5.4 SPECIAL ADDRESSES

In classful addressing some addresses were reserved for special purposes. The classless addressing scheme inherits some of these special addresses from classful addressing.

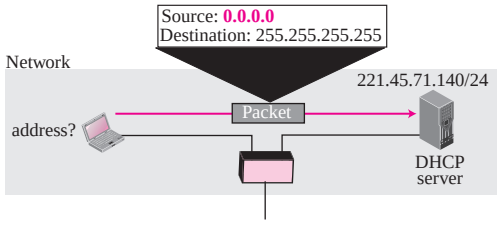
Special Blocks

Some blocks of addresses are reserved for special purposes.

All-Zeros Address

The block 0.0.0.0/32, which contains only one single address, is reserved for communication when a host needs to send an IPv4 packet but it does not know its own address. This is normally used by a host at bootstrap time when it does not know its IPv4 address. The host sends an IPv4 packet to a bootstrap server (called DHCP server as discussed in Chapter 18) using this address as the source address and a **limited broadcast address** as the destination address to find its own address (see Figure 5.35).

Figure 5.35 *Examples of using the all-zeros address*



All-Ones Address: Limited Broadcast Address

The block 255.255.255.255/32, which contains one single address, is reserved for limited broadcast address in the current network. A host that wants to send a message to every other host can use this address as a destination address in an IPv4 packet. However, a router will block a packet having this type of address to confine the broadcasting to the local network. In Figure 5.36, a host sends a datagram using a destination IPv4 address consisting of all 1s. All devices on this network receive and process this datagram.

Loopback Addresses

The block 127.0.0.0/8 is used for the **loopback address,** which is an address used to test the software on a machine. When this address is used, a packet never leaves the machine; it simply returns to the protocol software. It can be used to test the IPv4 software. For example, an application such as "ping" can send a packet with a loopback address as the destination address to see if the IPv4 software is capable of receiving and processing a packet. As another example, the loopback address can be used by a *client process* (a running application program) to send a message to a server process on the same machine. Note that this can be used only as a destination address in an IPv4 packet (see Figure 5.37).

Figure 5.36 *Example of limited broadcast address*

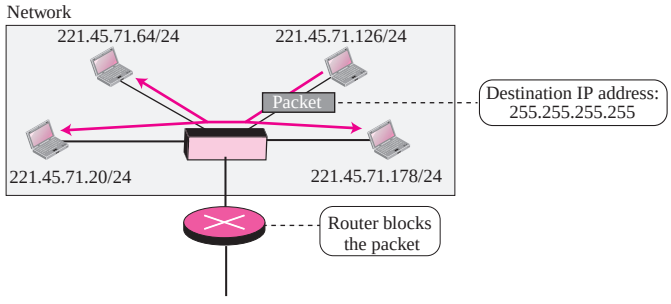
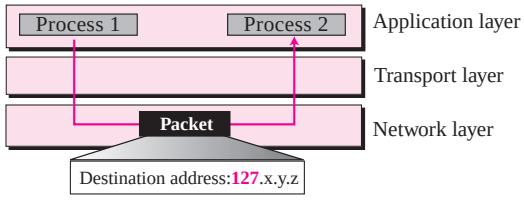


Figure 5.37 *Example of loopback address*



Private Addresses

A number of blocks are assigned for private use. They are not recognized globally. These blocks are depicted in Table 5.2. These addresses are used either in isolation or in connection with network address translation techniques (see NAT section later in this chapter).

Table 5.2 *Addresses for private networks*

Block	*Number of addresses*	*Block*	*Number of addresses*
10.0.0.0/8	16,777,216	192.168.0.0/16	65,536
172.16.0.0/12	1,047,584	169.254.0.0/16	65,536

Multicast Addresses

The block 224.0.0.0/4 is reserved for multicast communication. We discuss multicasting in Chapter 12 in detail.

Special Addresses in Each block

It is not mandatory, but it is recommended, that some addresses in a block be used for special addresses. These addresses are not assigned to any host. However, if a block (or subblock) is so small, we cannot afford to use part of the addresses as special addresses.

Network Address

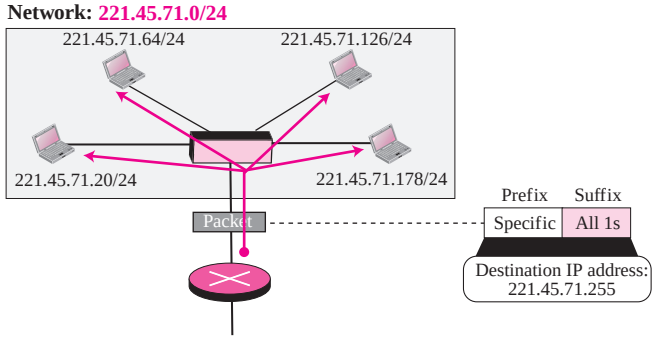
We have already discussed network addresses. The first address (with the suffix set all to 0s) in a block defines the network address. It actually defines the network itself

(cabling) and not any host in the network. Of course, the first address in a subnetwork is called the subnetwork address and plays the same role.

Direct Broadcast Address

The last address in a block or subblock (with the suffix set all to 1s) can be used as a **direct broadcast address.** This address is usually used by a router to send a packet to all hosts in a specific network. All hosts will accept a packet having this type of destination address. Note that this address can be used only as a destination address in an IPv4 packet. In Figure 5.38, the router sends a datagram using a destination IPv4 address with a suffix of all 1s. All devices on this network receive and process the datagram.

Figure 5.38 *Example of a direct broadcast address*



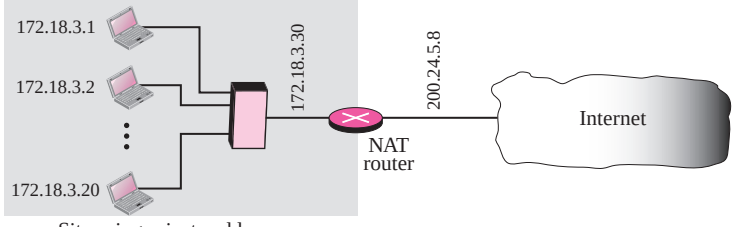
5.5 NAT

The distribution of addresses through ISPs has created a new problem. Assume that an ISP has granted a small range of addresses to a small business or a household. If the business grows or the household needs a larger range, the ISP may not be able to grant the demand because the addresses before and after the range may have already been allocated to other networks. In most situations, however, only a portion of computers in a small network need access to the Internet simultaneously. This means that the number of allocated addresses does not have to match the number of computers in the network. For example, assume a small business with 20 computers in which the maximum number of computers that access the Internet simultaneously is only 5. Most of the computers are either doing some task that does not need Internet access or communicating with each other. This small business can use the TCP/IP protocol for both internal and universal communication. The business can use 20 (or 25) addresses from the private block addresses discussed before for internal communication; five addresses for universal communication can be assigned by the ISP.

A technology that can provide the mapping between the private and universal addresses, and at the same time, support virtual private networks that we discuss in Chapter 30, is **network address translation (NAT).** The technology allows a site to use a set of private addresses for internal communication and a set of global Internet addresses

(at least one) for communication with the rest of the world. The site must have only one single connection to the global Internet through a NAT-capable router that runs NAT software. Figure 5.39 shows a simple implementation of NAT.

Figure 5.39 *NAT*

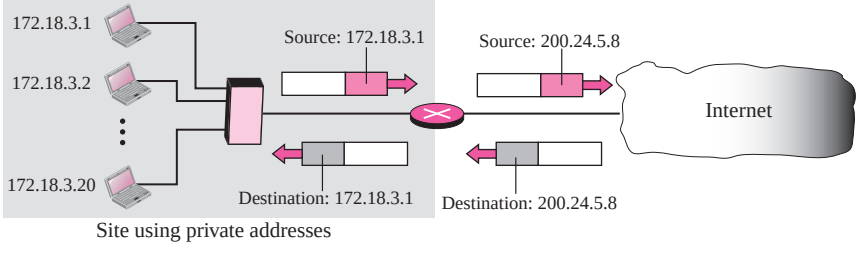


As the figure shows, the private network uses private addresses. The router that connects the network to the global address uses one private address and one global address. The private network is transparent to the rest of the Internet; the rest of the Internet sees only the NAT router with the address 200.24.5.8.

Address Translation

All of the outgoing packets go through the NAT router, which replaces the *source address* in the packet with the global NAT address. All incoming packets also pass through the NAT router, which replaces the *destination address* in the packet (the NAT router global address) with the appropriate private address. Figure 5.40 shows an example of address translation.

Figure 5.40 *Address translation*



Translation Table

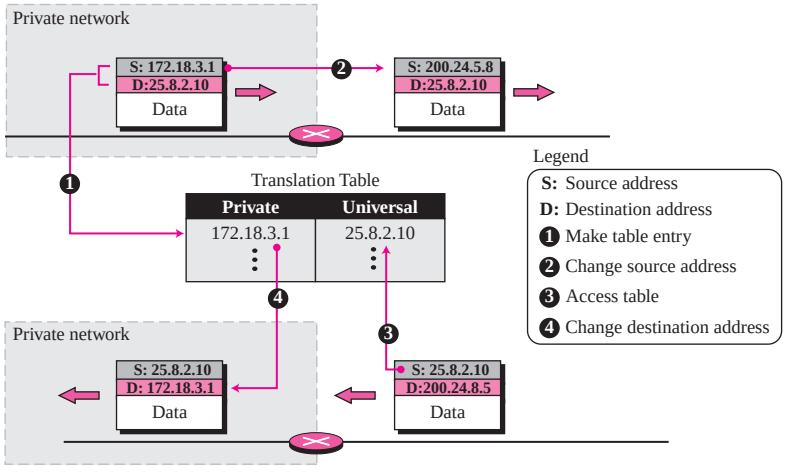
The reader may have noticed that translating the source addresses for an outgoing packet is straightforward. But how does the NAT router know the destination address for a packet coming from the Internet? There may be tens or hundreds of private IP

addresses, each belonging to one specific host. The problem is solved if the NAT router has a **translation table.**

Using One IP Address

In its simplest form, a translation table has only two columns: the private address and the external address (destination address of the packet). When the router translates the source address of the outgoing packet, it also makes note of the destination address—where the packet is going. When the response comes back from the destination, the router uses the source address of the packet (as the external address) to find the private address of the packet. Figure 5.41 shows the idea.

Figure 5.41 *Translation*



In this strategy, communication must always be initiated by the private network. The NAT mechanism described requires that the private network start the communication. As we will see, NAT is used mostly by ISPs that assign one single address to a customer. The customer, however, may be a member of a private network that has many private addresses. In this case, communication with the Internet is always initiated from the customer site, using a client program such as HTTP, TELNET, or FTP to access the corresponding server program. For example, when e-mail that originates from a non-customer site is received by the ISP e-mail server, it is stored in the mailbox of the customer until retrieved with a protocol such as POP.

A private network cannot run a server program for clients outside of its network if it is using NAT technology.

Using a Pool of IP Addresses

Using only one global address by the NAT router allows only one private-network host to access the same external host. To remove this restriction, the NAT router can use a

pool of global addresses. For example, instead of using only one global address (200.24.5.8), the NAT router can use four addresses (200.24.5.8, 200.24.5.9, 200.24.5.10, and 200.24.5.11). In this case, four private-network hosts can communicate with the same external host at the same time because each pair of addresses defines a connection. However, there are still some drawbacks. No more than four connections can be made to the same destination. No private-network host can access two external server programs (e.g., HTTP and TELNET) at the same time. And, likewise, two private-network hosts cannot access the same external server program (e.g., HTTP or TELNET) at the same time.

Using Both IP Addresses and Port Addresses

To allow a many-to-many relationship between private-network hosts and external server programs, we need more information in the translation table. For example, suppose two hosts inside a private network with addresses 172.18.3.1 and 172.18.3.2 need to access the HTTP server on external host 25.8.3.2. If the translation table has five columns, instead of two, that include the source and destination port addresses and the transport layer protocol, the ambiguity is eliminated. Table 5.3 shows an example of such a table.

Table 5.3 *Five-column translation table*

Private Address	*Private Port*	*External Address*	*External Port*	*Transport Protocol*
172.18.3.1	1400	25.8.3.2	80	TCP
172.18.3.2	1401	25.8.3.2	80	TCP
…	…	…	…	…

Note that when the response from HTTP comes back, the combination of source address (25.8.3.2) and destination port address (1400) defines the private network host to which the response should be directed. Note also that for this translation to work, the ephemeral port addresses (1400 and 1401) must be unique.

5.6 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], [Koz 05], [Ste 95].

RFCs

Several RFCs deal with IPv4 addressing including RFC 917, RFC 927, RFC 930, RFC 932, RFC 940, RFC 950, RFC 1122, and RFC 1519.

5.7 KEY TERMS

address aggregation
address space
binary notation
block of addresses
class A address
class B address
class C address
class D address
class E address
classful addressing
classless addressing
classless interdomain routing (CIDR)
default mask
direct broadcast address
dotted-decimal notation
hexadecimal notation
hostid
IP address
limited broadcast address
loopback address
netid
network address
Network Address Translation (NAT)
network mask
prefix
prefix length
slash notation
subnet
subnet mask
subnetting
subnetwork
suffix
suffix length
supernet mask
supernetting
translation table

5.8 SUMMARY

- The identifier used in the IP layer of the TCP/IP protocol suite is called the Internet address or IP address. An IPv4 address is 32 bits long. An address space is the total number of addresses used by the protocol. The address space of IPv4 is 2^{32} or 4,294,967,296.
- In classful addressing, the IPv4 address space is divided into five classes: A, B, C, D, and E. An organization is granted a block in one of the three classes, A, B, or C. Classes D and E is reserved for special purposes. An IP address in classes A, B, and C is divided into netid and hostid.
- In classful addressing, the first address in the block is called the network address. It defines the network to which an address belongs. The network address is used in routing a packet to its destination network.
- A network mask or a default mask in classful addressing is a 32-bit number with n leftmost bits all set to 1s and $(32 - n)$ rightmost bits all set to 0s. It is used by a router to find the network address from the destination address of a packet.
- The idea of splitting a network into smaller subnetworks is called subnetting. A subnetwork mask, like a network mask, is used to find the subnetwork address when a destination IP address is given. In supernetting, an organization can combine several class C blocks to create a larger range of addresses.
- In 1996, the Internet authorities announced a new architecture called classless addressing or CIDR that allows an organization to have a block of addresses of any size as long as the size of the block is a power of two.

- The address in classless addressing is also divided into two parts: the prefix and the suffix. The prefix plays the same role as the netid; the suffix plays the same role as the hostid. All addresses in the block have the same prefix; each address has a different suffix.
- Some of the blocks in IPv4 are reserved for special purposes. In addition, some addresses in a block are traditionally used for special addresses. These addresses are not assigned to any host.
- To improve the distribution of addresses, NAT technology has been created to allow the separation of private addresses in a network from the global addresses used in the Internet. A translation table can translate the private addresses, selected from the blocks allocated for this purpose, to global addresses. The translation table also translates the IP addresses as well as the port number for mapping from the private to global addresses and vice versa.

5.9 PRACTICE SET

Exercises

1. What is the address space in each of the following systems?
   a. a system with 8-bit addresses
   b. a system with 16-bit addresses
   c. a system with 64-bit addresses
2. An address space has a total of 1,024 addresses. How many bits are needed to represent an address?
3. An address space uses three symbols: 0, 1, and 2 to represent addresses. If each address is made of 10 symbols, how many addresses are available in this system?
4. Change the following IP addresses from dotted-decimal notation to binary notation:
   a. 114.34.2.8
   b. 129.14.6.8
   c. 208.34.54.12
   d. 238.34.2.1
5. Change the following IP addresses from dotted-decimal notation to hexadecimal notation:
   a. 114.34.2.8
   b. 129.14.6.8
   c. 208.34.54.12
   d. 238.34.2.1
6. Change the following IP addresses from hexadecimal notation to binary notation:
   a. 0x1347FEAB
   b. 0xAB234102

c. 0x0123A2BE

d. 0x00001111

7. How many hexadecimal digits are needed to define the netid in each of the following classes?

a. Class A

b. Class B

c. Class C

8. Change the following IP addresses from binary notation to dotted-decimal notation:

a. **01111111 11110000 01100111 01111101**

b. **10101111 11000000 11111000 00011101**

c. **11011111 10110000 00011111 01011101**

d. **11101111 11110111 11000111 00011101**

9. Find the class of the following IP addresses:

a. 208.34.54.12

b. 238.34.2.1

c. 242.34.2.8

d. 129.14.6.8

10. Find the class of the following IP addresses:

a. **11110111 11110011 10000111 11011101**

b. **10101111 11000000 11110000 00011101**

c. **11011111 10110000 00011111 01011101**

d. **11101111 11110111 11000111 00011101**

11. Find the netid and the hostid of the following IP addresses:

a. 114.34.2.8

b. 132.56.8.6

c. 208.34.54.12

d. 251.34.98.5

12. Find the number of addresses in the range if the first address is 14.7.24.0 and the last address is 14.14.34.255.

13. If the first address in a range is 122.12.7.0 and there are 2048 addresses in the range, what is the last address?

14. Find the result of each operation:

a. NOT (22.14.70.34)

b. NOT (145.36.12.20)

c. NOT (200.7.2.0)

d. NOT (11.20.255.255)

15. Find the result of each operation:

a. (22.14.70.34) AND (255.255.0.0)

b. (12.11.60.12) AND (255.0.0.0)

c. (14.110.160.12) AND (255.200.140.0)

d. (28.14.40.100) AND (255.128.100.0)

16. Find the result of each operation:

a. (22.14.70.34) OR (255.255.0.0)

b. (12.11.60.12) OR (255.0.0.0)

c. (14.110.160.12) OR (255.200.140.0)

d. (28.14.40.100) OR (255.128.100.0)

17. In a class A subnet, we know the IP address of one of the hosts and the subnet mask as given below:

IP Address: 25.34.12.56	Subnet mask: 255.255.0.0

What is the first address (subnet address)? What is the last address?

18. In a class B subnet, we know the IP address of one of the hosts and the subnet mask as given below:

IP Address: 131.134.112.66	Subnet mask: 255.255.224.0

What is the first address (subnet address)? What is the last address?

19. In a class C subnet, we know the IP address of one of the hosts and the subnet mask as given below:

IP Address: 202.44.82.16	Subnet mask: 255.255.255.192

What is the first address (subnet address)? What is the last address?

20. Find the subnet mask in each case:

a. 1024 subnets in class A

b. 256 subnets in class B

c. 32 subnets in class C

d. 4 subnets in class C

21. In a block of addresses, we know the IP address of one host is 25.34.12.56/16. What is the first address (network address) and the last address (limited broadcast address) in this block?

22. In a block of addresses, we know the IP address of one host is 182.44.82.16/26. What is the first address (network address) and the last address (limited broadcast address) in this block?

23. In fixed-length subnetting, find the number of 1s that must be added to the mask if the number of desired subnets is _____.

a. 2

b. 62

c. 122

d. 250

24. An organization is granted the block 16.0.0.0/8. The administrator wants to create 500 fixed-length subnets.

a. Find the subnet mask.

b. Find the number of addresses in each subnet.

c. Find the first and the last address in the first subnet.

d. Find the first and the last address in the last subnet (subnet 500).

25. An organization is granted the block 130.56.0.0/16. The administrator wants to create 1024 subnets.

a. Find the subnet mask.

b. Find the number of addresses in each subnet.

c. Find the first and the last address in the first subnet.

d. Find the first and the last address in the last subnet (subnet 1024).

26. An organization is granted the block 211.17.180.0/24. The administrator wants to create 32 subnets.

a. Find the subnet mask.

b. Find the number of addresses in each subnet.

c. Find the first and the last address in the first subnet.

d. Find the first and the last address in the last subnet (subnet 32).

27. Write the following mask in slash notation (/*n*):

a. 255.255.255.0

b. 255.0.0.0

c. 255.255.224.0

d. 255.255.240.0

28. Find the range of addresses in the following blocks:

a. 123.56.77.32/29

b. 200.17.21.128/27

c. 17.34.16.0/23

d. 180.34.64.64/30

29. In classless addressing, we know the first and the last address in the block. Can we find the prefix length? If the answer is yes, show the process and give an example.

30. In classless addressing, we know the first address and the number of addresses in the block. Can we find the prefix length? If the answer is yes, show the process and give an example.

31. In classless addressing, can two blocks have the same prefix length? Explain.

32. In classless addressing, we know the first address and one of the addresses in the block (not necessarily the last address). Can we find the prefix length? Explain.

33. An ISP is granted a block of addresses starting with 150.80.0.0/16. The ISP wants to distribute these blocks to 2600 customers as follows:

a. The first group has 200 medium-size businesses; each needs approximately 128 addresses.

b. The second group has 400 small businesses; each needs approximately 16 addresses.

c. The third group has 2000 households; each needs 4 addresses.

Design the subblocks and give the slash notation for each subblock. Find out how many addresses are still available after these allocations.

34. An ISP is granted a block of addresses starting with 120.60.4.0/20. The ISP wants to distribute these blocks to 100 organizations with each organization receiving 8 addresses only. Design the subblocks and give the slash notation for each subblock. Find out how many addresses are still available after these allocations.

35. An ISP has a block of 1024 addresses. It needs to divide the addresses to 1024 customers. Does it need subnetting? Explain your answer.

CHAPTER

6

Delivery and Forwarding of IP Packets

This chapter describes the delivery and forwarding of IP packets. **Delivery** refers to the way a packet is handled by the underlying networks under the control of the network layer. Concepts such as direct and indirect delivery are discussed. **Forwarding** refers to the way a packet is delivered to the next station. We discuss two trends in forwarding: forwarding based on destination address of the packet and forwarding based on the label attached to the packet.

OBJECTIVES

The chapter has several objectives:

- To discuss the delivery of packets in the network layer and distinguish between direct and indirect delivery.
- To discuss the forwarding of packets in the network layer and distinguish between destination-address–based forwarding and label-based forwarding.
- To discuss different forwarding techniques, including next-hop, network-specific, host-specific, and default.
- To discuss the contents of routing tables in classful and classless addressing and some algorithms used to search the tables.
- To introduce MPLS technology and show how it can achieve label-based forwarding.
- To list the components of a router and explain the purpose of each component and their relations to other components.

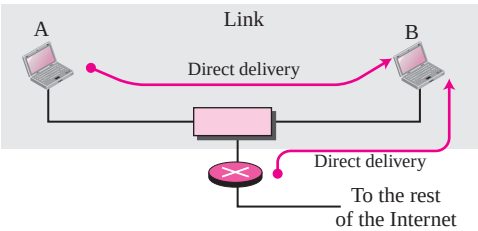
6.1 DELIVERY

The network layer supervises the handling of the packets by the underlying physical networks. We define this handling as the delivery of a packet. The delivery of a packet to its final destination is accomplished using two different methods of delivery: direct and indirect.

Direct Delivery

In a **direct delivery,** the final destination of the packet is a host connected to the same physical network as the deliverer. Direct delivery occurs when the source and destination of the packet are located on the same physical network or if the delivery is between the last router and the destination host (see Figure 6.1).

Figure 6.1 *Direct delivery*



The sender can easily determine if the delivery is direct. It can extract the network address of the destination (using the mask) and compare this address with the addresses of the networks to which it is connected. If a match is found, the delivery is direct.

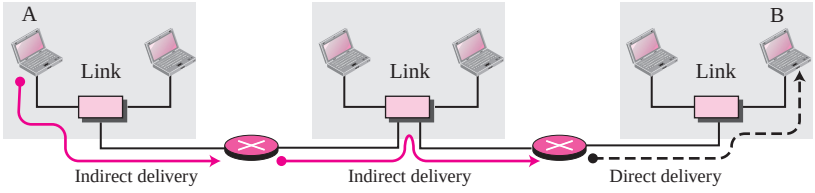
In direct delivery, the sender uses the destination IP address to find the destination physical address. The IP software then gives the destination IP address with the destination physical address to the data link layer for actual delivery. This process is called *mapping the IP address to the physical address*. Although this mapping can be done by finding a match in a table, we will see in Chapter 8 that a protocol called Address Resolution Protocol (ARP) dynamically maps an IP address to the corresponding physical address.

Indirect Delivery

If the destination host is not on the same network as the deliverer, the packet is delivered indirectly. In an **indirect delivery,** the packet goes from router to router until it

reaches the one connected to the same physical network as its final destination. Figure 6.2 shows the concept of indirect delivery.

Figure 6.2 *Indirect delivery*



In an indirect delivery, the sender uses the destination IP address and a routing table to find the IP address of the next router to which the packet should be delivered. The sender then uses ARP (see Chapter 8) to find the physical address of the next router. Note that in direct delivery, the address mapping is between the IP address of the final destination and the physical address of the final destination. In an indirect delivery, the address mapping is between the IP address of the next router and the physical address of the next router. Note that a delivery always involves one direct delivery but zero or more indirect deliveries. Note also that the last delivery is always a direct delivery.

6.2 FORWARDING

Forwarding means to place the packet in its route to its destination. Since the Internet today is made of a combination of links (networks), forwarding means to deliver the packet to the next hop (which can be the final destination or the intermediate connecting device). Although the IP protocol was originally designed as a connectionless protocol, today the tendency is to use IP as a connection-oriented protocol.

When IP is used as a connectionless protocol, forwarding is based on the destination address of the IP datagram; when the IP is used as a connection-oriented protocol, forwarding is based on the label attached to an IP datagram. We first discuss forwarding based on the destination address and then forwarding based on the label.

Forwarding Based on Destination Address

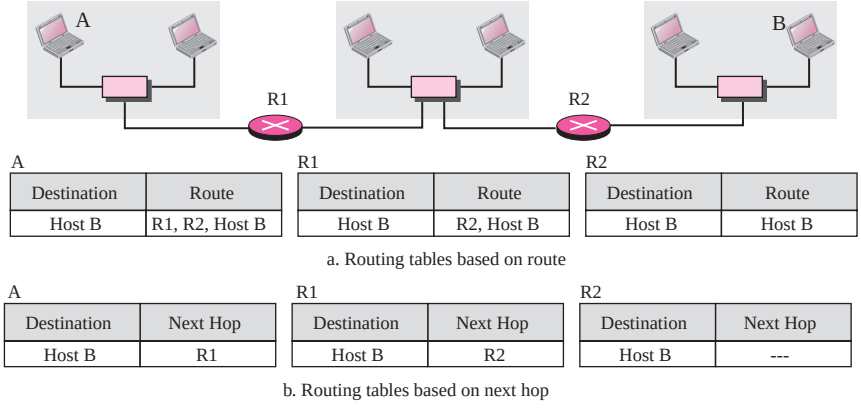
We first discuss forwarding based on the destination address. This is a traditional approach, which is prevalent today. In this case, forwarding requires a host or a router to have a routing table. When a host has a packet to send or when a router has received a packet to be forwarded, it looks at this table to find the route to the final destination. However, this simple solution is inefficient today in an internetwork such as the Internet because the number of entries needed in the routing table would make table lookups inefficient.

Forwarding Techniques

Several techniques can make the size of the routing table manageable and also handle issues such as security. We briefly discuss these methods here.

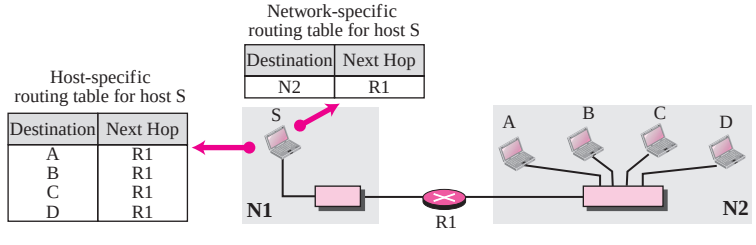
Next-Hop Method One technique to reduce the contents of a routing table is called the **next-hop method.** In this technique, the routing table holds only the address of the next hop instead of information about the complete route. The entries of a routing table must be consistent with each other. Figure 6.3 shows how routing tables can be simplified using this technique.

Figure 6.3 *Next-hop method*



Network-Specific Method A second technique to reduce the routing table and simplify the searching process is called the **network-specific method.** Here, instead of having an entry for every destination host connected to the same physical network, we have only one entry that defines the address of the destination network itself. In other words, we treat all hosts connected to the same network as one single entity. For example, if 1000 hosts are attached to the same network, only one entry exists in the routing table instead of 1000. Figure 6.4 shows the concept.

Figure 6.4 *Network-specific method*

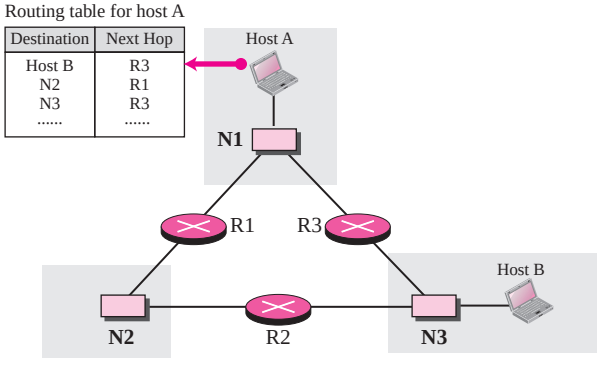


Host-Specific Method In the **host-specific method,** the destination host address is given in the routing table. The rationale behind this method is the inverse of the network-specific method. Here efficiency is sacrificed for other advantages: Although it is not efficient to put the host address in the routing table, there are occasions in

which the administrator wants to have more control over routing. For example, in Figure 6.5 if the administrator wants all packets arriving for host B delivered to router R3 instead of R1, one single entry in the routing table of host A can explicitly define the route.

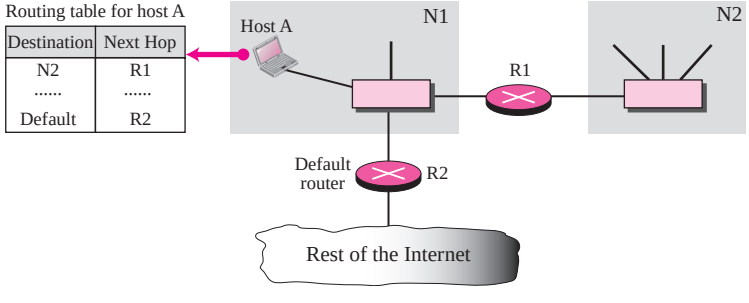
Host-specific routing is used for purposes such as checking the route or providing security measures.

Figure 6.5 *Host-specific routing*



Default Method Another technique to simplify routing is called the **default method.** In Figure 6.6 host A is connected to a network with two routers. Router R1 routes the packets to hosts connected to network N2. However, for the rest of the Internet, router R2 is used. So instead of listing all networks in the entire Internet, host A can just have one entry called the *default* (normally defined as network address 0.0.0.0).

Figure 6.6 *Default routing*



Forwarding with Classful Addressing

As we mentioned in the previous chapter, classful addressing has several disadvantages. However, the existence of a default mask in a classful address makes the forwarding process simple. In this section, we first show the contents of a routing table

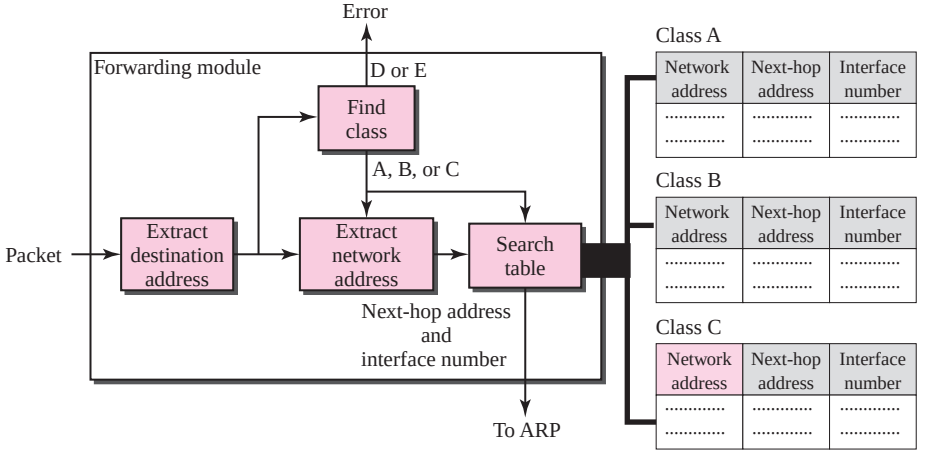
and forwarding module for the situation in which there is no subnetting. We then show how the module changes if subnetting is involved.

Forwarding without Subnetting In classful addressing, most of the routers in the global Internet are not involved in subnetting. Subnetting happens inside the organization. A typical forwarding module in this case can be designed using three tables, one for each unicast class (A, B, C). If the router supports multicasting, another table can be added to handle class D addresses. Having three different tables makes searching more efficient. Each routing table has a minimum of three columns:

1. The network address of the destination network tells us where the destination host is located. Note that we use network-specific forwarding and not the rarely used host-specific forwarding.
2. The next-hop address tells us to which router the packet must be delivered for an indirect delivery. This column is empty for a direct delivery.
3. The interface number defines the outgoing port from which the packet is sent out. A router is normally connected to several networks. Each connection has a different numbered port or interface. We show them as m0, m1, and so on.

Figure 6.7 shows a simplified module.

Figure 6.7 *Simplified forwarding module in classful address without subnetting*



In its simplest form, the forwarding module follows these steps:

1. The destination address of the packet is extracted.
2. A copy of the destination address is used to find the class of the address. This is done by shifting the copy of the address 28 bits to the right. The result is a 4-bit number between 0 and 15. If the result is
   a. 0 to 7, the class is A.
   b. 8 to 11, the class is B.
   c. 12 or 13, the class is C

d. 14, the class is D.

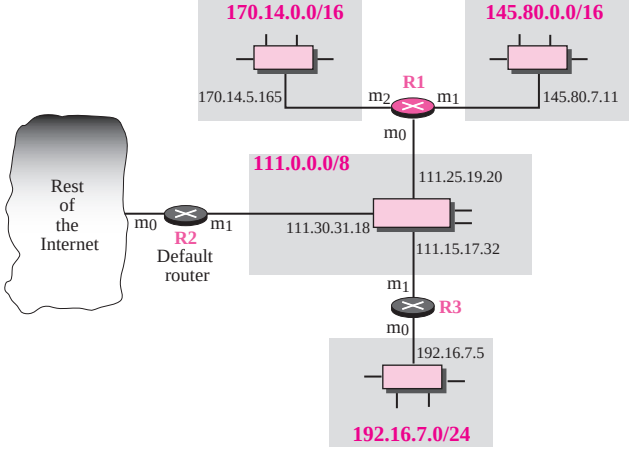
e. 15, the class is E.

3. The result of Step 2 for class A, B, or C and the destination address are used to extract the network address. This is done by masking off (changing to 0s) the rightmost 8, 16, or 24 bits based on the class.
4. The class of the address and the network address are used to find next-hop information. The class determines which table is to be searched. The module searches this table for the network address. If a match is found, the **next-hop address** and the interface number of the output port are extracted from the table. If no match is found, the default is used.
5. The ARP module (Chapter 8) uses the next-hop address and the interface number to find the physical address of the next router. It then asks the data link layer to deliver the packet to the next hop.

Example 6.1

Figure 6.8 shows an imaginary part of the Internet. Show the routing tables for router R1.

Figure 6.8 *Configuration for routing, Example 1*



Solution

Figure 6.9 shows the three tables used by router R1. Note that some entries in the next-hop address column are empty because in these cases, the destination is in the same network to which the router is connected (direct delivery). In these cases, the next-hop address used by ARP is simply the destination address of the packet as we will see in Chapter 8.

Example 6.2

Router R1 in Figure 6.8 receives a packet with destination address 192.16.7.14. Show how the packet is forwarded.

Figure 6.9 *Tables for Example 6.1*

Class A

Network address	Next-hop address	Interface
111.0.0.0	----------	m0

Class B

Network address	Next-hop address	Interface
145.80.0.0	----------	m1
170.14.0.0	----------	m2

Class C

Network address	Next-hop address	Interface
192.16.7.0	111.15.17.32	m0

Default: 111.30.31.18, m0

Solution

The destination address in binary is 11000000 00010000 00000111 00001110. A copy of the address is shifted 28 bits to the right. The result is 00000000 00000000 00000000 0000**1100** or 12. The destination network is class C. The network address is extracted by masking off the leftmost 24 bits of the destination address; the result is 192.16.7.0. The table for Class C is searched. The network address is found in the first row. The next-hop address 111.15.17.32. and the interface m0 are passed to ARP (see Chapter 8).

Example 6.3

Router R1 in Figure 6.8 receives a packet with destination address 167.24.160.5. Show how the packet is forwarded.

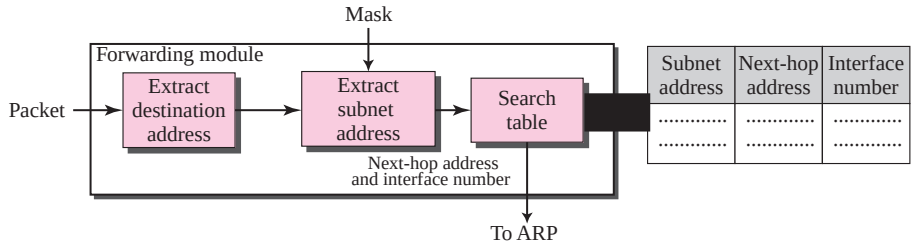
Solution

The destination address in binary is 10100111 00011000 10100000 00000101. A copy of the address is shifted 28 bits to the right. The result is 00000000 00000000 00000000 0000**1010** or 10. The class is B. The network address can be found by masking off 16 bits of the destination address, the result is 167.24.0.0. The table for Class B is searched. No matching network address is found. The packet needs to be forwarded to the default router (the network is somewhere else in the Internet). The next-hop address 111.30.31.18 and the interface number m0 are passed to ARP.

Forwarding with Subnetting In classful addressing, subnetting happens inside the organization. The routers that handle subnetting are either at the border of the organization site or inside the site boundary. If the organization is using variable-length subnetting, we need several tables; otherwise, we need only one table. Figure 6.10 shows a simplified module for fixed-length subnetting.

1. The module extracts the destination address of the packet.
2. If the destination address matches any of the host-specific addresses in the table, the next-hop and the interface number is extracted from the table.
3. The destination address and the mask are used to extract the subnet address.
4. The table is searched using the subnet address to find the next-hop address and the interface number. If no match is found, the default is used.
5. The next-hop address and the interface number are given to ARP (see Chapter 8).

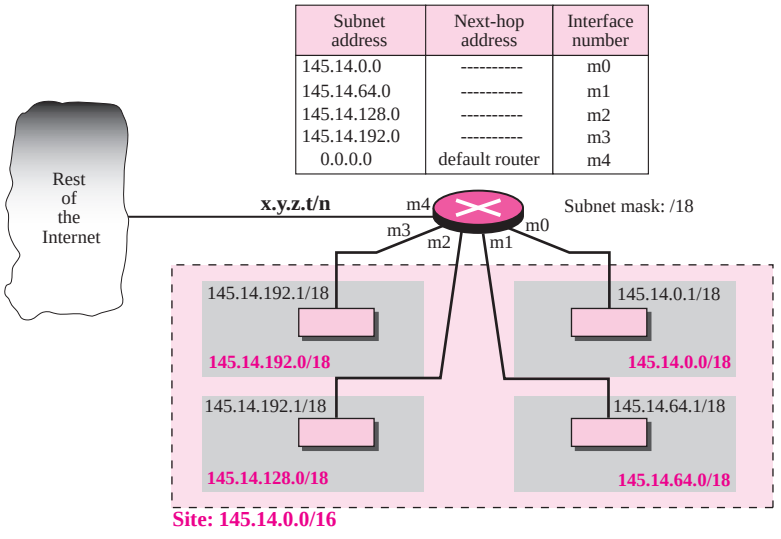
Figure 6.10 *Simplified forwarding module in classful address with subnetting*



Example 6.4

Figure 6.11 shows a router connected to four subnets.

Figure 6.11 *Configuration for Example 6.4*



Note several points. First, the site address is 145.14.0.0/16 (a class B address). Every packet with destination address in the range 145.14.0.0 to 145.14.255.255 is delivered to the interface m4 and distributed to the final destination subnet by the router. Second, we have used the address x.y.z.t/*n* for the interface m4 because we do not know to which network this router is connected. Third, the table has a default entry for packets that are to be sent out of the site. The router is configured to apply the subnet mask /18 to any destination address.

Example 6.5

The router in Figure 6.11 receives a packet with destination address 145.14.32.78. Show how the packet is forwarded.

Solution

The mask is /18. After applying the mask, the subnet address is 145.14.0.0. The packet is delivered to ARP (see Chapter 8) with the next-hop address 145.14.32.78 and the outgoing interface m0.

Example 6.6

A host in network 145.14.0.0 in Figure 6.11 has a packet to send to the host with address 7.22.67.91. Show how the packet is routed.

Solution

The router receives the packet and applies the mask (/18). The network address is 7.22.64.0. The table is searched and the address is not found. The router uses the address of the default router (not shown in figure) and sends the packet to that router.

Forwarding with Classless Addressing

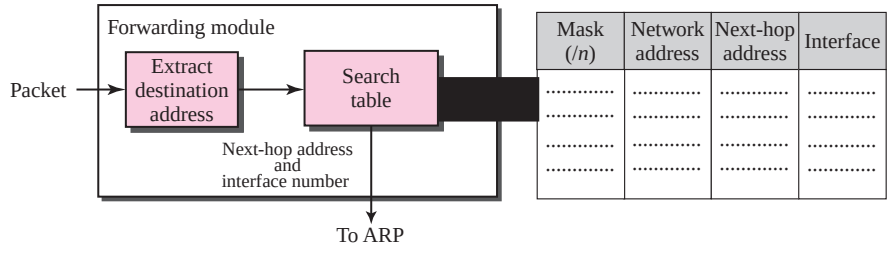
In classless addressing, the whole address space is one entity; there are no classes. This means that forwarding requires one row of information for each block involved. The table needs to be searched based on the network address (first address in the block). Unfortunately, the destination address in the packet gives no clue about the network address (as it does in classful addressing).

To solve the problem, we need to include the mask (/*n*) in the table; we need to have an extra column that includes the mask for the corresponding block. In other words, although a classful routing table can be designed with three columns, a classless routing table needs at least four columns.

In classful addressing we can have a routing table with three columns; in classless addressing, we need at least four columns.

Figure 6.12 shows a simple forwarding module for classless addressing. Note that network address extraction is done at the same time as table searching because there is no inherent information in the destination address that can be used for network address extraction.

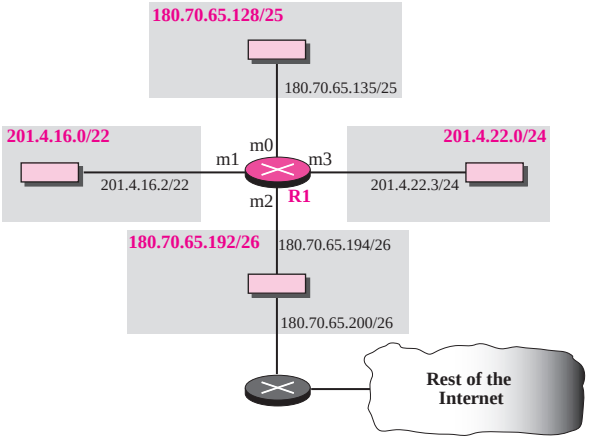
Figure 6.12 *Simplified forwarding module in classless address*



Example 6.7

Make a routing table for router R1 using the configuration in Figure 6.13.

Figure 6.13 *Configuration for Example 6.7*



Solution

Table 6.1 shows the corresponding table.

Table 6.1 *Routing table for router R1 in Figure 6.13*

Mask	*Network Address*	*Next Hop*	*Interface*
/26	180.70.65.192	-	m2
/25	180.70.65.128	-	m0
/24	201.4.22.0	-	m3
/22	201.4.16.0	....	m1
Default	Default	180.70.65.200	m2

Example 6.8

Show the forwarding process if a packet arrives at R1 in Figure 6.13 with the destination address 180.70.65.140.

Solution

The router performs the following steps:

1. The first mask (/26) is applied to the destination address. The result is 180.70.65.128, which does not match the corresponding network address.
2. The second mask (/25) is applied to the destination address. The result is 180.70.65.128, which matches the corresponding network address. The next-hop address (the destination address of the packet in this case) and the interface number m0 are passed to ARP (see Chapter 8) for further processing.

Example 6.9

Show the forwarding process if a packet arrives at R1 in Figure 6.13 with the destination address 201.4.22.35.

Solution

The router performs the following steps:

1. The first mask (/26) is applied to the destination address. The result is 201.4.22.0, which does not match the corresponding network address (row 1).
2. The second mask (/25) is applied to the destination address. The result is 201.4.22.0, which does not match the corresponding network address (row 2).
3. The third mask (/24) is applied to the destination address. The result is 201.4.22.0, which matches the corresponding network address. The destination address of the packet and the interface number m3 are passed to ARP (see Chapter 8).

Example 6.10

Show the forwarding process if a packet arrives at R1 in Figure 6.13 with the destination address 18.24.32.78.

Solution

This time all masks are applied to the destination address, but no matching network address is found. When it reaches the end of the table, the module gives the next-hop address 180.70.65.200 and interface number m2 to ARP (see Chapter 8). This is probably an outgoing package that needs to be sent, via the default router, to someplace else in the Internet.

Example 6.11

Now let us give a different type of example. Can we find the configuration of a router if we know only its routing table? The routing table for router R1 is given in Table 6.2. Can we draw its topology?

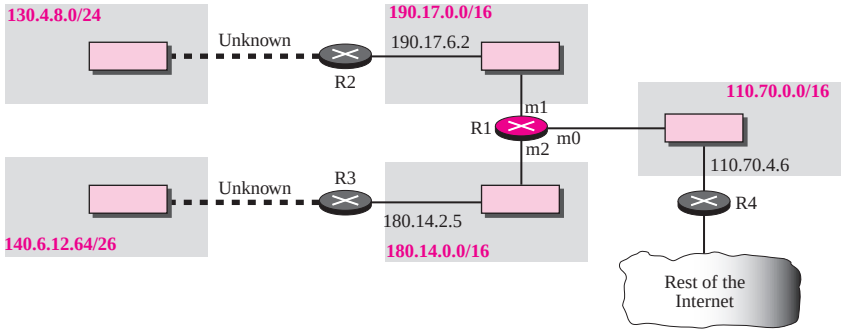
Table 6.2 *Routing table for Example 6.11*

Mask	*Network Address*	*Next-Hop Address*	*Interface Number*
/26	140.6.12.64	180.14.2.5	m2
/24	130.4.8.0	190.17.6.2	m1
/16	110.70.0.0	----------	m0
/16	180.14.0.0	----------	m2
/16	190.17.0.0	----------	m1
Default	Default	110.70.4.6	m0

Solution

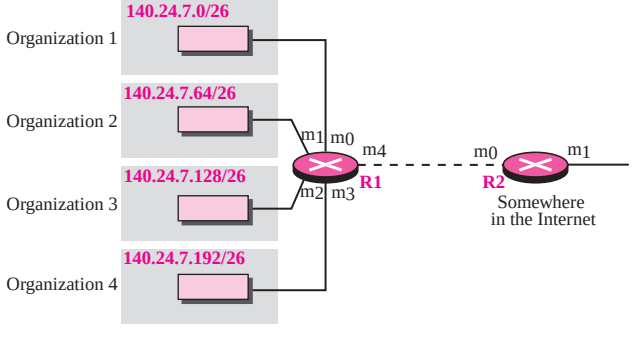
We know some facts but we don't have all for a definite topology. We know that router R1 has three interfaces: m0, m1, and m2. We know that there are three networks directly connected to router R1. We know that there are two networks indirectly connected to R1. There must be at least three other routers involved (see next-hop column). We know to which networks these routers are connected by looking at their IP addresses. So we can put them at their appropriate place. We know that one router, the default router, is connected to the rest of the Internet. But there is some missing information.We do not know if network 130.4.8.0 is directly connected to router R2 or through a point-to-point network (WAN) and another router. We do not know if network 140.6.12.64 is connected to router R3 directly or through a point-to-point network (WAN) and another router. Point-to-point networks normally do not have an entry in the routing table because no hosts are connected to them. Figure 6.14 shows our guessed topology.

Figure 6.14 *Guessed topology for Example 6.11*



Address Aggregation When we use classful addressing, there is only one entry in the routing table for each site outside the organization. The entry defines the site even if that site is subnetted. When a packet arrives at the router, the router checks the corresponding entry and forwards the packet accordingly. When we use classless addressing, it is likely that the number of routing table entries will increase. This is because the intent of classless addressing is to divide up the whole address space into manageable blocks. The increased size of the table results in an increase in the amount of time needed to search the table. To alleviate the problem, the idea of **address aggregation** was designed. In Figure 6.15 we have two routers.

Figure 6.15 *Address aggregation*



Mask	Network address	Next-hop address	Interface
/26	140.24.7.0	----------	m0
/26	140.24.7.64	----------	m1
/26	140.24.7.128	----------	m2
/26	140.24.7.192	----------	m3
/0	0.0.0.0	default router	m4

Routing table for R1

Mask	Network address	Next-hop address	Interface
/24	140.24.7.0	----------	m0
/0	0.0.0.0	default router	m1

Routing table for R2

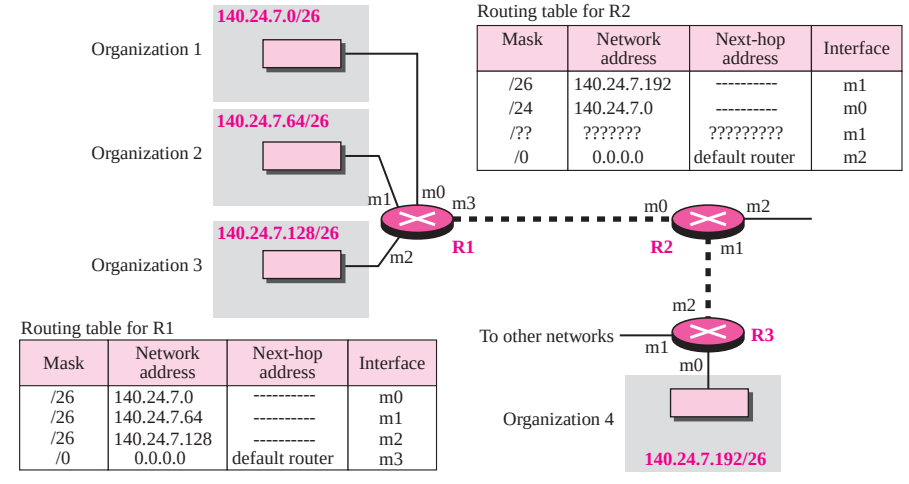
R1 is connected to networks of four organizations that each use 64 addresses. R2 is somewhere far from R1. R1 has a longer routing table because each packet must be correctly routed to the appropriate organization. R2, on the other hand, can have a very

small routing table. For R2, any packet with destination 140.24.7.0 to 140.24.7.255 is sent out from interface m0 regardless of the organization number. This is called address aggregation because the blocks of addresses for four organizations are aggregated into one larger block. R2 would have a longer routing table if each organization had addresses that could not be aggregated into one block.

Note that although the idea of address aggregation is similar to the idea of subnetting, we do not have a common site here; the network for each organization is independent. In addition, we can have several levels of aggregation.

Longest Mask Matching What happens if one of the organizations in the previous figure is not geographically close to the other three? For example, if organization 4 cannot be connected to router R1 for some reason, can we still use the idea of address aggregation and still assign block 140.24.7.192/26 to organization 4? The answer is yes because routing in classless addressing uses another principle, **longest mask matching.** This principle states that the routing table is sorted from the longest mask to the shortest mask. In other words, if there are three masks, /27, /26, and /24, the mask /27 must be the first entry and /24 must be last. Let us see if this principle solves the situation in which organization 4 is separated from the other three organizations. Figure 6.16 shows the situation.

Figure 6.16 *Longest mask matching*



Routing table for R2

Mask	Network address	Next-hop address	Interface
/26	140.24.7.192	----------	m1
/24	140.24.7.0	----------	m0
/??	???????	?????????	m1
/0	0.0.0.0	default router	m2

Routing table for R1

Mask	Network address	Next-hop address	Interface
/26	140.24.7.0	----------	m0
/26	140.24.7.64	----------	m1
/26	140.24.7.128	----------	m2
/0	0.0.0.0	default router	m3

Routing table for R3

Mask	Network address	Next-hop address	Interface
/26	140.24.7.192	----------	m0
/??	???????	?????????	m1
/0	0.0.0.0	default router	m2

Suppose a packet arrives for organization 4 with destination address 140.24.7.200. The first mask at router R2 is applied, which gives the network address 140.24.7.192. The packet is routed correctly from interface m1 and reaches organization 4. If, however, the routing table was not stored with the longest prefix first, applying the /24 mask would result in the incorrect routing of the packet to router R1.

Hierarchical Routing To solve the problem of gigantic routing tables, we can create a sense of hierarchy in the routing tables. In Chapter 1, we mentioned that the Internet today has a sense of hierarchy. We said that the Internet is divided into backbone, regional and local ISPs. If the routing table has a sense of hierarchy like the Internet architecture, the routing table can decrease in size.

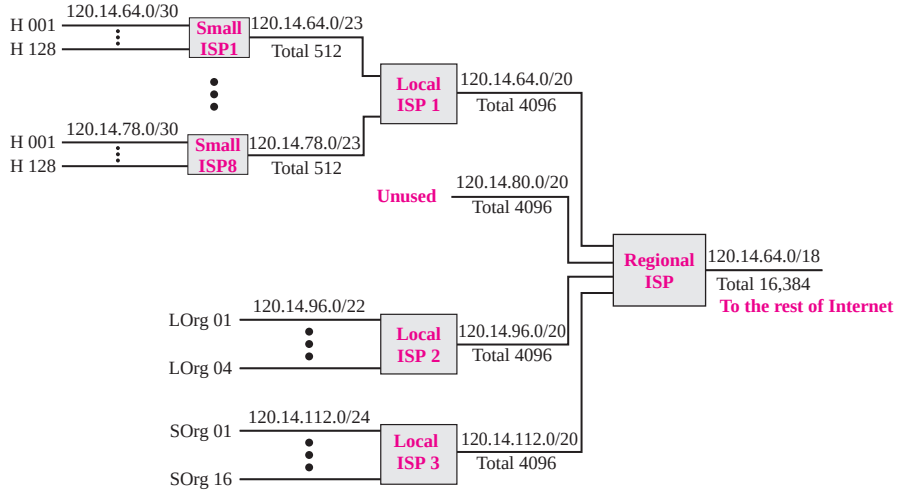
Let us take the case of a local ISP. A local ISP can be assigned a single, but large, block of addresses with a certain prefix length. The local ISP can divide this block into smaller blocks of different sizes, and assign these to individual users and organizations, both large and small. If the block assigned to the local ISP starts with a.b.c.d/*n*, the ISP can create blocks starting with e.f.g.h/*m*, where *m* may vary for each customer and is greater than *n*.

How does this reduce the size of the routing table? The rest of the Internet does not have to be aware of this division. All customers of the local ISP are defined as a.b.c.d/*n* to the rest of the Internet. Every packet destined for one of the addresses in this large block is routed to the local ISP. There is only one entry in every router in the world for all of these customers. They all belong to the same group. Of course, inside the local ISP, the router must recognize the subblocks and route the packet to the destined customer. If one of the customers is a large organization, it also can create another level of hierarchy by subnetting and dividing its subblock into smaller subblocks (or sub-subblocks). In classless routing, the levels of hierarchy are unlimited so long as we follow the rules of classless addressing.

Example 6.12

As an example of hierarchical routing, let us consider Figure 6.17. A regional ISP is granted 16,384 addresses starting from 120.14.64.0. The regional ISP has decided to divide this block into 4 subblocks, each with 4096 addresses. Three of these subblocks are assigned to three local

Figure 6.17 *Hierarchical routing with ISPs*



ISPs, the second subblock is reserved for future use. Note that the mask for each block is /20 because the original block with mask /18 is divided into 4 blocks.

The first local ISP has divided its assigned subblock into 8 smaller blocks and assigned each to a small ISP. Each small ISP provides services to 128 households (H001 to H128), each using four addresses. Note that the mask for each small ISP is now /23 because the block is further divided into 8 blocks. Each household has a mask of /30, because a household has only 4 addresses ($2^{32-30} = 4$). The second local ISP has divided its block into 4 blocks and has assigned the addresses to 4 large organizations (LOrg01 to LOrg04). Note that each large organization has 1024 addresses and the mask is /22.

The third local ISP has divided its block into 16 blocks and assigned each block to a small organization (SOrg01 to SOrg15). Each small organization has 256 addresses and the mask is /24. There is a sense of hierarchy in this configuration. All routers in the Internet send a packet with destination address 120.14.64.0 to 120.14.127.255 to the regional ISP. The regional ISP sends every packet with destination address 120.14.64.0 to 120.14.79.255 to Local ISP1. Local ISP1 sends every packet with destination address 120.14.64.0 to 120.14.64.3 to H001.

Geographical Routing To decrease the size of the routing table even further, we need to extend hierarchical routing to include geographical routing. We must divide the entire address space into a few large blocks. We assign a block to America, a block to Europe, a block to Asia, a block to Africa, and so on. The routers of ISPs outside of Europe will have only one entry for packets to Europe in their routing tables. The routers of ISPs outside of America will have only one entry for packets to North America in their routing tables. And so on.

Routing Table Search Algorithms

The algorithms in classful addressing that search the routing tables must be changed to make classless routing more efficient. This includes the algorithms that update routing tables. We will discuss this updating issue in Chapter 11.

Searching in Classful Addressing In classful addressing, the routing table is organized as a list. However, to make searching more efficient, the routing table can be divided into three tables (sometimes called buckets), one for each class. When the packet arrives, the router applies the default mask (which is inherent in the address itself) to find the corresponding bucket (A, B, or C). The router then searches the corresponding bucket instead of the whole table. Some routers even assign 8 buckets for class A, 4 buckets for class B, and 2 buckets for class C based on the outcome of the class finding process.

Searching in Classless Addressing In classless addressing, there is no network information in the destination address. The simplest, but not the most efficient, search method is called the **longest prefix match** (as we discussed before**)**. The routing table can be divided into buckets, one for each prefix. The router first tries the longest prefix. If the destination address is found in this bucket, the search is complete. If the address is not found, the next prefix is searched. And so on. It is obvious that this type of search takes a long time.

One solution is to change the data structure used for searching. Instead of a list, other data structures (such as a tree or a binary tree) can be used. One candidate is a trie (a special kind of tree). However, this discussion is beyond the scope of this book.

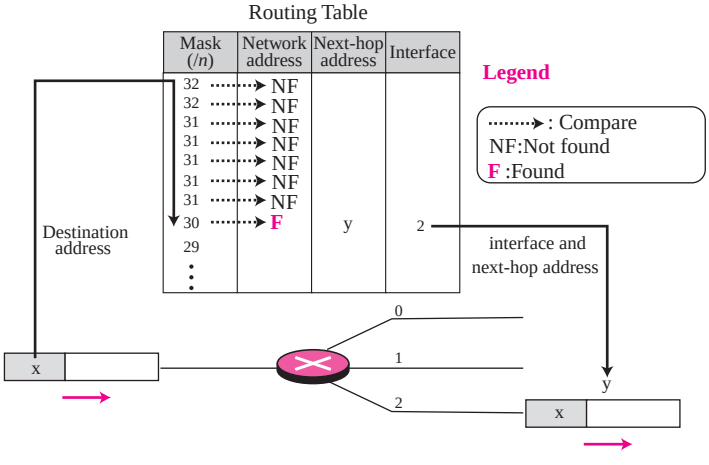
Forwarding Based on Label

In 1980s, an effort started to somehow change IP to behave like a connection-oriented protocol in which the routing is replaced by switching. As we discussed in Chapter 4, in a connectionless network (datagram approach), a router forwards a packet based on the destination address in the header of packet. On the other hand, in a connection-oriented network (virtual-circuit approach), a switch forwards a packet based on the label *attached* to a packet. Routing is normally based on searching the contents of a table; switching can be done by accessing a table using an index. In other words, routing involves searching; switching involves accessing.

Example 6.13

Figure 6.18 shows a simple example of searching in a routing table using the longest match algorithm. Although there are some more efficient algorithms today, the principle is the same.

Figure 6.18 *Example 6.13: Forwarding based on destination address*



When the forwarding algorithm gets the destination address of the packet, it needs to delve into the mask column. For each entry, it needs to apply the mask to find the destination network address. It then needs to check the network addresses in the table until it finds the match. The router then extracts the next-hop address and the interface number to be delivered to the ARP protocol for delivery of the packet to the next hop.

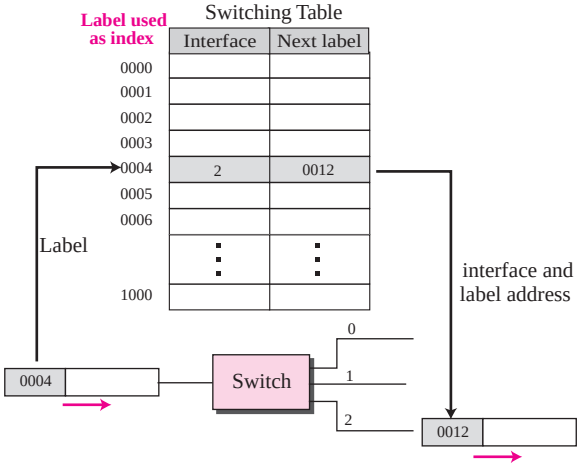
Example 6.14

Figure 6.19 shows a simple example of using a label to access a switching table. Since the labels are used as the index to the table, finding the information in the table is immediate.

MPLS

During the 1980s, several vendors created routers that implement switching technology. Later IETF approved a standard that is called Multi-Protocol Label Switching. In

Figure 6.19 *Example 6.14: Forwarding based on label*

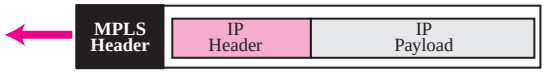


this standard, some conventional routers in the Internet can be replaced by MPLS routers that can behave like a router and a switch. When behaving like a router, MPLS can forward the packet based on the destination address; when behaving like a switch, it can forward a packet based on the label.

A New Header

To simulate connection-oriented switching using a protocol like IP, the first thing that is needed is to add a field to the packet that carry the label discussed in Chapter 4. The IPv4 packet format does not allow this extension (although this field is provided in IPv6 packet format, as we will see in Chapter 27). The solution is to encapsulate the IPv4 packet in an MPLS packet (as though MPLS is a layer between the data link layer and the network layer). The whole IP packet is encapsulated as the payload in an MPLS packet and MPLS header is added. Figure 6.20 shows the encapsulation.

Figure 6.20 *MPLS header added to an IP packet*

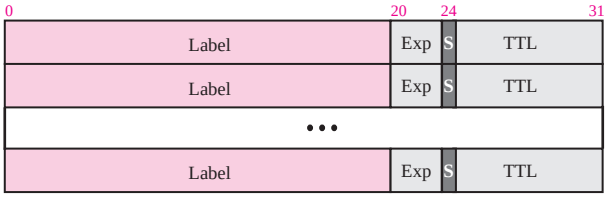


The MPLS header is actually a stack of subheaders that is used for multilevel hierarchical switching as we discuss shortly. Figure 6.21 shows the format of an MPLS header in which each subheader is 32 bits (4 bytes) long.

The following is a brief description of each field:

- **Label.** This 20-bit field defines the label that is used to index the routing table in the router.
- **Exp.** This 3-bit field is reserved for experimental purposes.

Figure 6.21 *MPLS header made of stack of labels*



- **S.** The one-bit stack field defines the situation of the subheader in the stack. When the bit is 1, it means that the header is the last one in the stack.
- **TTL.** This 8-bit field is similar to the TTL field in the IP datagram (see Chapter 7). Each visited router decrement the value of this field. When it reaches zero, the packet is discarded to prevent looping.

Hierarchical Switching

A stack of labels in MPLS allows hierarchical switching. This is similar to conventional hierarchical routing. For example, a packet with two labels can use the top label to forward the packet through switches outside an organization; the bottom label can be used to route the packet inside the organization to reach the destination subnet.

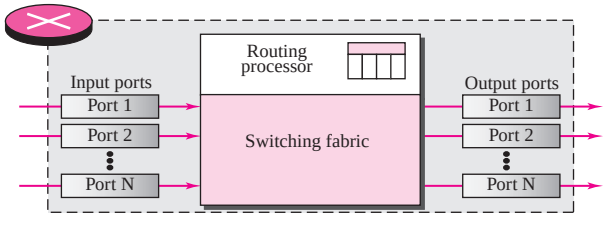
6.3 STRUCTURE OF A ROUTER

In our discussion of forwarding and routing, we represented a router as a black box that accepts incoming packets from one of the input ports (interfaces), uses a routing table to find the output port from which the packet departs, and sends the packet from this output port. In this section we open the black box and look inside. However, our discussion won't be very detailed; entire books have been written about routers. We just give an overview to the reader.

Components

We can say that a router has four components: **input ports, output ports,** the **routing processor,** and the **switching fabric,** as shown in Figure 6.22.

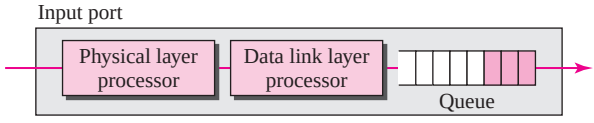
Figure 6.22 *Router components*



Input Ports

Figure 6.23 shows a schematic diagram of an input port.

Figure 6.23 *Input port*

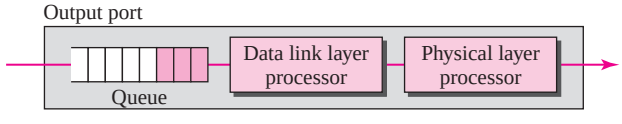


An input port performs the physical and data link layer functions of the router. The bits are constructed from the received signal. The packet is decapsulated from the frame. Errors are detected and corrected. The packet is ready to be forwarded by the network layer. In addition to a physical layer processor and a data link processor, the input port has buffers (queues) to hold the packets before they are directed to the switching fabric.

Output Ports

An output port performs the same functions as the input port, but in the reverse order. First the outgoing packets are queued, then the packet is encapsulated in a frame, and finally the physical layer functions are applied to the frame to create the signal to be sent on the line. Figure 6.24 shows a schematic diagram of an output port.

Figure 6.24 *Output port*



Routing Processor

The routing processor performs the functions of the network layer. The destination address is used to find the address of the next hop and, at the same time, the output port number from which the packet is sent out. This activity is sometimes referred to as *table lookup* because the routing processor searches the routing table. In the newer routers, this function of the routing processor is being moved to the input ports to facilitate and expedite the process.

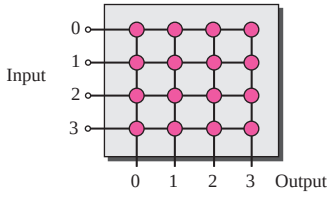
Switching Fabrics

The most difficult task in a router is to move the packet from the input queue to the output queue. The speed with which this is done affects the size of the input/output queue and the overall delay in packet delivery. In the past, when a router was actually a dedicated computer, the memory of the computer or a bus was used as the switching fabric. The input port stored the packet in memory; the output port got the packet from the

memory. Today, routers use a variety of switching fabrics. We briefly discuss some of these fabrics here.

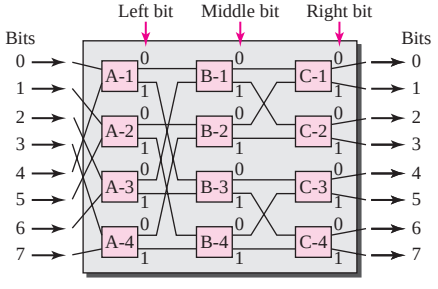
Crossbar Switch The simplest type of switching fabric is the crossbar switch shown in Figure 6.25. A **crossbar switch** connects *n* inputs to *n* outputs in a grid, using electronic microswitches at each **crosspoint.**

Figure 6.25 *Crossbar switch*



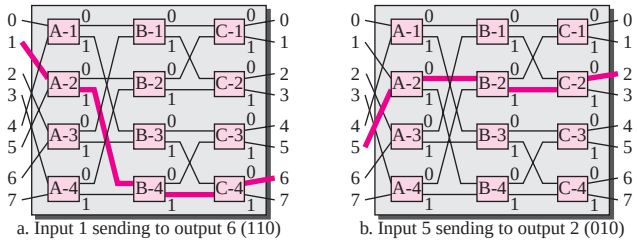
Banyan Switch More realistic than the crossbar switch is the **banyan switch** (named after the banyan tree) as shown in Figure 6.26.

Figure 6.26 *A banyan switch*



A banyan switch is a multistage switch with microswitches at each stage that route the packets based on the output port represented as a binary string. For *n* inputs and *n* outputs, we have $\log_2(n)$ stages with $n/2$ microswitches at each stage. The first stage routes the packet based on the highest order bit of the binary string. The second stage routes the packets based on the second highest order bit, and so on. Figure 6.27 shows a banyan switch with eight inputs and eight outputs. The number of stages is $\log_2(8) = 3$. Suppose a packet has arrived at input port 1 and must go to output port 6 (110 in binary). The first microswitch (A-2) routes the packet based on the first bit (1), the second microswitch (B-4) routes the packet based on the second bit (1), and the third microswitch (C-4) routes the packet based on the third bit (0). In part b, a packet has arrived at input port 5 and must go to output port 2 (010 in binary). The first microswitch (A-2) routes the packet based on the first bit (0), the second microswitch (B-2) routes the packet based on the second bit (1), and the third microswitch (C-2) routes the packet based on the third bit (0).

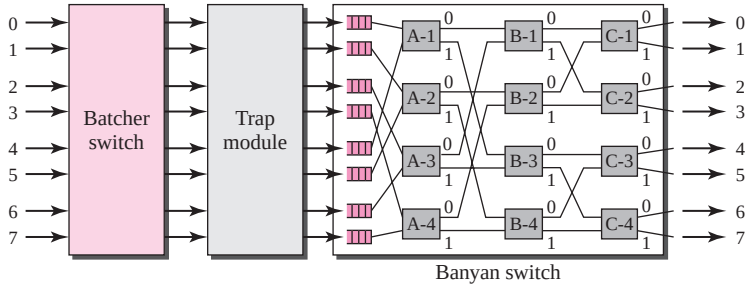
Figure 6.27 *Examples of routing in a banyan switch*



a. Input 1 sending to output 6 (110) b. Input 5 sending to output 2 (010)

Batcher-Banyan Switch The problem with the banyan switch is the possibility of internal collision even when two packets are not heading for the same output port. We can solve this problem by sorting the arriving packets based on their destination port. K. E. Batcher designed a switch that comes before the banyan switch and sorts the incoming packets according to their final destination. The combination is called the **Batcher-banyan switch** (see Figure 6.28).

Figure 6.28 *Batcher-banyan switch*



The sorting switch uses hardware merging techniques, but we will not discuss the details here. Normally, another hardware module called a trap is added between the Batcher switch and the banyan switch. The trap module prevents duplicate packets (packets with the same output destination) from passing to the banyan switch simultaneously. Only one packet for each destination is allowed at each tick; if there is more than one, they wait for the next tick.

6.4 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], and [Kur & Ros 08].

RFCs

Forwarding is discussed in RFC 1812, RFC 1971, and RFC 1980. MPLS is discussed in RFC 3031, RFC 3032, RFC 3036, and RFC 3212.

6.5 KEY TERMS

banyan switch
Batcher-banyan switch
crossbar switch
crosspoint
default method
delivery
direct delivery
forwarding
host-specific method
indirect delivery
input ports
longest mask matching
longest prefix match
network-specific method
next-hop address
next-hop method
output ports
routing processor

6.6 SUMMARY

- The network layer supervises the handling of the packets by the underlying physical networks.We define this handling as the delivery of a packet. The delivery of a packet is called direct if the deliverer (host or router) and the destination are on the same network. The delivery of a packet is called indirect if the deliverer (host or router) and the destination are on different networks.
- Forwarding means to deliver the packet to the next hop. Two categories of forwarding were discussed in this chapter: forwarding based on the destination address of the IP datagram and forwarding based on the label attached to an IP datagram. The first searches a routing table to forward a packet; the second uses the label as an index to a switching table to forward a packet.
- We discussed several methods in destination-address–based forwarding including host-specific method, next-hop method, network-specific method, and the default method.
- In destination-address–based forwarding, the routing table for classful forwarding can have three columns. The routing table for classless addressing needs at least four columns. Address aggregation simplifies the forwarding process in classless addressing. Longest mask matching is required in classless addressing.

- In label-based forwarding, a switching table is used instead of a routing table. The Multi-Protocol Label Switching (MPLS) is the standard approved by IETF, which adds a pseudo layer to the TCP/IP protocol suite by encapsulating the IP packet in an MPLS packet.
- A router is normally made of four components: input ports, output ports, the routing processor, and the switching fabric.

6.7 PRACTICE SET

Exercises

1. A host with IP address 137.23.56.23/16 sends a packet to a host with IP address 137.23.67.9/16. Is the delivery direct or indirect? Assume no subnetting.
2. A host with IP address 137.23.56.23/16 sends a packet to a host with IP address 142.3.6.9/24. Is the delivery direct or indirect? Assume no subnetting.
3. In Figure 6.8, find the routing table for router R2.
4. In Figure 6.8, find the routing table for router R3.
5. A packet arrives at router R1 in Figure 6.8 with destination address 192.16.7.42. Show how it is forwarded.
6. A packet arrives at router R1 in Figure 6.8 with destination address 145.80.14.26. Show how it is forwarded.
7. A packet arrives at router R1 in Figure 6.8 with destination address 147.26.50.30. Show how it is forwarded.
8. A packet arrives at the router in Figure 6.11 with destination address 145.14.192.71. Show how it is forwarded.
9. A packet arrives at the router in Figure 6.11 with destination address 135.11.80.21. Show how it is forwarded.
10. A packet arrives at router R1 in Figure 6.13 with destination address 201.4.16.70. Show how it is forwarded.
11. A packet arrives at router R1 in Figure 6.13 with destination address 202.70.20.30. Show how it is forwarded.
12. Show a routing table for a host that is totally isolated.
13. Show a routing table for a host that is connected to a LAN without being connected to the Internet.
14. Find the topology of the network if Table 6.3 is the routing table for router R1.

Table 6.3 *Routing table for Exercise 14*

Mask	*Network Address*	*Next-Hop Address*	*Interface*
/27	202.14.17.224	----	m1
/18	145.23.192.0	----	m0
default	default	130.56.12.4	m2

15. Can router R1 in Figure 6.16 receive a packet with destination address 140.24.7.194? Explain your answer.
16. Can router R1 in Figure 6.16 receive a packet with destination address 140.24.7.42? Explain your answer.
17. Show the routing table for regional ISP in Figure 6.17.
18. Show the routing table for local ISP 1 in Figure 6.17.
19. Show the routing table for local ISP 2 in Figure 6.17.
20. Show the routing table for local ISP 3 in Figure 6.17.
21. Show the routing table for small ISP 1 in Figure 6.17.

Research Activities

22. Show how an MPLS packet is encapsulated in a frame. Find out what should be the value of the type field when the protocol is the Ethernet.
23. Compare Multi-Layer Switching (MLS) used by Cisco Systems Inc. with MPLS technology we described here.
24. Some people argue that the MPLS should be called layer 2.5 in the TCP/IP protocol suite. Do you agree? Explain.
25. Find how your ISP uses address aggregation and longest mask match principles.
26. Find whether or not your IP address is part of the geographical address allocation.
27. If you are using a router, find the number and names of the columns in the routing table.
28. Cisco is one of the dominant manufacturers of routers. Find information about the different types of routers manufactured by this company.

CHAPTER

7

Internet Protocol Version 4 (IPv4)

After discussing the IP addressing mechanism and delivery and forwarding of IP packets, we discuss the format of the IP packet in this chapter. We show how an IP packet is made of a base header and options that sometimes are used to facilitate or control the delivery of packets.

OBJECTIVES

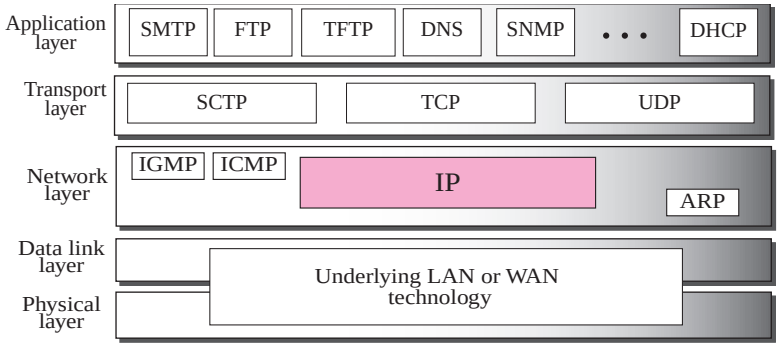
The chapter has several objectives:

- To explain the general idea behind the IP protocol and show the position of IP in relation to other protocols in TCP/IP protocol suite.
- To show the general format of an IPv4 datagram and list the fields in the header.
- To discuss fragmentation and reassembly of datagrams and how the original datagram can be recovered out of fragmented ones.
- To discuss several options that can be in an IPv4 datagram and their applications.
- To discuss some reserved blocks in the address space and their applications.
- To show how a checksum is calculated for the header of an IPv4 datagram at the sending site and how the checksum is checked at the receiver site.
- To discuss IP over ATM and compare it with IP over LANs and/or point-to-point WANs.
- To show a simplified version of the IP package and give the pseudocode for some modules.

7.1 INTRODUCTION

The **Internet Protocol (IP)** is the transmission mechanism used by the TCP/IP protocols at the network layer. Figure 7.1 shows the position of IP in the suite.

Figure 7.1 *Position of IP in TCP/IP protocol suite*



IP is an unreliable and connectionless datagram protocol—a **best-effort delivery** service. The term *best-effort* means that IP packets can be corrupted, lost, arrive out of order, or delayed and may create congestion for the network.

If reliability is important, IP must be paired with a reliable protocol such as TCP. An example of a more commonly understood best-effort delivery service is the post office. The post office does its best to deliver the mail but does not always succeed. If an unregistered letter is lost, it is up to the sender or would-be recipient to discover the loss and rectify the problem. The post office itself does not keep track of every letter and cannot notify a sender of loss or damage.

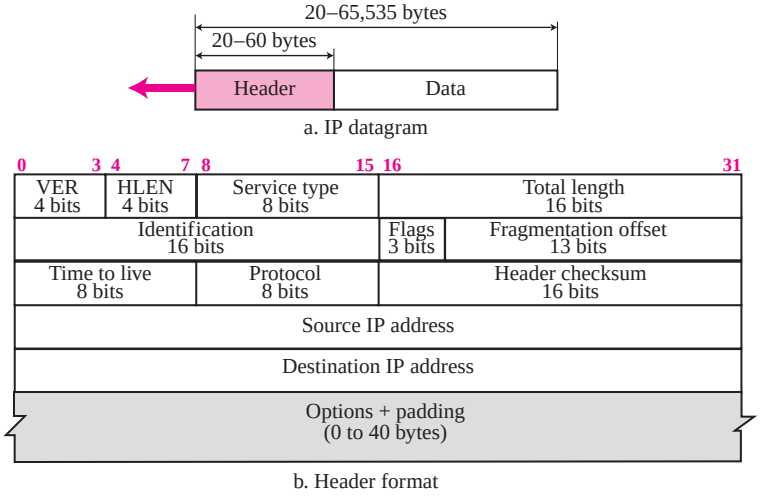
IP is also a connectionless protocol for a packet switching network that uses the datagram approach (see Chapter 4). This means that each datagram is handled independently, and each datagram can follow a different route to the destination. This implies that datagrams sent by the same source to the same destination could arrive out of order. Also, some could be lost or corrupted during transmission. Again, IP relies on a higher-level protocol to take care of all these problems.

7.2 DATAGRAMS

Packets in the network (internet) layer are called **datagrams.** Figure 7.2 shows the IP datagram format. A datagram is a variable-length packet consisting of two parts: header and data. The header is 20 to 60 bytes in length and contains information essential to

routing and delivery. It is customary in TCP/IP to show the header in 4-byte sections. A brief description of each field is in order.

Figure 7.2 *IP datagram*

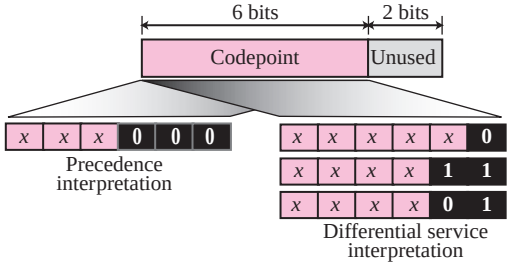


- **Version (VER).** This 4-bit field defines the version of the IP protocol. Currently the version is 4. However, version 6 (or IPv6) may totally replace version 4 in the future. This field tells the IP software running in the processing machine that the datagram has the format of version 4. All fields must be interpreted as specified in the fourth version of the protocol. If the machine is using some other version of IP, the datagram is discarded rather than interpreted incorrectly.
- **Header length (HLEN).** This 4-bit field defines the total length of the datagram header in 4-byte words. This field is needed because the length of the header is variable (between 20 and 60 bytes). When there are no options, the **header length** is 20 bytes, and the value of this field is 5 ($5 \times 4 = 20$). When the option field is at its maximum size, the value of this field is 15 ($15 \times 4 = 60$).
- **Service type.** In the original design of IP header, this field was referred to as **type of service (TOS)**, which defined how the datagram should be handled. Part of the field was used to define the precedence of the datagram; the rest defined the type of service (low delay, high throughput, and so on). IETF has changed the interpretation of this 8-bit field. This field now defines a set of **differentiated services.** The new interpretation is shown in Figure 7.3.

  In this interpretation, the first 6 bits make up the **codepoint** subfield and the last 2 bits are not used. The codepoint subfield can be used in two different ways.

  a. When the 3 right-most bits are 0s, the 3 left-most bits are interpreted the same as the precedence bits in the service type interpretation. In other words, it is compatible with the old interpretation. The precedence defines the eight-level

Figure 7.3 *Service type*



priority of the datagram (0 to 7) in issues such as congestion. If a router is congested and needs to discard some datagrams, those datagrams with lowest precedence are discarded first. Some datagrams in the Internet are more important than the others. For example, a datagram used for network management is much more urgent and important than a datagram containing optional information for a group.

b. When the 3 right-most bits are not all 0s, the 6 bits define 56 (64 – 8) services based on the priority assignment by the Internet or local authorities according to Table 7.1. The first category contains 24 service types; the second and the third each contain 16. The first category is assigned by the Internet authorities (IETF). The second category can be used by local authorities (organizations). The third category is temporary and can be used for experimental purposes. Note that these assignments have not yet been finalized.

Table 7.1 *Values for codepoints*

Category	*Codepoint*	*Assigning Authority*
1	XXXXX0	Internet
2	XXXX11	Local
3	XXXX01	Temporary or experimental

❑ **Total length.** This is a 16-bit field that defines the total length (header plus data) of the IP datagram in bytes. To find the length of the data coming from the upper layer, subtract the header length from the total length. The header length can be found by multiplying the value in the HLEN field by four.

Length of data = total length – header length

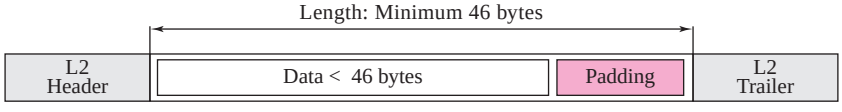
Since the field length is 16 bits, the total length of the IP datagram is limited to 65,535 ($2^{16} - 1$) bytes, of which 20 to 60 bytes are the header and the rest is data from the upper layer.

The *total length* field defines the total length of the datagram including the header.

Though a size of 65,535 bytes might seem large, the size of the IP datagram may increase in the near future as the underlying technologies allow even more throughput (more bandwidth). When we discuss fragmentation in the next section, we will see that some physical networks are not able to encapsulate a datagram of 65,535 bytes in their frames. The datagram must be fragmented to be able to pass through those networks.

One may ask why we need this field anyway. When a machine (router or host) receives a frame, it drops the header and the trailer leaving the datagram. Why include an extra field that is not needed? The answer is that in many cases we really do not need the value in this field. However, there are occasions in which the datagram is not the only thing encapsulated in a frame; it may be that padding has been added. For example, the Ethernet protocol has a minimum and maximum restriction on the size of data that can be encapsulated in a frame (46 to 1500 bytes). If the size of an IP datagram is less than 46 bytes, some padding will be added to meet this requirement. In this case, when a machine decapsulates the datagram, it needs to check the total length field to determine how much is really data and how much is padding (see Figure 7.4).

Figure 7.4 *Encapsulation of a small datagram in an Ethernet frame*



- **Identification.** This field is used in fragmentation (discussed in the next section).
- **Flags.** This field is used in fragmentation (discussed in the next section).
- **Fragmentation offset.** This field is used in fragmentation (discussed in the next section).
- **Time to live.** A datagram has a limited lifetime in its travel through an internet. This field was originally designed to hold a timestamp, which was decremented by each visited router. The datagram was discarded when the value became zero. However, for this scheme, all the machines must have synchronized clocks and must know how long it takes for a datagram to go from one machine to another. Today, this field is mostly used to control the maximum number of hops (routers) visited by the datagram. When a source host sends the datagram, it stores a number in this field. This value is approximately two times the maximum number of routes between any two hosts. Each router that processes the datagram decrements this number by one. If this value, after being decremented, is zero, the router discards the datagram.

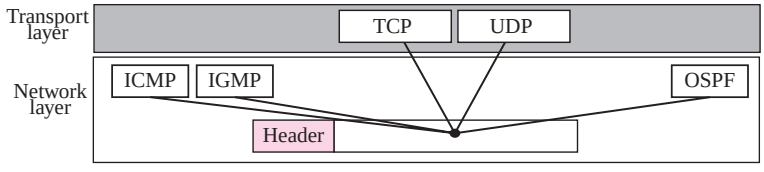
  This field is needed because routing tables in the Internet can become corrupted. A datagram may travel between two or more routers for a long time without ever getting delivered to the destination host. This field limits the lifetime of a datagram.

  Another use of this field is to intentionally limit the journey of the packet. For example, if the source wants to confine the packet to the local network, it can store

1 in this field. When the packet arrives at the first router, this value is decremented to 0, and the datagram is discarded.

- **Protocol.** This 8-bit field defines the higher-level protocol that uses the services of the IP layer. An IP datagram can encapsulate data from several higher level protocols such as TCP, UDP, ICMP, and IGMP. This field specifies the final destination protocol to which the IP datagram should be delivered. In other words, since the IP protocol multiplexes and demultiplexes data from different higher-level protocols, the value of this field helps in the demultiplexing process when the datagram arrives at its final destination (see Figure 7.5).

Figure 7.5 *Multiplexing*



Some of the value of this field for different higher-level protocols is shown in Table 7.2.

Table 7.2 *Protocols*

Value	*Protocol*	*Value*	*Protocol*
1	ICMP	17	UDP
2	IGMP	89	OSPF
6	TCP		

- **Checksum.** The checksum concept and its calculation are discussed later in this chapter.
- **Source address.** This 32-bit field defines the IP address of the source. This field must remain unchanged during the time the IP datagram travels from the source host to the destination host.
- **Destination address.** This 32-bit field defines the IP address of the destination. This field must remain unchanged during the time the IP datagram travels from the source host to the destination host.

Example 7.1

An IP packet has arrived with the first 8 bits as shown:

```
01000010
```

The receiver discards the packet. Why?

Solution

There is an error in this packet. The 4 left-most bits (0100) show the version, which is correct. The next 4 bits (0010) show the wrong header length ($2 \times 4 = 8$). The minimum number of bytes in the header must be 20. The packet has been corrupted in transmission.

Example 7.2

In an IP packet, the value of HLEN is 1000 in binary. How many bytes of options are being carried by this packet?

Solution

The HLEN value is 8, which means the total number of bytes in the header is 8×4 or 32 bytes. The first 20 bytes are the base header, the next 12 bytes are the options.

Example 7.3

In an IP packet, the value of HLEN is 5_{16} and the value of the total length field is 0028_{16}. How many bytes of data are being carried by this packet?

Solution

The HLEN value is 5, which means the total number of bytes in the header is 5×4 or 20 bytes (no options). The total length is 40 bytes, which means the packet is carrying 20 bytes of data ($40 - 20$).

Example 7.4

An IP packet has arrived with the first few hexadecimal digits as shown below:

```
45000028000100000102 . . .
```

How many hops can this packet travel before being dropped? The data belong to what upper layer protocol?

Solution

To find the time-to-live field, we skip 8 bytes (16 hexadecimal digits). The time-to-live field is the ninth byte, which is 01. This means the packet can travel only one hop. The protocol field is the next byte (02), which means that the upper layer protocol is IGMP (see Table 7.2).

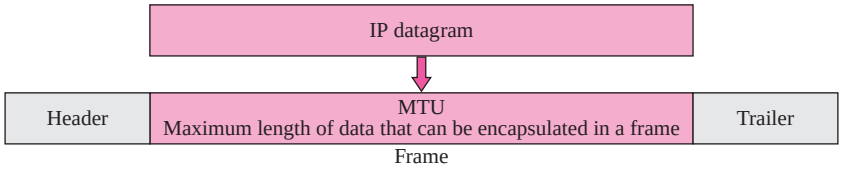
7.3 FRAGMENTATION

A datagram can travel through different networks. Each router decapsulates the IP datagram from the frame it receives, processes it, and then encapsulates it in another frame. The format and size of the received frame depend on the protocol used by the physical network through which the frame has just traveled. The format and size of the sent frame depend on the protocol used by the physical network through which the frame is going to travel. For example, if a router connects a LAN to a WAN, it receives a frame in the LAN format and sends a frame in the WAN format.

Maximum Transfer Unit (MTU)

Each data link layer protocol has its own frame format in most protocols. One of the fields defined in the format is the maximum size of the data field. In other words, when a datagram is encapsulated in a frame, the total size of the datagram must be less than this maximum size, which is defined by the restrictions imposed by the hardware and software used in the network (see Figure 7.6).

Figure 7.6 *MTU*



The value of the MTU differs from one physical network protocol to another. For example, the value for the Ethernet LAN is 1500 bytes, for FDDI LAN is 4352 bytes, and for PPP is 296 bytes.

In order to make the IP protocol independent of the physical network, the designers decided to make the maximum length of the IP datagram equal to 65,535 bytes. This makes transmission more efficient if we use a protocol with an MTU of this size. However, for other physical networks, we must divide the datagram to make it possible to pass through these networks. This is called **fragmentation.**

The source usually does not fragment the IP packet. The transport layer will instead segment the data into a size that can be accommodated by IP and the data link layer in use.

When a datagram is fragmented, each fragment has its own header with most of the fields repeated, but some changed. A fragmented datagram may itself be fragmented if it encounters a network with an even smaller MTU. In other words, a datagram can be fragmented several times before it reaches the final destination.

A datagram can be fragmented by the source host or any router in the path. The reassembly of the datagram, however, is done only by the destination host because each fragment becomes an independent datagram. Whereas the fragmented datagram can travel through different routes, and we can never control or guarantee which route a fragmented datagram may take, all of the fragments belonging to the same datagram should finally arrive at the destination host. So it is logical to do the reassembly at the final destination. An even stronger objection for reassembling packets during the transmission is the loss of efficiency it incurs.

When a datagram is fragmented, required parts of the header must be copied by all fragments. The option field may or may not be copied as we will see in the next section. The host or router that fragments a datagram must change the values of three fields: flags, fragmentation offset, and total length. The rest of the fields must be copied. Of course, the value of the checksum must be recalculated regardless of fragmentation.

Only data in a datagram is fragmented.

Fields Related to Fragmentation

The fields that are related to fragmentation and reassembly of an IP datagram are the identification, flags, and fragmentation offset fields.

- **Identification.** This 16-bit field identifies a datagram originating from the source host. The combination of the identification and source IP address must uniquely

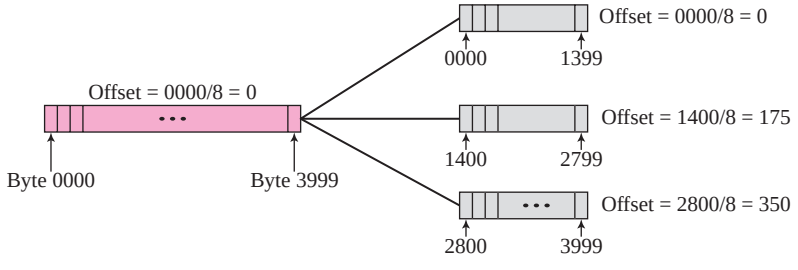
define a datagram as it leaves the source host. To guarantee uniqueness, the IP protocol uses a counter to label the datagrams. The counter is initialized to a positive number. When the IP protocol sends a datagram, it copies the current value of the counter to the identification field and increments the counter by one. As long as the counter is kept in the main memory, uniqueness is guaranteed. When a datagram is fragmented, the value in the identification field is copied into all fragments. In other words, all fragments have the same identification number, which is also the same as the original datagram. The identification number helps the destination in reassembling the datagram. It knows that all fragments having the same identification value should be assembled into one datagram.

- **Flags.** This is a three-bit field. The first bit is reserved (not used). The second bit is called the *do not fragment* bit. If its value is 1, the machine must not fragment the datagram. If it cannot pass the datagram through any available physical network, it discards the datagram and sends an ICMP error message to the source host (see Chapter 9). If its value is 0, the datagram can be fragmented if necessary. The third bit is called the *more fragment* bit. If its value is 1, it means the datagram is not the last fragment; there are more fragments after this one. If its value is 0, it means this is the last or only fragment (see Figure 7.7).

Figure 7.7 *Flags field*

- **Fragmentation offset.** This 13-bit field shows the relative position of this fragment with respect to the whole datagram. It is the offset of the data in the original datagram measured in units of 8 bytes. Figure 7.8 shows a datagram with a data size of 4000 bytes fragmented into three fragments. The bytes in the original datagram are numbered 0 to 3999. The first fragment carries bytes 0 to 1399. The offset for this datagram is 0/8 = 0. The second fragment carries bytes 1400 to 2799; the offset value for this fragment is 1400/8 = 175. Finally, the third fragment carries bytes 2800 to 3999. The offset value for this fragment is 2800/8 = 350.

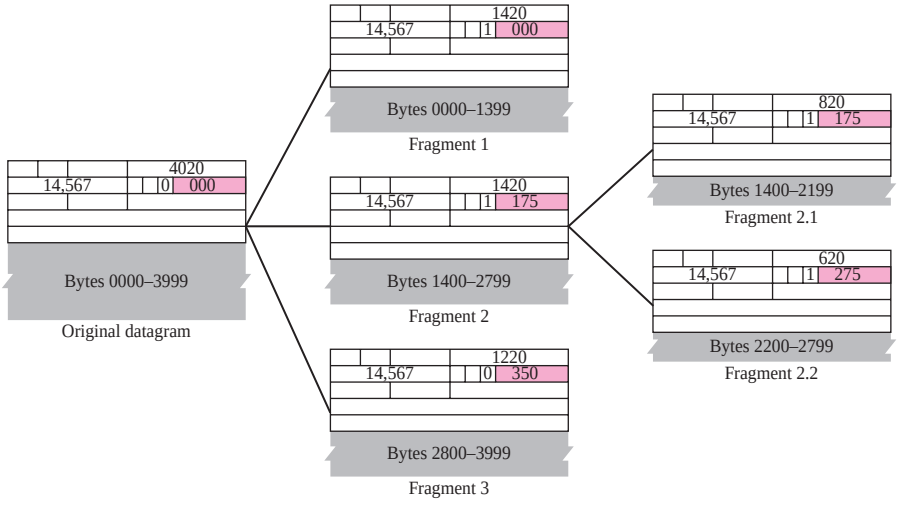
Figure 7.8 *Fragmentation example*



Remember that the value of the offset is measured in units of 8 bytes. This is done because the length of the offset field is only 13 bits long and cannot represent a sequence of bytes greater than 8191. This forces hosts or routers that fragment datagrams to choose the size of each fragment so that the first byte number is divisible by 8.

Figure 7.9 shows an expanded view of the fragments in the previous figure. Notice the value of the identification field is the same in all fragments. Notice the value of the flags field with the *more* bit set for all fragments except the last. Also, the value of the offset field for each fragment is shown.

Figure 7.9 *Detailed fragmentation example*



The figure also shows what happens if a fragment itself is fragmented. In this case the value of the offset field is always relative to the original datagram. For example, in the figure, the second fragment is itself fragmented later to two fragments of 800 bytes and 600 bytes, but the offset shows the relative position of the fragments to the original data.

It is obvious that even if each fragment follows a different path and arrives out of order, the final destination host can reassemble the original datagram from the fragments received (if none of them is lost) using the following strategy:

a. The first fragment has an offset field value of zero.

b. Divide the length of the first fragment by 8. The second fragment has an offset value equal to that result.

c. Divide the total length of the first and second fragment by 8. The third fragment has an offset value equal to that result.

d. Continue the process. The last fragment has a *more* bit value of 0.

Example 7.5

A packet has arrived with an *M* bit value of 0. Is this the first fragment, the last fragment, or a middle fragment? Do we know if the packet was fragmented?

Solution

If the *M* bit is 0, it means that there are no more fragments; the fragment is the last one. However, we cannot say if the original packet was fragmented or not. A nonfragmented packet is considered the last fragment.

Example 7.6

A packet has arrived with an *M* bit value of 1. Is this the first fragment, the last fragment, or a middle fragment? Do we know if the packet was fragmented?

Solution

If the *M* bit is 1, it means that there is at least one more fragment. This fragment can be the first one or a middle one, but not the last one. We don't know if it is the first one or a middle one; we need more information (the value of the fragmentation offset). See also the next example.

Example 7.7

A packet has arrived with an *M* bit value of 1 and a fragmentation offset value of zero. Is this the first fragment, the last fragment, or a middle fragment?

Solution

Because the *M* bit is 1, it is either the first fragment or a middle one. Because the offset value is 0, it is the first fragment.

Example 7.8

A packet has arrived in which the offset value is 100. What is the number of the first byte? Do we know the number of the last byte?

Solution

To find the number of the first byte, we multiply the offset value by 8. This means that the first byte number is 800. We cannot determine the number of the last byte unless we know the length of the data.

Example 7.9

A packet has arrived in which the offset value is 100, the value of HLEN is 5 and the value of the total length field is 100. What is the number of the first byte and the last byte?

Solution

The first byte number is $100 \times 8 = 800$. The total length is 100 bytes and the header length is 20 bytes (5×4), which means that there are 80 bytes in this datagram. If the first byte number is 800, the last byte number must be 879.

7.4 OPTIONS

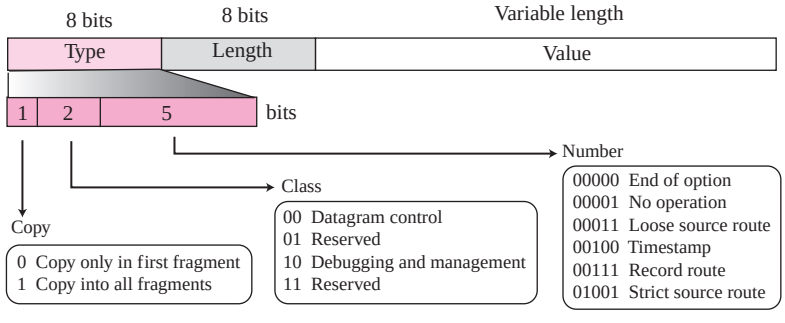
The header of the IP datagram is made of two parts: a fixed part and a variable part. The fixed part is 20 bytes long and was discussed in the previous section. The variable part comprises the options, which can be a maximum of 40 bytes.

Options, as the name implies, are not required for a datagram. They can be used for network testing and debugging. Although options are not a required part of the IP header, option processing is required of the IP software. This means that all implementations must be able to handle options if they are present in the header.

Format

Figure 7.10 shows the format of an option. It is composed of a 1-byte type field, a 1-byte length field, and a variable-sized value field. The three fields are often referred to as type-length-value or TLV.

Figure 7.10 *Option format*



Type

The **type field** is 8 bits long and contains three subfields: copy, class, and number.

- **Copy.** This 1-bit subfield controls the presence of the option in fragmentation. When its value is 0, it means that the option must be copied only to the first fragment. If its value is 1, it means the option must be copied to all fragments.
- **Class.** This 2-bit subfield defines the general purpose of the option. When its value is 00, it means that the option is used for datagram control. When its value is 10, it means that the option is used for debugging and management. The other two possible values (01 and 11) have not yet been defined.
- **Number.** This 5-bit subfield defines the type of option. Although 5 bits can define up to 32 different types, currently only 6 types are in use. These will be discussed in a later section.

Length

The **length field** defines the total length of the option including the type field and the length field itself. This field is not present in all of the option types.

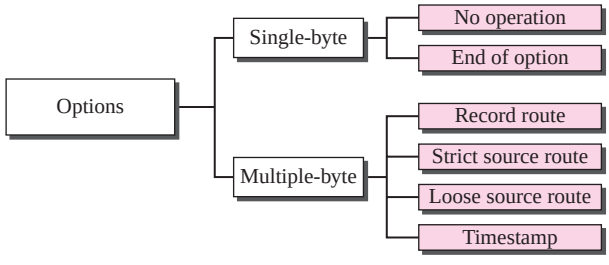
Value

The **value field** contains the data that specific options require. Like the length field, this field is also not present in all option types.

Option Types

As mentioned previously, only six options are currently being used. Two of these are 1-byte options, and they do not require the length or the data fields. Four of them are multiple-byte options; they require the length and the data fields (see Figure 7.11).

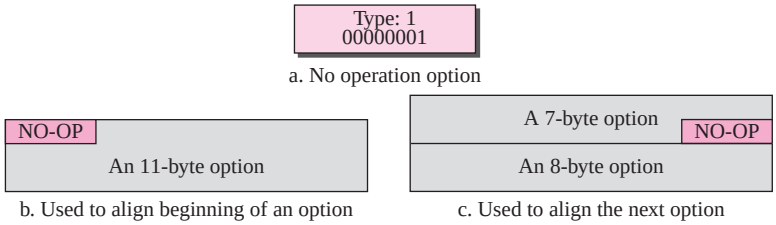
Figure 7.11 *Categories of options*



No-Operation Option

A **no-operation option** is a 1-byte option used as a filler between options. For example, it can be used to align the next option on a 16-bit or 32-bit boundary (see Figure 7.12).

Figure 7.12 *No operation option*

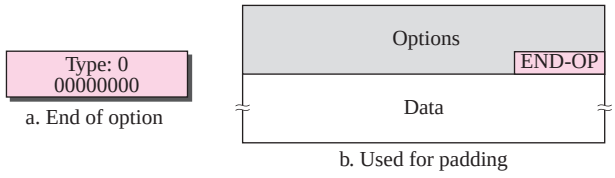


End-of-Option Option

An **end-of-option option** is also a 1-byte option used for padding at the end of the option field. It, however, can only be used as the last option. Only one end-of-option option can be used. After this option, the receiver looks for the payload data. This

means that if more than 1 byte is needed to align the option field, some no-operation options must be used, followed by an end-of-option option (see Figure 7.13).

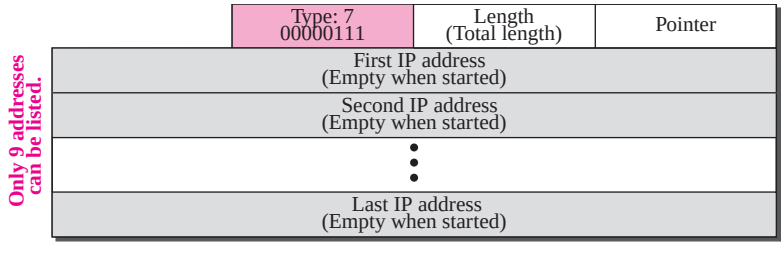
Figure 7.13 *End-of-option option*



Record-Route Option

A **record-route option** is used to record the Internet routers that handle the datagram. It can list up to nine router IP addresses since the maximum size of the header is 60 bytes, which must include 20 bytes for the base header. This implies that only 40 bytes are left over for the option part. The source creates placeholder fields in the option to be filled by the visited routers. Figure 7.14 shows the format of the record route option.

Figure 7.14 *Record-route option*

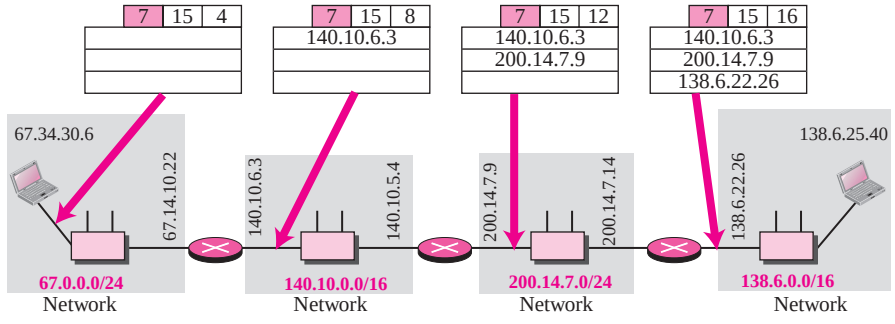


Both the code and length fields have been described above. The **pointer field** is an offset integer field containing the byte number of the first empty entry. In other words, it points to the first available entry.

The source creates empty fields for the IP addresses in the data field of the option. When the datagram leaves the source, all of the fields are empty. The pointer field has a value of 4, pointing to the first empty field.

When the datagram is traveling, each router that processes the datagram compares the value of the pointer with the value of the length. If the value of the pointer is greater than the value of the length, the option is full and no changes are made. However, if the value of the pointer is not greater than the value of the length, the router inserts its outgoing IP address in the next empty field (remember that a router has more than one IP address). In this case, the router adds the IP address of its interface from which the datagram is leaving. The router then increments the value of the pointer by 4. Figure 7.15 shows the entries as the datagram travels left to right from router to router.

Figure 7.15 *Record-route concept*



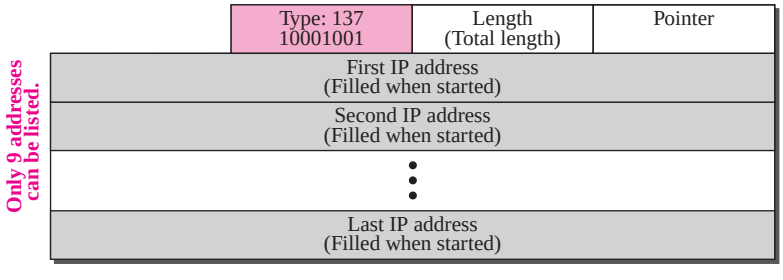
Strict-Source-Route Option

A **strict-source-route option** is used by the source to predetermine a route for the datagram as it travels through the Internet. Dictation of a route by the source can be useful for several purposes. The sender can choose a route with a specific type of service, such as minimum delay or maximum throughput. Alternatively, it may choose a route that is safer or more reliable for the sender's purpose. For example, a sender can choose a route so that its datagram does not travel through a competitor's network.

If a datagram specifies a strict source route, all of the routers defined in the option must be visited by the datagram. A router must not be visited if its IP address is not listed in the datagram. If the datagram visits a router that is not on the list, the datagram is discarded and an error message is issued. If the datagram arrives at the destination and some of the entries were not visited, it will also be discarded and an error message issued.

Regular users of the Internet, however, are not usually aware of the physical topology of the Internet. Consequently, strict source routing is not the choice of most users. Figure 7.16 shows the format of the strict source route option.

Figure 7.16 *Strict-source-route option*

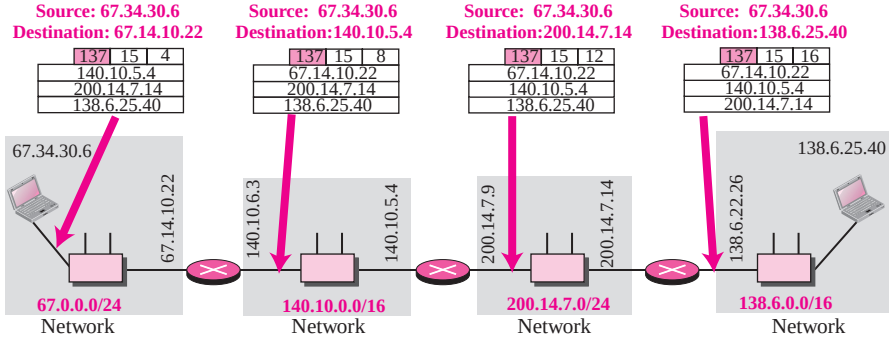


The format is similar to the record route option with the exception that all of the IP addresses are entered by the sender.

When the datagram is traveling, each router that processes the datagram compares the value of the pointer with the value of the length. If the value of the pointer is greater

than the value of the length, the datagram has visited all of the predefined routers. The datagram cannot travel anymore; it is discarded and an error message is created. If the value of the pointer is not greater than the value of the length, the router compares the destination IP address with its incoming IP address: If they are equal, it processes the datagram, swaps the IP address pointed by the pointer with the destination address, increments the pointer value by 4, and forwards the datagram. If they are not equal, it discards the datagram and issues an error message. Figure 7.17 shows the actions taken by each router as a datagram travels from source to destination.

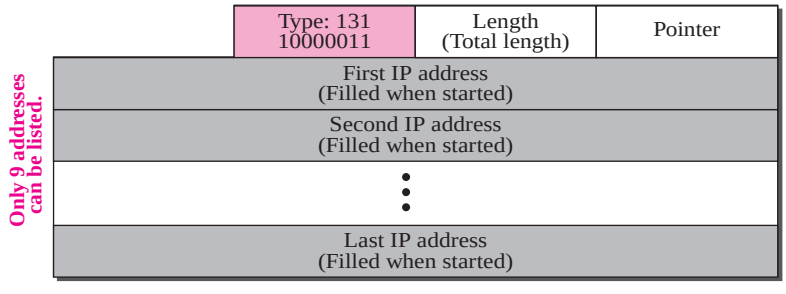
Figure 7.17 *Strict-source-route concept*



Loose-Source-Route Option

A **loose-source-route option** is similar to the strict source route, but it is more relaxed. Each router in the list must be visited, but the datagram can visit other routers as well. Figure 7.18 shows the format of the loose source route option.

Figure 7.18 *Loose-source-route option*



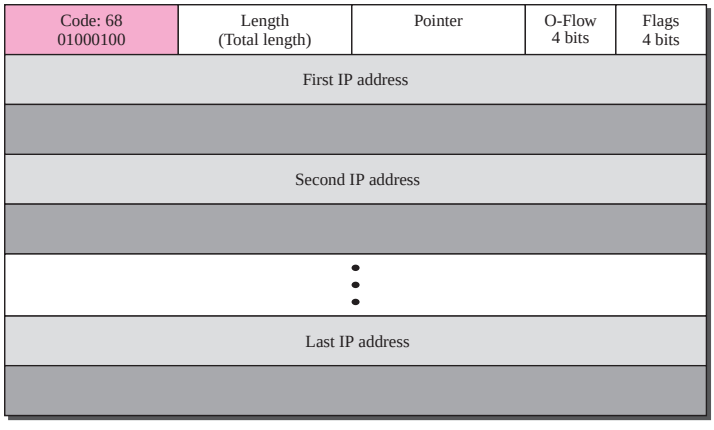
Timestamp

A **timestamp option** is used to record the time of datagram processing by a router. The time is expressed in milliseconds from midnight, Universal Time. Knowing the time a

datagram is processed can help users and managers track the behavior of the routers in the Internet. We can estimate the time it takes for a datagram to go from one router to another. We say *estimate* because, although all routers may use Universal Time, their local clocks may not be synchronized.

However, nonprivileged users of the Internet are not usually aware of the physical topology of the Internet. Consequently, a timestamp option is not a choice for most users. Figure 7.19 shows the format of the timestamp option.

Figure 7.19 *Timestamp option*



In this figure, the definitions of the code and length fields are the same as before. The overflow field records the number of routers that could not add their timestamp because no more fields were available. The flags field specifies the visited router responsibilities. If the flag value is 0, each router adds only the timestamp in the provided field. If the flag value is 1, each router must add its outgoing IP address and the timestamp. If the value is 3, the IP addresses are given, and each router must check the given IP address with its own incoming IP address. If there is a match, the router overwrites the IP address with its outgoing IP address and adds the timestamp (see Figure 7.20).

Figure 7.20 *Use of flag in timestamp*

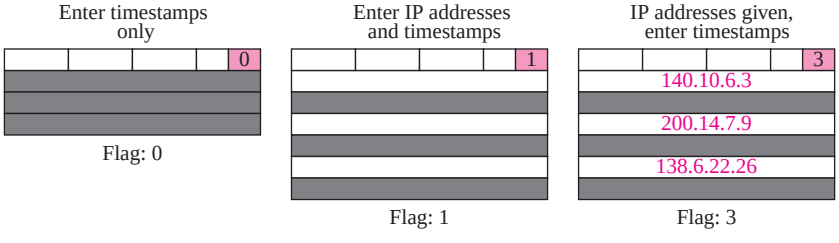
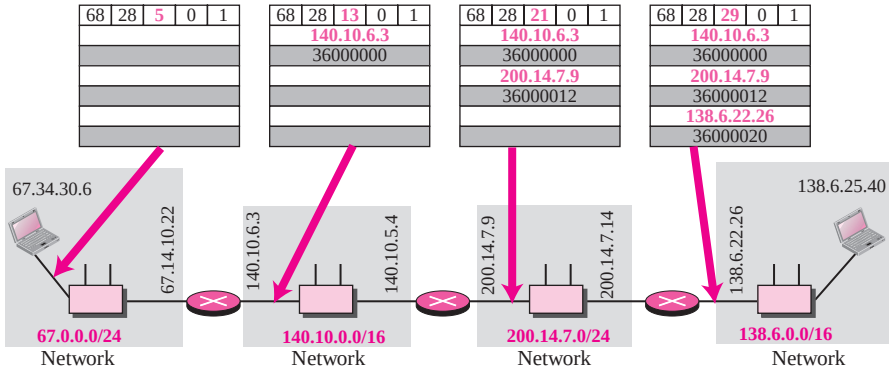


Figure 7.21 shows the actions taken by each router when a datagram travels from source to destination. The figure assumes a flag value of 1.

Figure 7.21 *Timestamp concept*



Example 7.10

Which of the six options must be copied to each fragment?

Solution

We look at the first (left-most) bit of the type for each option.

- **a.** No operation: type is **0**0000001; not copied.
- **b.** End of option: type is **0**0000000; not copied.
- **c.** Record route: type is **0**0000111; not copied.
- **d.** Strict source route: type is **1**0001001; copied.
- **e.** Loose source route: type is **1**0000011; copied.
- **f.** Timestamp: type is **0**1000100; not copied.

Example 7.11

Which of the six options are used for datagram control and which are used for debugging and management?

Solution

We look at the second and third (left-most) bits of the type.

- **a.** No operation: type is 0**00**00001; datagram control.
- **b.** End of option: type is 0**00**00000; datagram control.
- **c.** Record route: type is 0**00**00111; datagram control.
- **d.** Strict source route: type is 1**00**01001; datagram control.
- **e.** Loose source route: type is 1**00**00011; datagram control.
- **f.** Timestamp: type is 0**10**00100; debugging and management control.

Example 7.12

One of the utilities available in UNIX to check the traveling of the IP packets is **ping.** In the next chapter, we talk about the *ping* program in more detail. In this example, we want to show how to use the program to see if a host is available. We ping a server at De Anza College named *fhda.edu*. The result shows that the IP address of the host is 153.18.8.1. The result also shows the number of bytes used.

```
$  ping fhda.edu
PING fhda.edu (153.18.8.1) 56(84) bytes of data.
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq =
0 ttl=62 time=1.87 ms
...
```

Example 7.13

We can also use the *ping* utility with the -R option to implement the record route option. The result shows the interfaces and IP addresses.

```
$ ping -R fhda.edu
PING fhda.edu (153.18.8.1) 56(124) bytes of data.
64 bytes from tiptoe.fhda.edu
(153.18.8.1): icmp_seq=0 ttl=62 time=2.70 ms
RR:  voyager.deanza.fhda.edu (153.18.17.11)
     Dcore_G0_3-69.fhda.edu (153.18.251.3)
     Dbackup_V13.fhda.edu (153.18.191.249)
     tiptoe.fhda.edu (153.18.8.1)
     Dbackup_V62.fhda.edu (153.18.251.34)
     Dcore_G0_1-6.fhda.edu (153.18.31.254)
     voyager.deanza.fhda.edu (153.18.17.11)
```

Example 7.14

The **traceroute** utility can also be used to keep track of the route of a packet. The result shows the three routers visited.

```
$ traceroute fhda.edu
traceroute to fhda.edu (153.18.8.1), 30 hops max, 38 byte packets
 1 Dcore_G0_1-6.fhda.edu (153.18.31.254)  0.972 ms  0.902 ms
   0.881 ms
 2 Dbackup_V69.fhda.edu (153.18.251.4)  2.113 ms  1.996 ms
   2.059 ms
 3 tiptoe.fhda.edu (153.18.8.1)  1.791 ms  1.741 ms  1.751 ms
```

Example 7.15

The traceroute program can be used to implement loose source routing. The -g option allows us to define the routers to be visited, from the source to destination. The following shows how we can send a packet to the fhda.edu server with the requirement that the packet visit the router 153.18.251.4.

```
$ traceroute -g  153.18.251.4 fhda.edu.
traceroute to fhda.edu (153.18.8.1), 30 hops max, 46 byte packets
 1  Dcore_G0_1-6.fhda.edu (153.18.31.254)  0.976 ms  0.906 ms
    0.889 ms
 2  Dbackup_V69.fhda.edu (153.18.251.4)  2.168 ms  2.148 ms
    2.037 ms
```

Example 7.16

The traceroute program can also be used to implement strict source routing. The -G option forces the packet to visit the routers defined in the command line. The following shows how we can send a packet to the *fhda.edu* server and force the packet to visit only the router 153.18.251.4.

```
$ traceroute -G  153.18.251.4 fhda.edu.
traceroute to fhda.edu (153.18.8.1), 30 hops max, 46 byte packets
  1  Dbackup_V69.fhda.edu (153.18.251.4)  2.168 ms  2.148 ms
     2.037 ms
```

7.5 CHECKSUM

The error detection method used by most TCP/IP protocols is called the **checksum.** The checksum protects against the corruption that may occur during the transmission of a packet. It is redundant information added to the packet.

The checksum is calculated at the sender and the value obtained is sent with the packet. The receiver repeats the same calculation on the whole packet including the checksum. If the result is satisfactory (see below), the packet is accepted; otherwise, it is rejected.

Checksum Calculation at the Sender

At the sender, the packet header is divided into *n*-bit sections (*n* is usually 16). These sections are added together using one's complement arithmetic (see Appendix D), resulting in a sum that is also *n* bits long. The sum is then complemented (all 0s changed to 1s and all 1s to 0s) to produce the checksum.

To create the checksum the sender does the following:

- The packet is divided into *k* sections, each of *n* bits.
- All sections are added together using one's complement arithmetic.
- The final result is complemented to make the checksum.

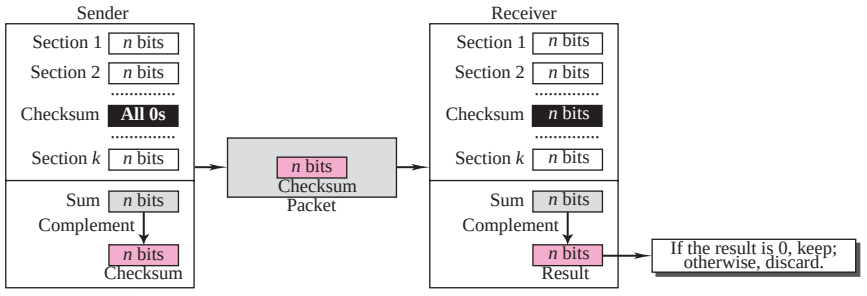
Checksum Calculation at the Receiver

The receiver divides the received packet into *k* sections and adds all sections. It then complements the result. If the final result is 0, the packet is accepted; otherwise, it is rejected. Figure 7.22 shows graphically what happens at the sender and the receiver.

We said when the receiver adds all of the sections and complements the result, it should get zero if there is no error in the data during transmission or processing. This is true because of the rules in one's complement arithmetic.

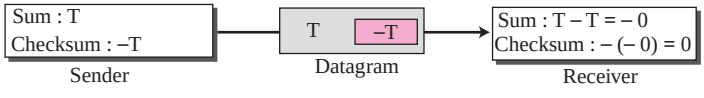
Assume that we get a number called *T* when we add all the sections in the sender. When we complement the number in one's complement arithmetic, we get the negative of the number. This means that if the sum of all sections is *T*, the checksum is $-T$.

Figure 7.22 *Checksum concept*



When the receiver receives the packet, it adds all the sections. It adds T and $-T$ which, in one's complement, is -0 (minus zero). When the result is complemented, -0 becomes 0. Thus if the final result is 0, the packet is accepted; otherwise, it is rejected (see Figure 7.23).

Figure 7.23 *Checksum in one's complement arithmetic*



Checksum in the IP Packet

The implementation of the checksum in the IP packet follows the same principles discussed above. First, the value of the checksum field is set to 0. Then, the entire header is divided into 16-bit sections and added together. The result (sum) is complemented and inserted into the checksum field.

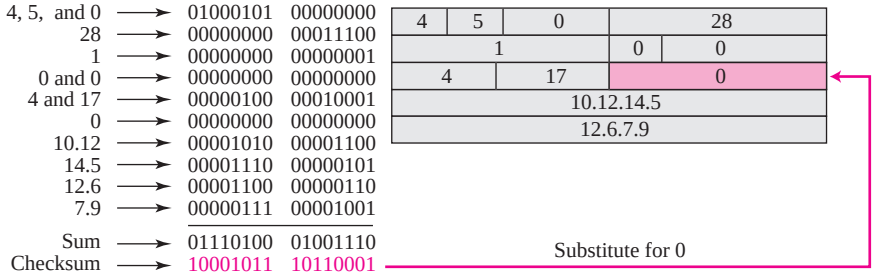
The checksum in the IP packet covers only the header, not the data. There are two good reasons for this. First, all higher-level protocols that encapsulate data in the IP datagram have a checksum field that covers the whole packet. Therefore, the checksum for the IP datagram does not have to check the encapsulated data. Second, the header of the IP packet changes with each visited router, but the data do not. So the checksum includes only the part that has changed. If the data were included, each router would have to recalculate the checksum for the whole packet, which means an increase in processing time.

Checksum in IP covers only the header, not the data.

Example 7.17

Figure 7.24 shows an example of a checksum calculation at the sender site for an IP header without options. The header is divided into 16-bit sections. All the sections are added and the sum is complemented. The result is inserted in the checksum field.

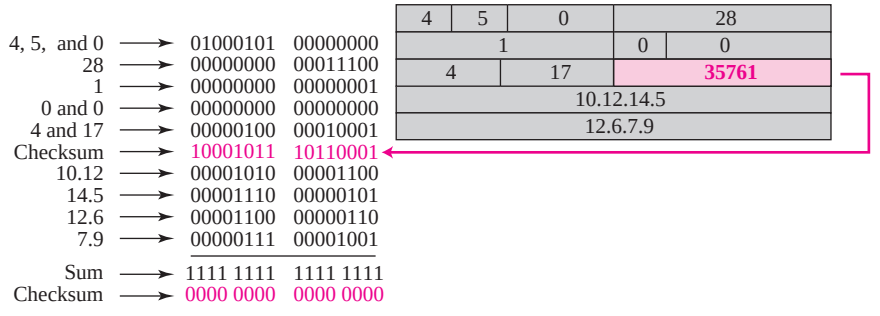
Figure 7.24 *Example of checksum calculation at the sender*



Example 7.18

Figure 7.25 shows the checking of checksum calculation at the receiver site (or intermediate router) assuming that no errors occurred in the header. The header is divided into 16-bit sections. All the sections are added and the sum is complemented. Since the result is 16 0s, the packet is accepted.

Figure 7.25 *Example of checksum calculation at the receiver*



7.6 IP OVER ATM

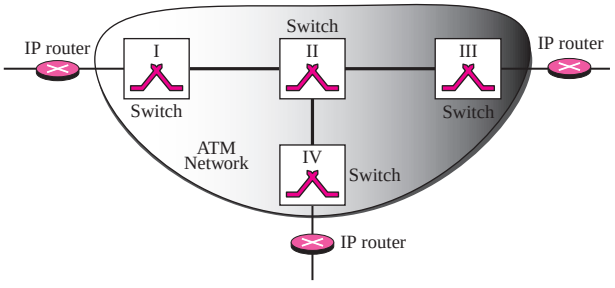
In the previous sections, we assumed that the underlying networks over which the IP datagrams are moving are either LANs or point-to-point WANs. In this section, we want to see how an IP datagram is moving through a switched WAN such as an ATM. We will see that there are similarities as well as differences. The IP packet is encapsulated in cells (not just one). An ATM network has its own definition for the physical address of a device. Binding between an IP address and a physical address is attained through a protocol called ATMARP (discussed in Chapter 8).

Appendix D gives an algorithm for checksum calculation.

ATM WANs

We discussed ATM WANs in Chapter 3. ATM, a cell-switched network, can be a highway for an IP datagram. Figure 7.26 shows how an ATM network can be used in the Internet.

Figure 7.26 *An ATM WAN in the Internet*



AAL Layer

In Chapter 3, we discussed different AAL layers and their applications. The only AAL used by the Internet is AAL5. It is sometimes called the *simple and efficient adaptation layer* (SEAL). AAL5 assumes that all cells created from one IP datagram belong to a single message. AAL5 therefore provides no addressing, sequencing, or other header information. Instead, only padding and a four-field trailer are added to the IP packet.

AAL5 accepts an IP packet of no more than 65,536 bytes and adds an 8-byte trailer as well as any padding required to ensure that the position of the trailer falls where the receiving equipment expects it (at the last 8 bytes of the last cell). Once the padding and trailer are in place, AAL5 passes the message in 48-byte segments to the ATM layer.

The AAL layer used by the IP protocol is AAL5.

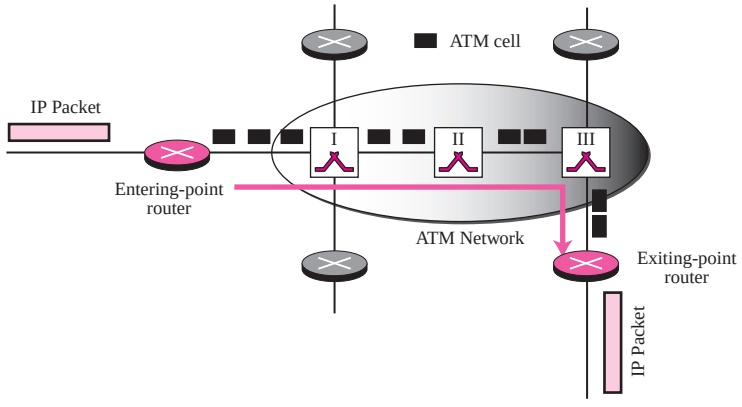
Why Use AAL5?

A question that frequently comes up is why do we use AAL5. Why can't we just encapsulate an IP packet in a cell? The answer is that it is more efficient to use AAL5. If an IP datagram is to be encapsulated in a cell, the data at the IP level must be $53 - 5 - 20 = 27$ bytes because a minimum of 20 bytes is needed for the IP header and 5 bytes is needed for the ATM header. The efficiency is 27/53, or almost 51 percent. By letting an IP datagram span over several cells, we are dividing the IP overhead (20 bytes) among those cells and increasing efficiency.

Routing the Cells

The ATM network creates a route between two routers. We call these routers entering-point and exiting-point routers. The cells start from the entering-point router and end at the exiting-point router as shown in Figure 7.27.

Figure 7.27 *Entering-point and exiting-point routers*



Addresses

Routing the cells from one specific entering-point router to one specific exiting-point router requires three types of addressing: IP addresses, physical addresses, and virtual circuit identifiers.

IP Addresses Each router connected to the ATM network has an IP address. Later we will see that the addresses may or may not have the same prefix. The IP address defines the router at the IP layer. It does not have anything to do with the ATM network.

Physical Addresses Each router (or any other device) connected to the ATM network has also a physical address. The physical address is associated with the ATM network and does not have anything to do with the Internet. The ATM Forum defines 20-byte addresses for ATM networks. Each address must be unique in a network and is defined by the network administrator. The physical addresses in an ATM network play the same role as the MAC addresses in a LAN. The physical addresses are used during connection establishment.

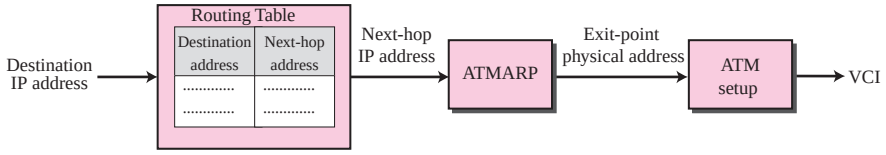
Virtual Circuit Identifiers The switches inside the ATM network route the cells based on the virtual circuit identifiers (VPIs and VCIs), as we discussed in Chapter 3. The virtual circuit identifiers are used during data transfer.

Address Binding

An ATM network needs virtual circuit identifiers to route the cells. The IP datagram contains only source and destination IP addresses. Virtual circuit identifiers must be determined from the destination IP address. Figure 7.28 shows how this is done. These are the steps:

1. The **entering-point router** receives an IP datagram. It uses the destination address and its routing table to find the IP address of the next router, the **exiting-point router.** This is exactly the same step followed when a datagram passes through a LAN.

Figure 7.28 *Address binding in IP over ATM*



2. The entering-point router uses the services of a protocol called ATMARP to find the physical address of the exiting-point router. ATMARP is similar to ARP (discussed in Chapter 8).
3. The virtual circuit identifiers are bound to the physical addresses as discussed in Chapter 3.

7.7 SECURITY

The IPv4 protocol, as well as the whole Internet, was started when the Internet users trusted each other. No security was provided for the IPv4 protocol. Today, however, the situation is different; the Internet is not secure any more. Although we discuss network security in general and IP security in particular in Chapters 29 and 30, we give a brief idea about the security issues in IP protocol and the solution.

Security Issues

There are three security issues that are particularly applicable to the IP protocol: packet sniffing, packet modification, and IP spoofing.

Packet Sniffing

An intruder may intercept an IP packet and make a copy of it. Packet sniffing is a passive attack, in which the attacker does not change the contents of the packet. This type of attack is very difficult to detect because the sender and the receiver may never know that the packet has been copied. Although packet sniffing cannot be stopped, *encryption* of the packet can make the attacker effort useless. The attacker may still sniff the packet, but it cannot find its contents.

Packet Modification

The second type of attack is to modify the packet. The attacker intercepts the packet, changes its contents, and sends the new packet to the receiver. The receiver believes that the packet is coming from the original sender. This type of attack can be detected using a *data integrity* mechanism. The receiver before opening and using the contents of the message can use this mechanism to make sure that the packet has not been changed during the transmission.

IP Spoofing

An attacker can masquerade as somebody else and create an IP packet that carries the source address of another computer. An attacker can send an IP packet to a bank

pretending that it is coming from one of the customers. This type of attack can be prevented using an *origin authentication* mechanism.

IPSec

The IP packets today can be protected from the previously mentioned attacks using a protocol called IPSec (IP Security). This protocol, which is used in conjunction with the IP protocol, creates a connection-oriented service between two entities in which they can exchange IP packets without worrying about the three attacks discussed before. We will discuss IPSec in detail in Chapter 30, it is enough to mention that IPSec provides the following four services:

Defining Algorithms and Keys

The two entities that want to create a secure channel between themselves can agree on some available algorithms and keys to be used for security purposes.

Packets Encryption

The packets exchanged between two parties can be encrypted for privacy using one of the encryption algorithms and a shared key agreed upon in the first step. This makes the packet sniffing attack useless.

Data Integrity

Data integrity guarantees that the packet is not modified during the transmission. If the received packet does not pass the data integrity test, it is discarded. This prevents the second attack, packet modification, described above.

Origin Authentication

IPsec can authenticate the origin of the packet to be sure that the packet is not created by an imposter. This can prevent IP spoofing attack as described above.

7.8 IP PACKAGE

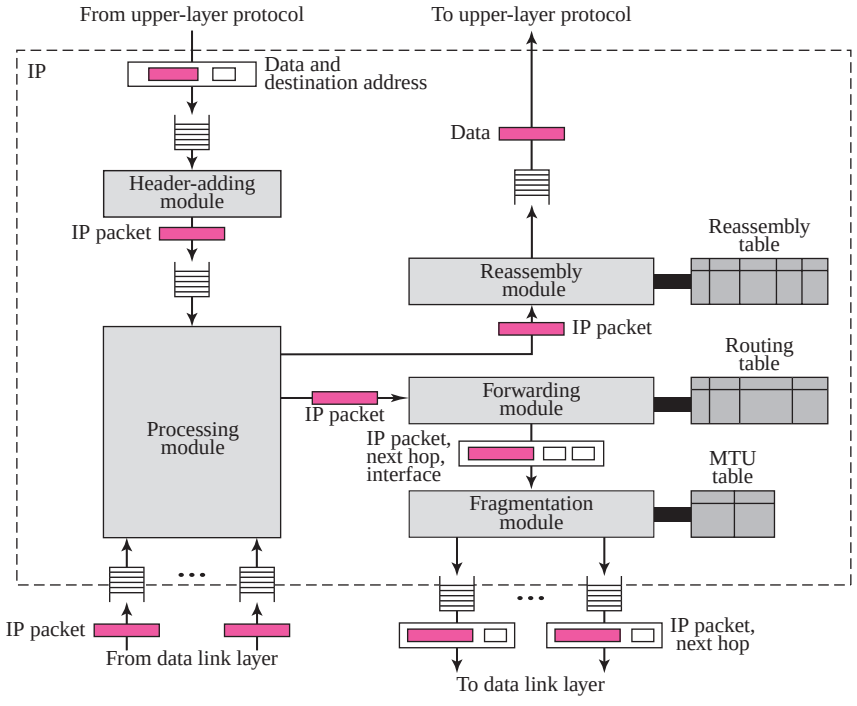
In this section, we present a simplified example of a hypothetical IP package. Our purpose is to show the relationships between the different concepts discussed in this chapter. Figure 7.29 shows eight components and their interactions.

Although IP supports several options, we have omitted option processing in our package to make it easier to understand at this level. In addition, we have sacrificed efficiency for the sake of simplicity.

We can say that the IP package involves eight components: a header-adding module, a processing module, a forwarding module, a fragmentation module, a reassembly module, a routing table, an MTU table, and a reassembly table. In addition, the package includes input and output queues.

The package receives a packet, either from the data link layer or from a higher-level protocol. If the packet comes from an upper-layer protocol, it is delivered to the data link layer for transmission (unless it has a loopback address of 127.X.Y.Z). If the packet comes from the data link layer, it is either delivered to the data link layer for forwarding (in a router) or it is delivered to a higher-layer protocol if the destination IP

Figure 7.29 *IP components*



address of the packet is the same as the station IP address. Note that we used multiple queues to and from the data link layer because a router is multihomed.

Header-Adding Module

The **header-adding module** (Table 7.3) receives data from an upper-layer protocol along with the destination IP address. It encapsulates the data in an IP datagram by adding the IP header.

Table 7.3 *Adding module*

```
1  IP_Adding_Module (data, destination_address)
2  {
3       Encapsulate data in an IP datagram
4       Calculate checksum and insert it in the checksum field
5       Send data to the corresponding queue
6       Return
7  }
```

Processing Module

The **processing module** (Table 7.4) is the heart of the IP package. In our package, the processing module receives a datagram from an interface or from the header-adding module. It treats both cases the same. A datagram must be processed and routed regardless of where it comes from.

Table 7.4 *Processing module*

```
IP_Processing_Module (Datagram)
{
      Remove one datagram from one of the input queues.
      If (destination address matches a local address)
      {
          Send the datagram to the reassembly module.
          Return.
      }
      If (machine is a router)
      {
          Decrement TTL.
      }
      If (TTL less than or equal to zero)
      {
          Discard the datagram.
          Send an ICMP error message.
          Return.
      }
      Send the datagram to the forwarding module.
      Return.
}
```

The processing module first checks to see if the datagram has reached its final destination. In this case, the packet is sent to the reassembly module.

If the node is a router, it decrements the time-to-live (TTL) field by one. If this value is less than or equal to zero, the datagram is discarded and an ICMP message (see Chapter 9) is sent to the original sender. If the value of TTL is greater than zero after decrement, the processing module sends the datagram to the forwarding module.

Queues

Our package uses two types of queues: input queues and output queues. The **input queues** store the datagrams coming from the data link layer or the upper-layer protocols. The **output queues** store the datagrams going to the data link layer or the upper-layer protocols. The processing module dequeues (removes) the datagrams from the input queues. The fragmentation and reassembly modules enqueue (add) the datagrams into the output queues.

Routing Table

We discussed the routing table in Chapter 6. The routing table is used by the forwarding module to determine the next-hop address of the packet.

Forwarding Module

We discussed the **forwarding module** in Chapter 6. The forwarding module receives an IP packet from the processing module. If the packet is to be forwarded, it is passed to this module. The module finds the IP address of the next station along with the interface number to which the packet should be sent. It then sends the packet with this information to the fragmentation module.

MTU Table

The MTU table is used by the fragmentation module to find the **maximum transfer unit (MTU)** of a particular interface. It can have only two columns: interface and MTU.

Fragmentation Module

In our package, the **fragmentation module** (Table 7.5) receives an IP datagram from the forwarding module. The forwarding module gives the IP datagram, the IP address of the next station (either the final destination in a direct delivery or the next router in an indirect delivery), and the interface number through which the datagram is sent out.

The fragmentation module consults the MTU table to find the MTU for the specific interface number. If the length of the datagram is larger than the MTU, the fragmentation module fragments the datagram, adds a header to each fragment, and sends them to the ARP package (see Chapter 8) for address resolution and delivery.

Table 7.5 *Fragmentation module*

```
IP_Fragmentation_Module (datagram)
{
     Extract the size of datagram
     If (size > MTU of the corresponding network)
     {
          If (D bit is set)
          {
               Discard datagram
               Send an ICMP error message
               return
          }
          Else
          {
               Calculate maximum size
               Divide the segment into fragments
               Add header to each fragment
               Add required options to each fragment
```

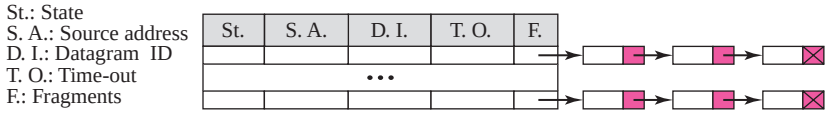
Table 7.5 *Fragmentation module (continued)*

```
                Send fragment
                return
            }
        }
        Else
        {
            Send the datagram
        }
        Return.
}
```

Reassembly Table

The **reassembly table** is used by the reassembly module. In our package, the reassembly table has five fields: state, source IP address, datagram ID, time-out, and fragments (see Figure 7.30).

Figure 7.30 *Reassembly table*



The value of the state field can be either FREE or IN-USE. The IP address field defines the source IP address of the datagram. The datagram ID is a number that uniquely defines a datagram and all of the fragments belonging to that datagram. The time-out is a predetermined amount of time in which all fragments must arrive. Finally, the fragments field is a pointer to a linked list of fragments.

Reassembly Module

The **reassembly module** (Table 7.6) receives, from the processing module, those datagram fragments that have arrived at their final destinations. In our package, the reassembly module treats an unfragmented datagram as a fragment belonging to a datagram with only one fragment.

Because the IP protocol is a connectionless protocol, there is no guarantee that the fragments arrive in order. Besides, the fragments from one datagram can be intermixed with fragments from another datagram. To keep track of these situations, the module uses a reassembly table with associated linked lists, as we described earlier.

The job of the reassembly module is to find the datagram to which a fragment belongs, to order the fragments belonging to the same datagram, and reassemble all fragments of a datagram when all have arrived. If the established time-out has expired and any fragment is missing, the module discards the fragments.

Table 7.6 *Reassembly module*

```
IP_Reassembly_Module (datagram)
{
     If (offset value = 0 AND M = 0)
     {
          Send datagram to the appropriate queue
          Return
     }
     Search the reassembly table for the entry
     If (entry not found)
     {
          Create a new entry
     }
     Insert datagram into the linked list
     If (all fragments have arrived)
     {
          Reassemble the fragment
          Deliver the fragment to upper-layer protocol
          return
     }
     Else
     {
          If (time-out expired)
          {
               Discard all fragments
               Send an ICMP error message
          }
     }
     Return.
}
```

7.9 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], [Kur & Ros 08], and [Gar & Vid 04].

RFCs

Several RFCs discuss the IPv4 protocol including: RFC 791, RFC 815, RFC 894, RFC 1122, RFC 2474, and RFC 2475.

7.10 KEY TERMS

best-effort delivery
checksum
codepoint
datagram
destination address
differentiated services
end-of-option option
entering-point router
exiting-point router
forwarding module
fragmentation
fragmentation module
fragmentation offset
header length
header-adding module
input queue
Internet Protocol (IP)
length field
loose-source-route option
maximum transfer unit (MTU)
no-operation option
output queue
ping
pointer field
precedence
processing module
reassembly module
reassembly table
record-route option
service type
source address
strict-source-route option
time to live
timestamp option
traceroute
type field
type of service (TOS)
value field

7.11 SUMMARY

- IP is an unreliable connectionless protocol responsible for source-to-destination delivery. Packets in the IP layer are called datagrams.
- The MTU is the maximum number of bytes that a data link protocol can encapsulate. MTUs vary from protocol to protocol. Fragmentation is the division of a datagram into smaller units to accommodate the MTU of a data link protocol.
- The IP datagram header consists of a fixed, 20-byte section and a variable options section with a maximum of 40 bytes. The options section of the IP header is used for network testing and debugging. The six IP options each have a specific function.
- The error detection method used by IP is the checksum. The checksum, however, covers only the header, but not the data. The checksum uses one's complement arithmetic to add equal-size sections of the IP header. The complemented result is stored in the checksum field. The receiver also uses one's complement arithmetic to check the header.

- IP over ATM uses AAL5 layer in an ATM network. An ATM network creates a route between an entering-point router and an exiting-point router. The next-hop address of an IP packet can be mapped to a physical address of an exiting-point router using ATMARP.
- An IP package can consist of the following: a header-adding module, a processing module, a forwarding module, a fragmentation module, a reassembly module, a routing table, an MTU table, and a reassembly table.

7.12 PRACTICE SET

Exercises

1. Which fields of the IP header change from router to router?
2. Calculate the HLEN value if the total length is 1200 bytes, 1176 of which is data from the upper layer.
3. Table 7.3 lists the MTUs for many different protocols. The MTUs range from 296 to 65,535. What would be the advantages of having a large MTU? What would be the advantages of having a small MTU?
4. Given a fragmented datagram with an offset of 120, how can you determine the first and last byte number?
5. An IP datagram must go through router 128.46.10.5. There are no other restrictions on the routers to be visited. Draw the IP options with their values.
6. What is the maximum number of routers that can be recorded if the timestamp option has a flag value of 1? Why?
7. Can the value of the header length in an IP packet be less than 5? When is it exactly 5?
8. The value of HLEN in an IP datagram is 7. How many option bytes are present?
9. The size of the option field of an IP datagram is 20 bytes. What is the value of HLEN? What is the value in binary?
10. The value of the total length field in an IP datagram is 36 and the value of the header length field is 5. How many bytes of data is the packet carrying?
11. A datagram is carrying 1024 bytes of data. If there is no option information, what is the value of the header length field? What is the value of the total length field?
12. A host is sending 100 datagrams to another host. If the identification number of the first datagram is 1024, what is the identification number of the last?
13. An IP datagram arrives with fragmentation offset of 0 and an *M* bit (more fragment bit) of 0. Is this a first fragment, middle fragment, or last fragment?
14. An IP fragment has arrived with an offset value of 100. How many bytes of data were originally sent by the source before the data in this fragment?
15. An IP datagram has arrived with the following information in the header (in hexadecimal):

45 00 00 54 00 03 00 00 20 06 00 00 7C 4E 03 02 B4 0E 0F 02

a. Are there any options?
b. Is the packet fragmented?
c. What is the size of the data?
d. Is a checksum used?
e. How many more routers can the packet travel to?
f. What is the identification number of the packet?
g. What is the type of service?

16. In a datagram, the M bit is zero, the value of HLEN is 5, the value of total length is 200, and the offset value is 200. What is the number of the first byte and number of the last byte in this datagram? Is this the last fragment, the first fragment, or a middle fragment?

Research Activities

17. Use the *ping* utility with the -R option to check the routing of a packet to a destination. Interpret the result.

18. Use the *traceroute* utility with the -g option to implement the loose source route option. Choose some routers between the source and destination. Interpret the result and find if all defined routers have been visited.

19. Use the *traceroute* utility with the -G option to implement the strict source route option. Choose some routers between the source and destination. Interpret the result and find if all defined routers have been visited and no undefined router visited.