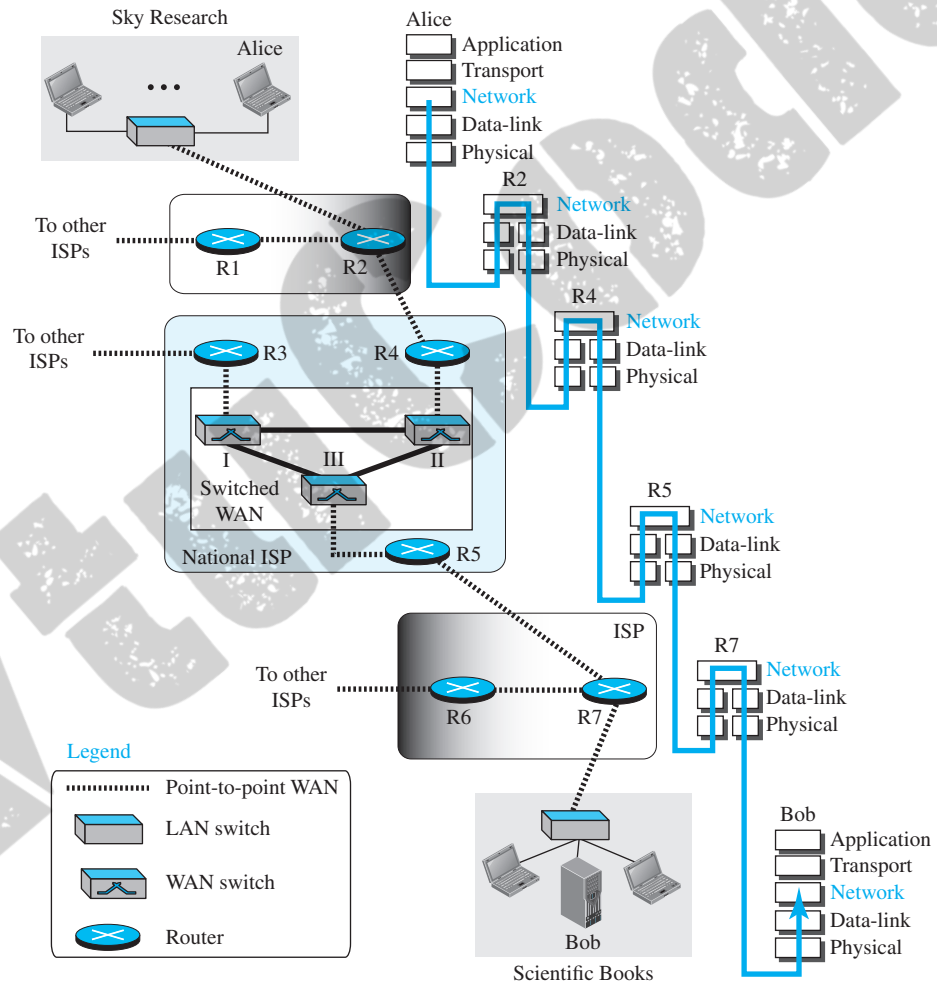


MODULE 3

18.1 NETWORK-LAYER SERVICES

Before discussing the network layer in the Internet today, let's briefly discuss the network-layer services that, in general, are expected from a network-layer protocol. Figure 18.1 shows the communication between Alice and Bob at the network layer. This is the same scenario we used in Chapters 3 and 9 to show the communication at the physical and the data-link layers, respectively.

Figure 18.1 *Communication at the network layer*



The figure shows that the Internet is made of many networks (or links) connected 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 think about the connecting devices (routers or switches) that connect the LANs and WANs.

As the figure shows, the network layer is involved at the source host, destination host, and all routers in the path (R2, R4, R5, and R7). At the source host (Alice), the network layer accepts a packet from a transport layer, encapsulates the packet in a datagram, and delivers the packet to the data-link layer. At the destination host (Bob), the datagram is decapsulated, and the packet is extracted and delivered to the corresponding transport layer. Although the source and destination hosts are involved in all five layers of the TCP/IP suite, the routers use three layers if they are routing packets only; however, they may need the transport and application layers for control purposes. A router in the path is normally shown with two data-link layers and two physical layers, because it receives a packet from one network and delivers it to another network.

18.1.1 Packetizing

The first duty of the network layer is definitely **packetizing**: encapsulating the payload (data received from upper layer) in a network-layer packet at the source and decapsulating the payload from the network-layer packet at the destination. In other words, one duty of the network layer is to carry a payload from the source to the destination without changing it or using it. The network layer is doing the service of a carrier such as the postal office, which is responsible for delivery of packages from a sender to a receiver without changing or using the contents.

The source host receives the payload from an upper-layer protocol, adds a header that contains the source and destination addresses and some other information that is required by the network-layer protocol (as discussed later) and delivers the packet to the data-link layer. The source is not allowed to change the content of the payload unless it is too large for delivery and needs to be fragmented.

The destination host receives the network-layer packet from its data-link layer, decapsulates the packet, and delivers the payload to the corresponding upper-layer protocol. If the packet is fragmented at the source or at routers along the path, the network layer is responsible for waiting until all fragments arrive, reassembling them, and delivering them to the upper-layer protocol.

The routers in the path are not allowed to decapsulate the packets they received unless the packets need to be fragmented. The routers are not allowed to change source and destination addresses either. They just inspect the addresses for the purpose of forwarding the packet to the next network on the path. However, if a packet is fragmented, the header needs to be copied to all fragments and some changes are needed, as we discuss in detail later.

18.1.2 Routing and Forwarding

Other duties of the network layer, which are as important as the first, are routing and forwarding, which are directly related to each other.

Routing

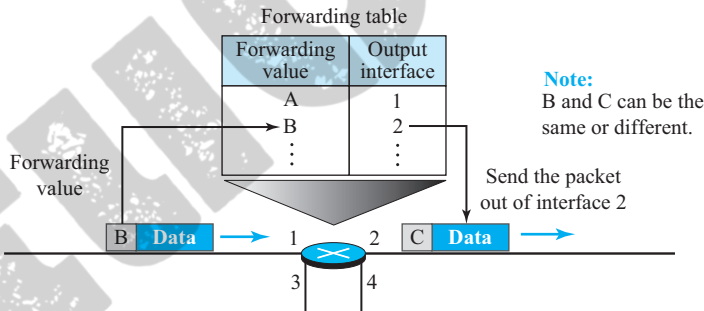
The network layer is responsible for routing the packet from its source to the destination. A physical network is a combination of networks (LANs and WANs) and routers

that connect them. This means that there is more than one route from the source to the destination. The network layer is responsible for finding the best one among these possible routes. The network layer needs to have some specific strategies for defining the best route. In the Internet today, this is done by running some routing protocols to help the routers coordinate their knowledge about the neighborhood and to come up with consistent tables to be used when a packet arrives. The routing protocols, which we discuss in Chapters 20 and 21, should be run before any communication occurs.

Forwarding

If routing is applying strategies and running some routing protocols to create the decision-making tables for each router, *forwarding* can be defined as the action applied by each router when a packet arrives at one of its interfaces. The decision-making table a router normally uses for applying this action is sometimes called the *forwarding table* and sometimes the *routing table*. When a router receives a packet from one of its attached networks, it needs to forward the packet to another attached network (in unicast routing) or to some attached networks (in multicast routing). To make this decision, the router uses a piece of information in the packet header, which can be the destination address or a label, to find the corresponding output interface number in the forwarding table. Figure 18.2 shows the idea of the forwarding process in a router.

Figure 18.2 Forwarding process



18.1.3 Other Services

Let us briefly discuss other services expected from the network layer.

Error Control

In Chapter 10, we discussed error detection and correction. Although error control also can be implemented in the network layer, the designers of the network layer in the Internet ignored this issue for the data being carried by the network layer. One reason for this decision is the fact that the packet in the network layer may be fragmented at each router, which makes error checking at this layer inefficient.

The designers of the network layer, however, have added a checksum field to the datagram to control any corruption in the header, but not in the whole datagram. This checksum may prevent any changes or corruptions in the header of the datagram.

We need to mention that although the network layer in the Internet does not directly provide error control, the Internet uses an auxiliary 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 Chapter 19.

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 that 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.

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 they are ready and do not have to consume the data as fast as it is received. Third, 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 efficient.

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 is 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 where the system collapses and no datagrams are delivered. We discuss congestion control at the network layer later in the chapter although it is not implemented in the Internet.

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 by providing better quality of service to support these applications. However, to keep the network layer untouched, these provisions are mostly implemented in the upper layer. We discuss this issue in Chapter 30 after we have discussed multimedia.

Security

Another issue related to 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 universities for 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 32.

18.2 PACKET SWITCHING

From the discussion of routing and forwarding in the previous section, we infer that a kind of *switching* occurs at the network layer. A router, in fact, is a switch that creates a connection between an input port and an output port (or a set of output ports), just as an electrical switch connects the input to the output to let electricity flow.

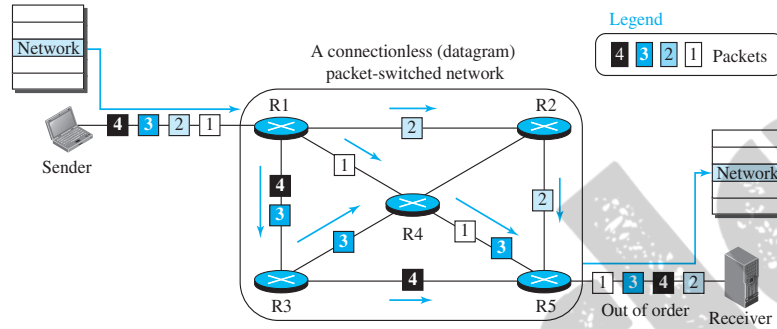
Although in data communication switching techniques are divided into two broad categories, circuit switching and packet switching, only packet switching is used at the network layer because the unit of data at this layer is a packet. Circuit switching is mostly used at the physical layer; the electrical switch mentioned earlier is a kind of circuit switch. We discussed circuit switching in Chapter 8; we discuss packet switching in this chapter.

At the network layer, 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-switched 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.

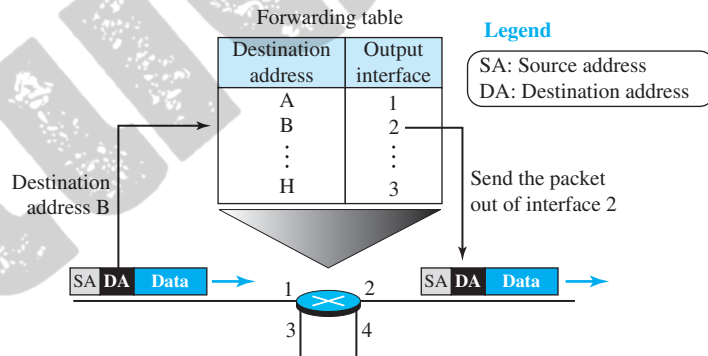
18.2.1 Datagram Approach: Connectionless Service

When the Internet started, to make it simple, 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 idea was that the network layer is only responsible for delivery of packets from the source to the destination. In this approach, the packets in a message may or may not travel the same path to their destination. Figure 18.3 shows the idea.

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

Figure 18.3 A connectionless packet-switched network

Each packet is routed based on the information contained in its header: source and destination addresses. 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 18.4 shows the forwarding process in a router in this case. We have used symbolic addresses such as A and B.

Figure 18.4 Forwarding process in a router when used in a connectionless network

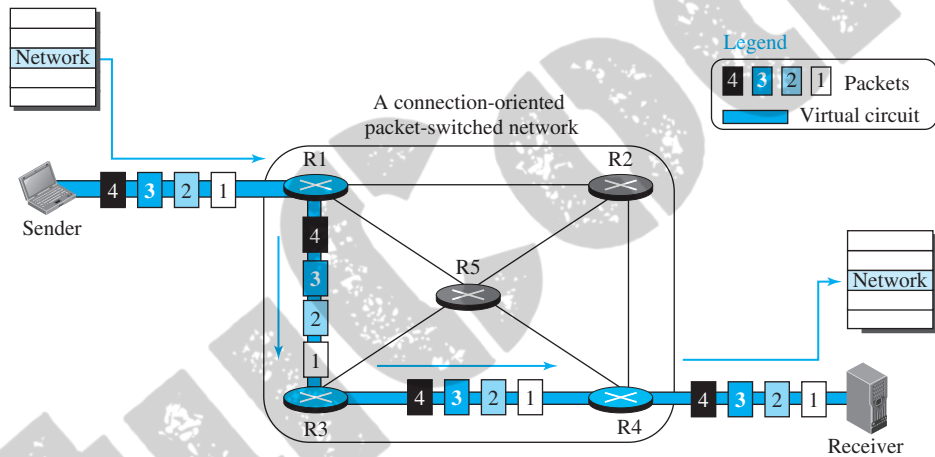
In the datagram approach, the forwarding decision is based on the destination address of the packet.

18.2.2 Virtual-Circuit Approach: Connection-Oriented Service

In a connection-oriented service (also called *virtual-circuit approach*), there is a relationship 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 all 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. Shortly, we will 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 unnecessary during the data transfer phase, parts of the Internet at the network layer still keep these addresses. 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 18.5 shows the concept of connection-oriented service.

Figure 18.5 A virtual-circuit packet-switched network

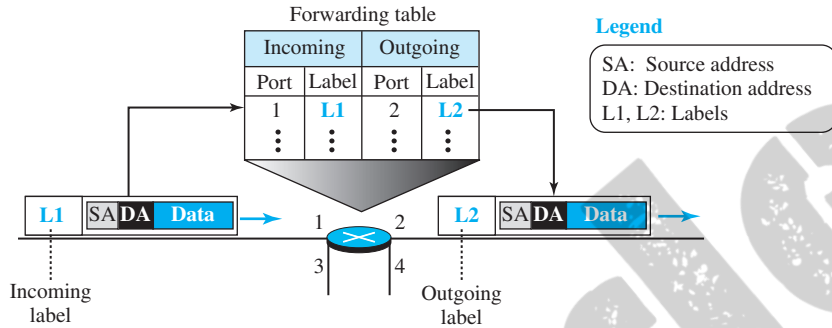


Each packet is forwarded based on the label in the packet. To follow the idea of connection-oriented design to be used in the Internet, we assume that the packet has a label when it reaches the router. Figure 18.6 shows the idea. In this case, the forwarding decision is based on the value of the label, 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 are 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.

Setup Phase

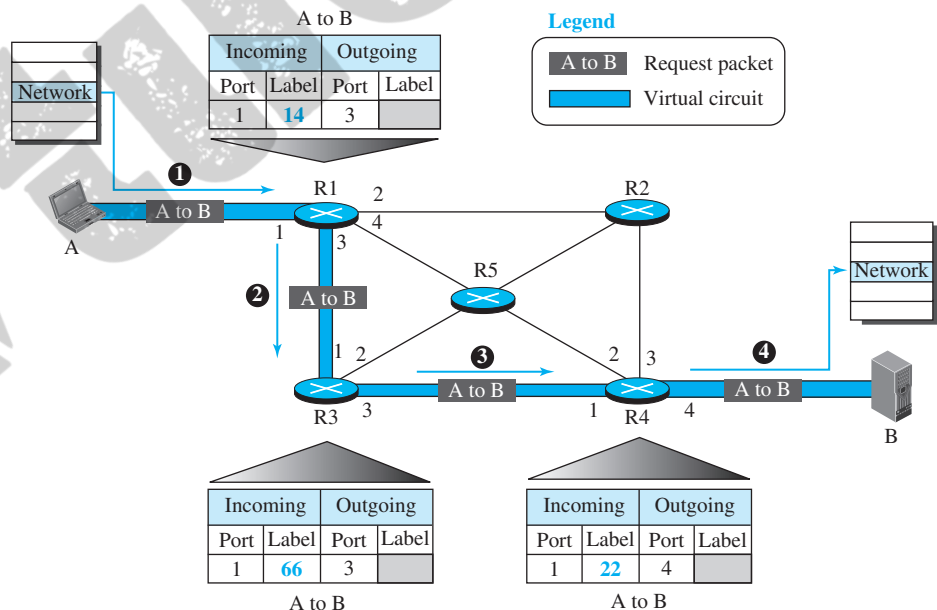
In the setup phase, a router creates an entry for a virtual circuit. For example, suppose source A needs to create a virtual circuit to destination B. Two auxiliary packets need to be exchanged between the sender and the receiver: the request packet and the acknowledgment packet.

Figure 18.6 Forwarding process in a router when used in a virtual-circuit network

In the virtual-circuit approach, the forwarding decision is based on the label of the packet.

Request packet

A request packet is sent from the source to the destination. This auxiliary packet carries the source and destination addresses. Figure 18.7 shows the process.

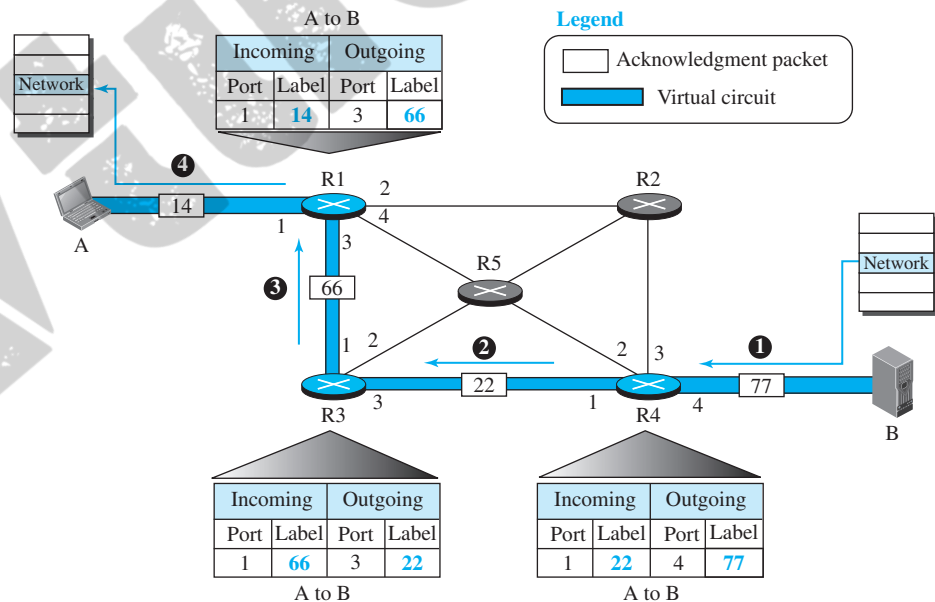
Figure 18.7 Sending request packet in a virtual-circuit network

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 covered later. 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 (3).
4. Router R4 receives the setup request packet. Again, three columns are completed: incoming port (1), incoming label (22), and outgoing port (4).
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, as shown in Figure 18.8. This label lets the destination know that the packets come from A, and not from other sources.

Acknowledgment Packet

A special packet, called the acknowledgment packet, completes the entries in the switching tables. Figure 18.8 shows the process.

Figure 18.8 Sending acknowledgments in a virtual-circuit network

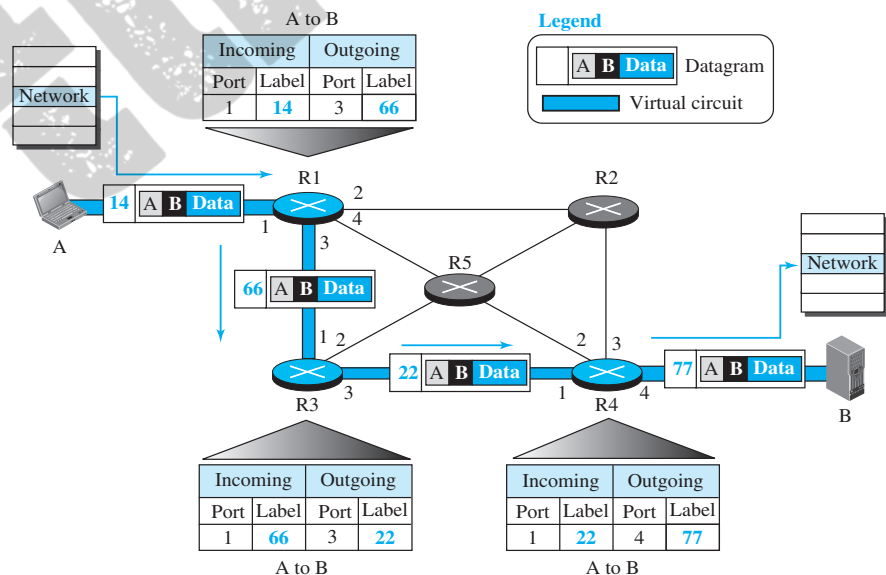


1. The destination sends an acknowledgment to router R4. The acknowledgment carries the global source and destination addresses so the router knows which entry in the table is to be completed. The packet also carries label 77, chosen by the destination as the incoming label for packets from A. Router 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 forwarding table for a specific virtual circuit, then the network-layer packets belonging to one message can be sent one after another. In Figure 18.9, we show the flow of a 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

Figure 18.9 Flow of one packet in an established virtual circuit



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, and the packets arrive in order at the destination.

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 entries from their tables.

18.4 IPV4 ADDRESSES

The identifier used in the IP layer of the TCP/IP protocol suite to identify the connection of each device 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. The IP address is the address of the connection, not the

host or the router, because if the device is moved to another network, the IP address may be changed.

IPv4 addresses are unique in the sense that each address defines one, and only one, connection to the Internet. If a device has two connections to the Internet, via two networks, it has two IPv4 addresses. 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.

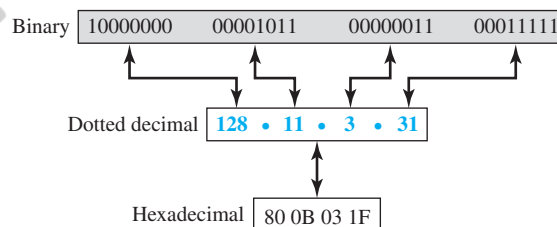
18.4.1 Address Space

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). If there were no restrictions, more than 4 billion devices could be connected to the Internet.

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). In *binary notation*, an IPv4 address is displayed as 32 bits. To make the address more readable, one or more spaces are usually inserted between each octet (8 bits). Each octet is often referred to as a byte. To make the IPv4 address more compact and easier to read, it is usually written in decimal form with a decimal point (dot) separating the bytes. This format is referred to as *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. 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. Figure 18.16 shows an IP address in the three discussed notations.

Figure 18.16 Three different notations in IPv4 addressing



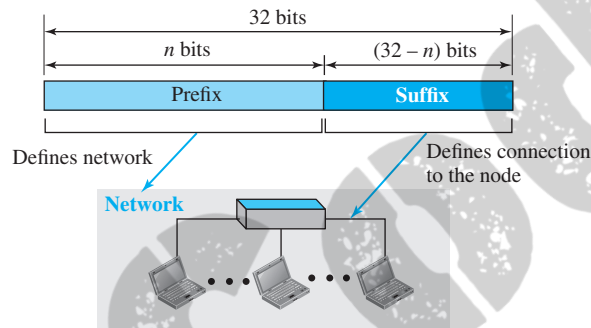
Hierarchy in Addressing

In any communication network that involves delivery, such as a telephone network or a postal network, the addressing system is hierarchical. In a postal network, the postal address (mailing address) includes the country, state, city, street, house number, and the

name of the mail recipient. Similarly, a telephone number is divided into the country code, area code, local exchange, and the connection.

A 32-bit IPv4 address is also hierarchical, but divided only into two parts. The first part of the address, called the *prefix*, defines the network; the second part of the address, called the *suffix*, defines the node (connection of a device to the Internet). Figure 18.17 shows the prefix and suffix of a 32-bit IPv4 address. The prefix length is n bits and the suffix length is $(32 - n)$ bits.

Figure 18.17 Hierarchy in addressing



A prefix can be fixed length or variable length. The network identifier in the IPv4 was first designed as a fixed-length prefix. This scheme, which is now obsolete, is referred to as classful addressing. The new scheme, which is referred to as classless addressing, uses a variable-length network prefix. First, we briefly discuss classful addressing; then we concentrate on classless addressing.

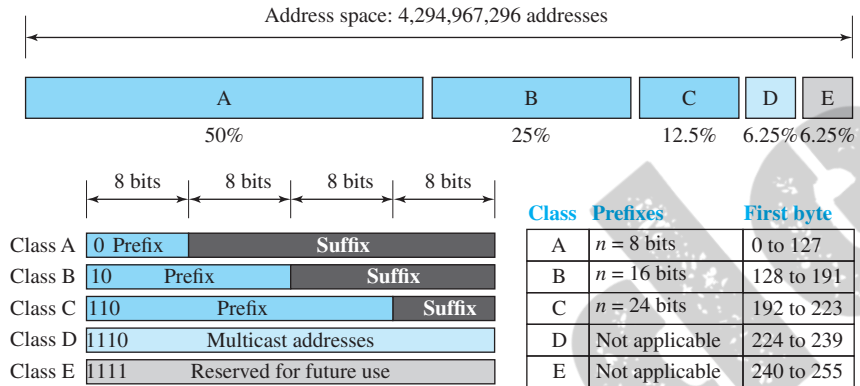
18.4.2 Classful Addressing

When the Internet started, an IPv4 address was designed with a fixed-length prefix, but to accommodate both small and large networks, three fixed-length prefixes were designed instead of one ($n = 8$, $n = 16$, and $n = 24$). The whole address space was divided into five classes (class A, B, C, D, and E), as shown in Figure 18.18. This scheme is referred to as **classful addressing**. Although classful addressing belongs to the past, it helps us to understand classless addressing, discussed later.

In class A, the network length is 8 bits, but since the first bit, which is 0, defines the class, we can have only seven bits as the network identifier. This means there are only $2^7 = 128$ networks in the world that can have a class A address.

In class B, the network length is 16 bits, but since the first two bits, which are $(10)_2$, define the class, we can have only 14 bits as the network identifier. This means there are only $2^{14} = 16,384$ networks in the world that can have a class B address.

All addresses that start with $(110)_2$ belong to class C. In class C, the network length is 24 bits, but since three bits define the class, we can have only 21 bits as the network identifier. This means there are $2^{21} = 2,097,152$ networks in the world that can have a class C address.

Figure 18.18 Occupation of the address space in classful addressing

Class D is not divided into prefix and suffix. It is used for multicast addresses. All addresses that start with 1111 in binary belong to class E. As in Class D, Class E is not divided into prefix and suffix and is used as reserve.

Address Depletion

The reason that classful addressing has become obsolete is address depletion. Since the addresses were not distributed properly, the Internet was faced with the problem of the addresses being rapidly used up, resulting in no more addresses available for organizations and individuals that needed to be connected to the Internet. To understand the problem, let us think about class A. This class can be assigned to only 128 organizations in the world, but each organization needs to have a single network (seen by the rest of the world) with 16,777,216 nodes (computers in this single network). Since there may be only a few organizations that are this large, most of the addresses in this class were wasted (unused). Class B addresses were designed for midsize organizations, but many of the addresses in this class also remained unused. Class C addresses have a completely different flaw in design. The number of addresses that can be used in each network (256) was so small that most companies were not comfortable using a block in this address class. Class E addresses were almost never used, wasting the whole class.

Subnetting and Supernetting

To alleviate address depletion, two strategies were proposed and, to some extent, implemented: subnetting and supernetting. In subnetting, a class A or class B block is divided into several subnets. Each subnet has a larger prefix length than the original network. For example, if a network in class A is divided into four subnets, each subnet has a prefix of $n_{\text{sub}} = 10$. At the same time, if all of the addresses in a network are not used, subnetting allows the addresses to be divided among several organizations. This idea did not work because most large organizations were not happy about dividing the block and giving some of the unused addresses to smaller organizations.

While subnetting was devised to divide a large block into smaller ones, supernetting was devised to combine several class C blocks into a larger block to be attractive to

organizations that need more than the 256 addresses available in a class C block. This idea did not work either because it makes the routing of packets more difficult.

Advantage of Classful Addressing

Although classful addressing had several problems and became obsolete, it had one advantage: Given an address, we can easily find the class of the address and, since the prefix length for each class is fixed, we can find the prefix length immediately. In other words, the prefix length in classful addressing is inherent in the address; no extra information is needed to extract the prefix and the suffix.

18.4.3 Classless Addressing

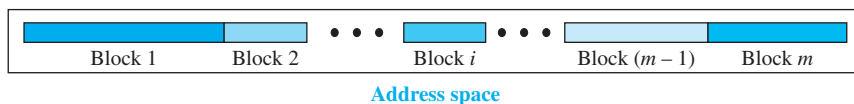
Subnetting and supernetting in classful addressing did not really solve the address depletion problem. With the growth of the Internet, it was clear that a larger address space was needed as a long-term solution. The larger address space, however, requires that the length of IP addresses also be increased, 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 (discussed later), 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 electronic mail) 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.

In classless addressing, the whole address space is divided into variable length blocks. The prefix in an address defines the block (network); the suffix defines the node (device). Theoretically, we can have a block of $2^0, 2^1, 2^2, \dots, 2^{32}$ addresses. One of the restrictions, 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 18.19 shows the division of the whole address space into nonoverlapping blocks.

Figure 18.19 Variable-length blocks in classless addressing



Unlike classful addressing, the prefix length in classless addressing is variable. We can have a prefix length that ranges from 0 to 32. The size of the network is inversely proportional to the length of the prefix. A small prefix means a larger network; a large prefix means a smaller network.

We need to emphasize that the idea of classless addressing can be easily applied to classful addressing. An address in class A can be thought of as a classless address in which the prefix length is 8. An address in class B can be thought of as a classless address in which the prefix is 16, and so on. In other words, classful addressing is a special case of classless addressing.

Prefix Length: Slash Notation

The first question that we need to answer in classless addressing is how to find the prefix length if an address is given. Since the prefix length is not inherent in the address, we need to separately give the length of the prefix. 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* and formally as **classless interdomain routing** or **CIDR** (pronounced cider) strategy. An address in classless addressing can then be represented as shown in Figure 18.20.

Figure 18.20 Slash notation (CIDR)



Examples:

12.24.76.8/8
23.14.67.92/12
220.8.24.255/25

In other words, an address in classless addressing does not, per se, define the block or network to which the address belongs; we need to give the prefix length also.

Extracting Information from an Address

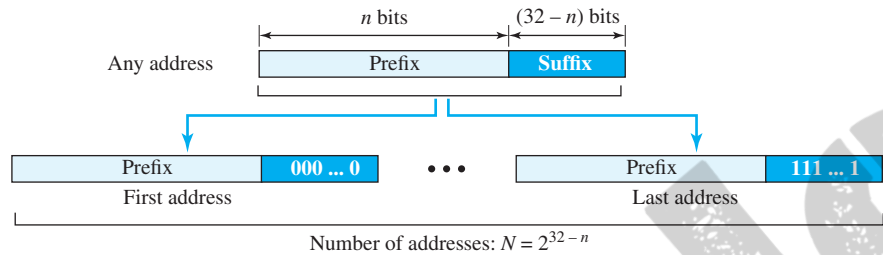
Given any address in the block, we normally like to know three pieces of information about the block to which the address belongs: the number of addresses, the first address in the block, and the last address. Since the value of prefix length, n , is given, we can easily find these three pieces of information, as shown in Figure 18.21.

1. The number of addresses in the block is found as $N = 2^{32-n}$.
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.

Example 18.1

A classless address is given as 167.199.170.82/27. We can find the above three pieces of information as follows. The number of addresses in the network is $2^{32-n} = 2^5 = 32$ addresses.

Figure 18.21 Information extraction in classless addressing



The first address can be found by keeping the first 27 bits and changing the rest of the bits to 0s.

Address: 167.199.170.82/27 10100111 11000111 10101010 01010010
 First address: 167.199.170.64/27 10100111 11000111 10101010 01000000

The last address can be found by keeping the first 27 bits and changing the rest of the bits to 1s.

Address: 167.199.170.82/27 10100111 11000111 10101010 01011111
 Last address: 167.199.170.95/27 10100111 11000111 10101010 01011111

Address Mask

Another way to find the first and last addresses in the block is to use the address mask. The address mask is a 32-bit number in which the n leftmost bits are set to 1s and the rest of the bits ($32 - n$) are set to 0s. A computer can easily find the address mask because it is the complement of $(2^{32-n} - 1)$. The reason for defining a mask in this way is that it can be used by a computer program to extract the information in a block, using the three bit-wise operations NOT, AND, and OR.

- 1. The number of addresses in the block $N = \text{NOT}(\text{mask}) + 1$.
- 2. The first address in the block = (Any address in the block) **AND** (mask).
- 3. The last address in the block = (Any address in the block) **OR** [(**NOT** (mask))].

Example 18.2

We repeat Example 18.1 using the mask. The mask in dotted-decimal notation is 256.256.256.224. The AND, OR, and NOT operations can be applied to individual bytes using calculators and applets at the book website.

Number of addresses in the block: $N = \text{NOT}(\text{mask}) + 1 = 0.0.0.31 + 1 = 32$ addresses
 First address: First = (address) **AND** (mask) = 167.199.170.82
 Last address: Last = (address) **OR** (**NOT** mask) = 167.199.170.255

Example 18.3

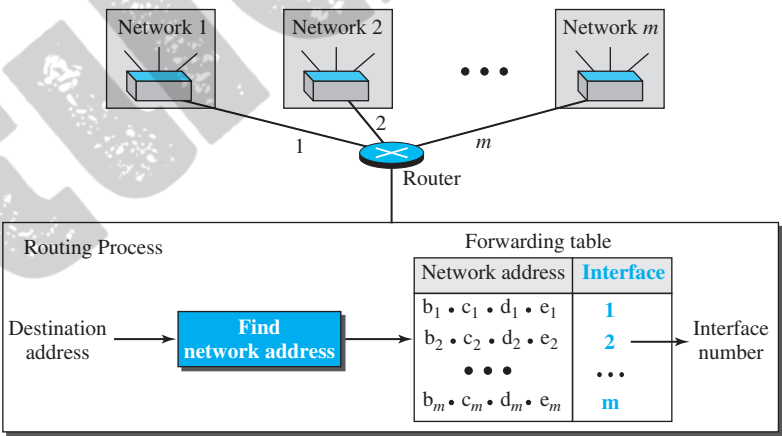
In classless addressing, an address cannot per se define the block the address belongs to. For example, the address 230.8.24.56 can belong to many blocks. Some of them are shown below with the value of the prefix associated with that block.

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 Address

The above examples show that, given any address, we can find all information about the block. The first address, the **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: 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 later. Figure 18.22 shows the idea. After the network address has been

Figure 18.22 Network address



found, the router consults its forwarding 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.

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 Numbers (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, two restrictions need to be applied to the allocated block.

1. The number of requested addresses, N , needs to be a power of 2. The reason is that $N = 2^{32-n}$ or $n = 32 - \log_2 N$. If N is not a power of 2, we cannot have an integer value for n .
2. The requested block needs to be allocated where there is an adequate number of contiguous addresses available in the address space. However, there is a restriction on choosing the first address in the block. The first address needs to be divisible by the number of addresses in the block. The reason is that the first address needs to be the prefix followed by $(32 - n)$ number of 0s. The decimal value of the first address is then

$$\text{first address} = (\text{prefix in decimal}) \times 2^{32-n} = (\text{prefix in decimal}) \times N.$$

Example 18.4

An ISP has requested a block of 1000 addresses. Since 1000 is not a power of 2, 1024 addresses are granted. The prefix length is calculated as $n = 32 - \log_2 1024 = 22$. An available block, 18.14.12.0/22, is granted to the ISP. It can be seen that the first address in decimal is 302,910,464, which is divisible by 1024.

Subnetting

More 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). 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} , and the prefix length for each subnetwork is n_{sub} . Then the following steps need to be carefully followed to guarantee the proper operation of the subnetworks.

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

$$n_{\text{sub}} = 32 - \log_2 N_{\text{sub}}$$

- ❑ 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 subnetworks.

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 18.5

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; one subblock of 10 addresses, one subblock of 60 addresses, and one subblock of 120 addresses. Design the subblocks.

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. To satisfy the third requirement, we assign addresses to subblocks, starting with the largest and ending with the smallest one.

- a. The number of addresses in the largest subblock, which requires 120 addresses, is not a power of 2. We allocate 128 addresses. The subnet mask for this subnet can be found as $n_1 = 32 - \log_2 128 = 25$. 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 largest subblock, which requires 60 addresses, is not a power of 2 either. We allocate 64 addresses. The subnet mask for this subnet can be found as $n_2 = 32 - \log_2 64 = 26$. 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 smallest subblock, which requires 10 addresses, is not a power of 2 either. We allocate 16 addresses. The subnet mask for this subnet can be found as $n_3 = 32 - \log_2 16 = 28$. The first address in this block is 14.24.74.192/28; the last address is 14.24.74.207/28.

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 14.24.74.208. The last address is 14.24.74.255. We don't know about the prefix length yet. Figure 18.23 shows the configuration of blocks. We have shown the first address in each block.

Address Aggregation

One of the advantages of the CIDR strategy is **address aggregation** (sometimes called *address summarization* or *route summarization*). When blocks of addresses are combined to create a larger block, routing can be done based on the prefix of the larger block. 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.

Example 18.6

Figure 18.24 shows how four small blocks of addresses are assigned to four organizations by an ISP. The ISP combines these four blocks into one single block and advertises the larger block to the rest of the world. Any packet destined for this larger block should be sent to this ISP. It is the responsibility of the ISP to forward the packet to the appropriate organization. This is similar to

Figure 18.23 Solution to Example 18.5

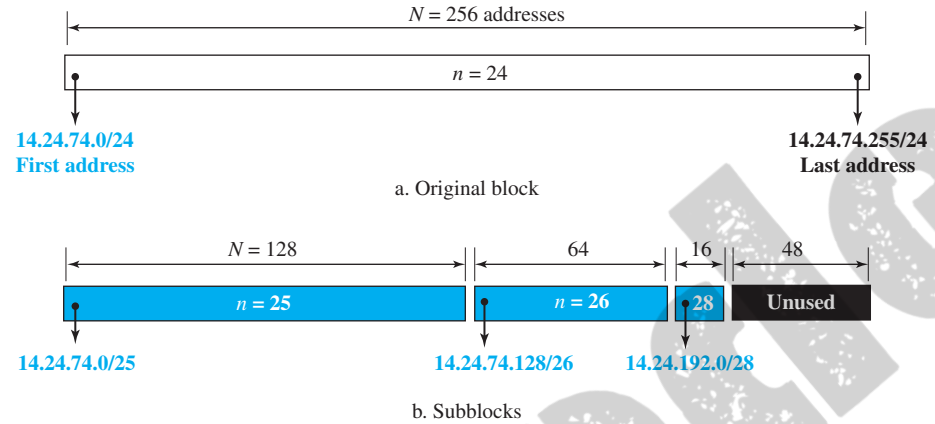
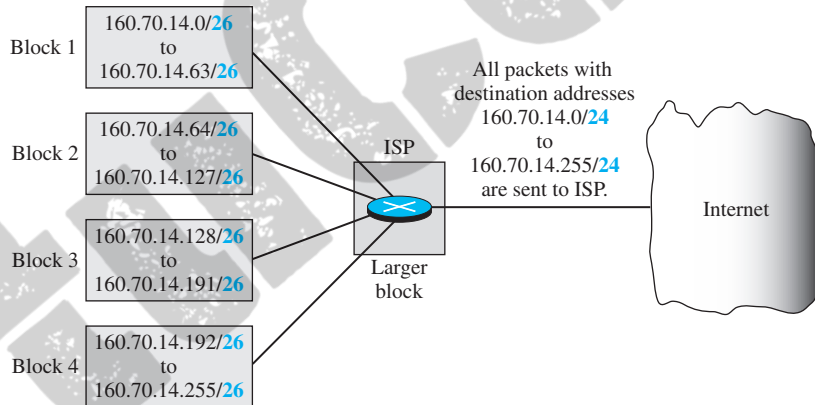


Figure 18.24 Example of address aggregation



routing we can find in a postal network. All packages coming from outside a country are sent first to the capital and then distributed to the corresponding destination.

Special Addresses

Before finishing the topic of addresses in IPv4, we need to mention five special addresses that are used for special purposes: *this-host* address, *limited-broadcast* address, *loopback* address, *private* addresses, and *multicast* addresses.

This-host Address

The only address in the block **0.0.0.0/32** is called the *this-host* address. It is used whenever a host needs to send an IP datagram but it does not know its own address to use as the source address. We will see an example of this case in the next section.

Limited-broadcast Address

The only address in the block **255.255.255.255/32** is called the *limited-broadcast* address. It is used whenever a router or a host needs to send a datagram to all devices in a network. The routers in the network, however, block the packet having this address as the destination; the packet cannot travel outside the network.

Loopback Address

The block **127.0.0.0/8** is called the *loopback* address. A packet with one of the addresses in this block as the destination address never leaves the host; it will remain in the host. Any address in the block is used to test a piece of software in the machine. For example, we can write a client and a server program in which one of the addresses in the block is used as the server address. We can test the programs using the same host to see if they work before running them on different computers.

Private Addresses

Four blocks are assigned as private addresses: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, and 169.254.0.0/16. We will see the applications of these addresses when we discuss NAT later in the chapter.

Multicast Addresses

The block 224.0.0.0/4 is reserved for multicast addresses. We discuss these addresses later in the chapter.

18.4.4 Dynamic Host Configuration Protocol (DHCP)

We have seen that a large organization or an ISP can receive a block of addresses directly from ICANN and a small organization can receive a block of addresses from an ISP. After a block of addresses are assigned to an organization, the network administration can manually assign addresses to the individual hosts or routers. However, address assignment in an organization can be done automatically using the **Dynamic Host Configuration Protocol (DHCP)**. DHCP is an application-layer program, using the client-server paradigm, that actually helps TCP/IP at the network layer.

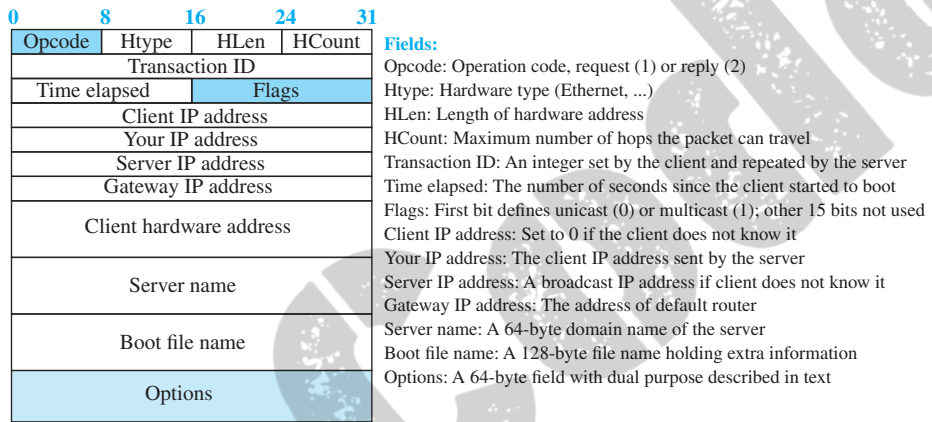
DHCP has found such widespread use in the Internet that it is often called a *plug-and-play protocol*. It can be used in many situations. A network manager can configure DHCP to assign permanent IP addresses to the host and routers. DHCP can also be configured to provide temporary, on demand, IP addresses to hosts. The second capability can provide a temporary IP address to a traveller to connect her laptop to the Internet while she is staying in the hotel. It also allows an ISP with 1000 granted addresses to provide services to 4000 households, assuming not more than one-fourth of customers use the Internet at the same time.

In addition to its IP address, a computer also needs to know the network prefix (or address mask). Most computers also need two other pieces of information, such as the address of a default router to be able to communicate with other networks and the address of a name server to be able to use names instead of addresses, as we will see in Chapter 26. In other words, four pieces of information are normally needed: the computer address, the prefix, the address of a router, and the IP address of a name server. DHCP can be used to provide these pieces of information to the host.

DHCP Message Format

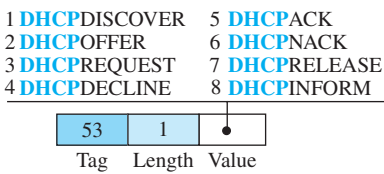
DHCP is a client-server protocol in which the client sends a request message and the server returns a response message. Before we discuss the operation of DHCP, let us show the general format of the DHCP message in Figure 18.25. Most of the fields are explained in the figure, but we need to discuss the option field, which plays a very important role in DHCP.

Figure 18.25 DHCP message format



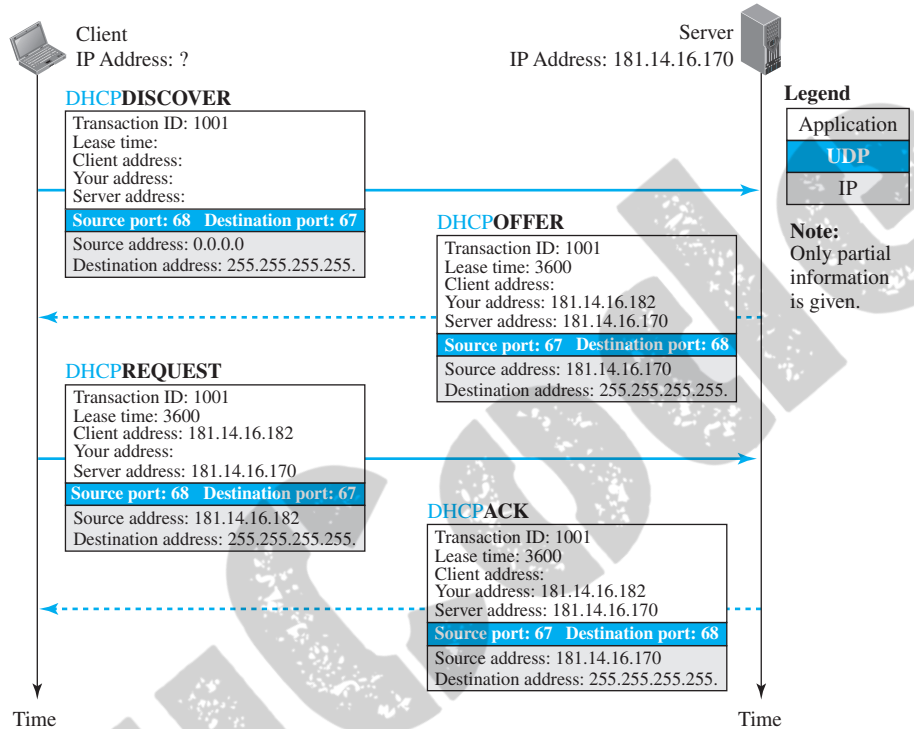
The 64-byte option field has a dual purpose. It can carry either additional information or some specific vendor information. The server uses a number, called a **magic cookie**, in the format of an IP address with the value of 99.130.83.99. When the client finishes reading the message, it looks for this magic cookie. If present, the next 60 bytes are options. An option is composed of three fields: a 1-byte tag field, a 1-byte length field, and a variable-length value field. There are several tag fields that are mostly used by vendors. If the tag field is 53, the value field defines one of the 8 message types shown in Figure 18.26. We show how these message types are used by DHCP.

Figure 18.26 Option format



DHCP Operation

Figure 18.27 shows a simple scenario.

Figure 18.27 Operation of DHCP

1. The joining host creates a **DHCPDISCOVER** message in which only the transaction-ID field is set to a random number. No other field can be set because the host has no knowledge with which to do so. This message is encapsulated in a UDP user datagram with the source port set to 68 and the destination port set to 67. We will discuss the reason for using two well-known port numbers later. The user datagram is encapsulated in an IP datagram with the source address set to **0.0.0.0** ("this host") and the destination address set to **255.255.255.255** (broadcast address). The reason is that the joining host knows neither its own address nor the server address.
2. The DHCP server or servers (if more than one) responds with a **DHCPOFFER** message in which the your address field defines the offered IP address for the joining host and the server address field includes the IP address of the server. The message also includes the lease time for which the host can keep the IP address. This message is encapsulated in a user datagram with the same port numbers, but in the reverse order. The user datagram in turn is encapsulated in a datagram with the server address as the source IP address, but the destination address is a broadcast address, in which the server allows other DHCP servers to receive the offer and give a better offer if they can.

3. The joining host receives one or more offers and selects the best of them. The joining host then sends a **DHCPREQUEST** message to the server that has given the best offer. The fields with known value are set. The message is encapsulated in a user datagram with port numbers as the first message. The user datagram is encapsulated in an IP datagram with the source address set to the new client address, but the destination address still is set to the broadcast address to let the other servers know that their offer was not accepted.
4. Finally, the selected server responds with a **DHCPACK** message to the client if the offered IP address is valid. If the server cannot keep its offer (for example, if the address is offered to another host in between), the server sends a **DHCPNACK** message and the client needs to repeat the process. This message is also broadcast to let other servers know that the request is accepted or rejected.

Two Well-Known Ports

We said that the DHCP uses two well-known ports (68 and 67) instead of one well-known and one ephemeral. The reason for choosing the well-known port 68 instead of an ephemeral port for the client is that the response from the server to the client is broadcast. Remember that an IP datagram with the limited broadcast message is delivered to every host on the network. Now assume that a DHCP client and a DAYTIME client, for example, are both waiting to receive a response from their corresponding server and both have accidentally used the same temporary port number (56017, for example). Both hosts receive the response message from the DHCP server and deliver the message to their clients. The DHCP client processes the message; the DAYTIME client is totally confused with a strange message received. Using a well-known port number prevents this problem from happening. The response message from the DHCP server is not delivered to the DAYTIME client, which is running on the port number 56017, not 68. The temporary port numbers are selected from a different range than the well-known port numbers.

The curious reader may ask what happens if two DHCP clients are running at the same time. This can happen after a power failure and power restoration. In this case the messages can be distinguished by the value of the transaction ID, which separates each response from the other.

Using FTP

The server does not send all of the information that a client may need for joining the network. In the **DHCPACK** message, the server defines the pathname of a file in which the client can find complete information such as the address of the DNS server. The client can then use a file transfer protocol to obtain the rest of the needed information.

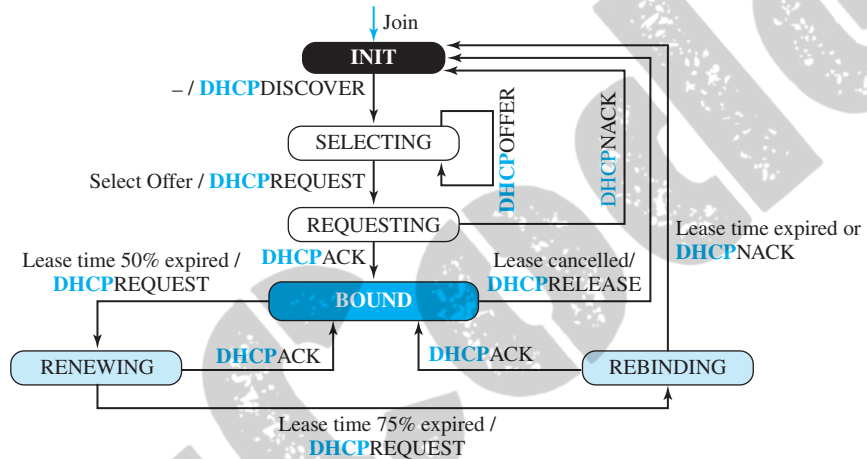
Error Control

DHCP uses the service of UDP, which is not reliable. To provide error control, DHCP uses two strategies. First, DHCP requires that UDP use the checksum. As we will see in Chapter 24, the use of the checksum in UDP is optional. Second, the DHCP client uses timers and a retransmission policy if it does not receive the DHCP reply to a request. However, to prevent a traffic jam when several hosts need to retransmit a request (for example, after a power failure), DHCP forces the client to use a random number to set its timers.

Transition States

The previous scenarios we discussed for the operation of the DHCP were very simple. To provide dynamic address allocation, the DHCP client acts as a state machine that performs transitions from one state to another depending on the messages it receives or sends. Figure 18.28 shows the transition diagram with the main states.

Figure 18.28 FSM for the DHCP client



When the DHCP client first starts, it is in the **INIT** state (initializing state). The client broadcasts a discover message. When it receives an offer, the client goes to the **SELECTING** state. While it is there, it may receive more offers. After it selects an offer, it sends a request message and goes to the **REQUESTING** state. If an ACK arrives while the client is in this state, it goes to the **BOUND** state and uses the IP address. When the lease is 50 percent expired, the client tries to renew it by moving to the **RENEWING** state. If the server renews the lease, the client moves to the **BOUND** state again. If the lease is not renewed and the lease time is 75 percent expired, the client moves to the **REBINDING** state. If the server agrees with the lease (ACK message arrives), the client moves to the **BOUND** state and continues using the IP address; otherwise, the client moves to the **INIT** state and requests another IP address. Note that the client can use the IP address only when it is in the **BOUND**, **RENEWING**, or **REBINDING** state. The above procedure requires that the client uses three timers: *renewal timer* (set to 50 percent of the lease time), *rebinding timer* (set to 75 percent of the lease time), and *expiration timer* (set to the lease time).

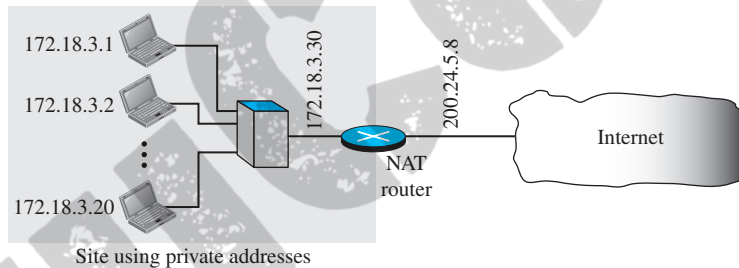
18.4.5 Network Address Resolution (NAT)

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 that in a small business with 20 computers the maximum number of computers that access the Internet simultaneously is only 4. 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, which we discuss in Chapter 32, 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 connection to the global Internet through a NAT-capable router that runs NAT software. Figure 18.29 shows a simple implementation of NAT.

Figure 18.29 NAT



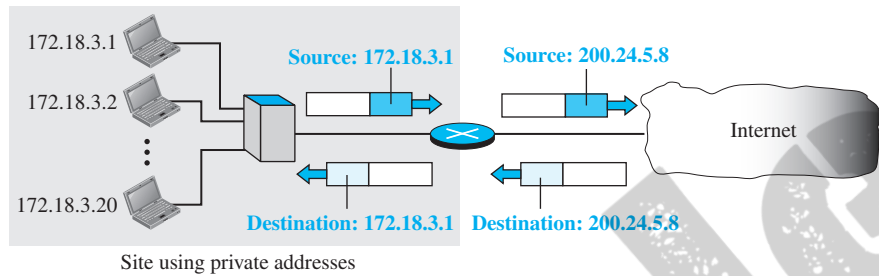
As the figure shows, the private network uses private addresses. The router that connects the network to the global address uses one private address and one global address. The private network is invisible to the rest of the Internet; the rest of the Internet sees only the NAT router with the address 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 18.30 shows an example of address translation.

Translation Table

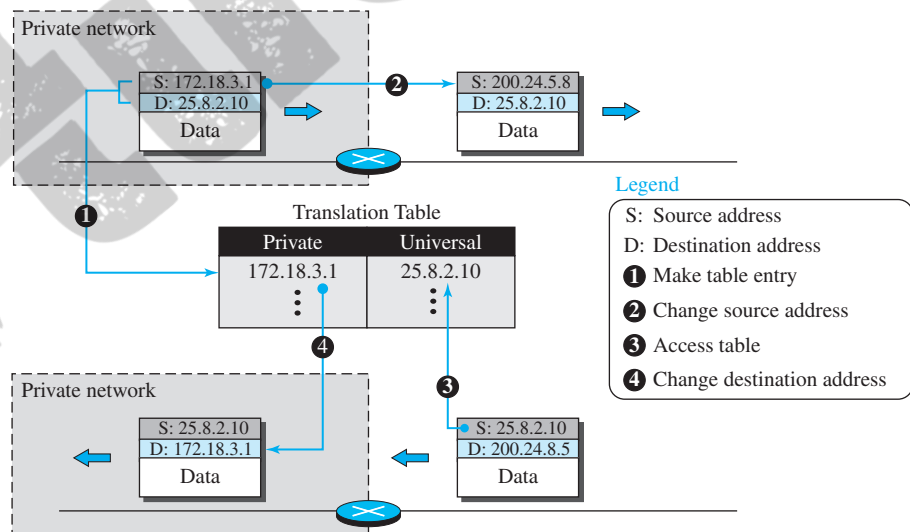
The reader may have noticed that translating the source addresses for an outgoing packet is straightforward. But how does the NAT router know the destination address for a packet coming from the Internet? There may be tens or hundreds of private IP

Figure 18.30 Address translation

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 18.31 shows the idea.

Figure 18.31 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 a 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 outside the network 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.

Using a Pool of IP Addresses

The use of only one global address by the NAT router allows only one private-network host to access a given 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 separate 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 18.1 shows an example of such a table.

Table 18.1 Five-column translation table

<i>Private address</i>	<i>Private port</i>	<i>External address</i>	<i>External port</i>	<i>Transport protocol</i>
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 (1401) 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.

22.2 THE IPv6 PROTOCOL

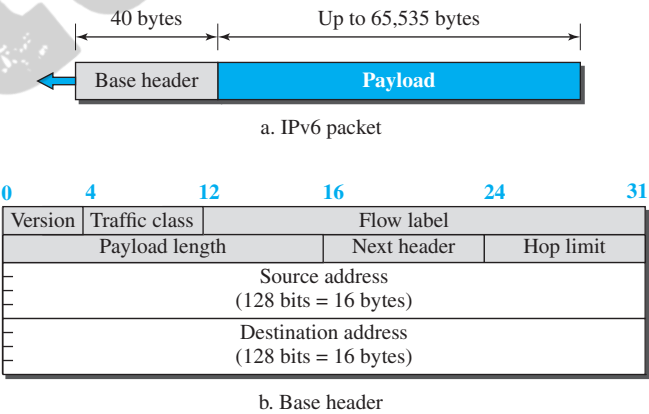
The change of the IPv6 address size requires the change in the IPv4 packet format. The designer of IPv6 decided to implement remedies for other shortcomings now that a change is inevitable. The following shows other changes implemented in the protocol in addition to changing address size and format.

- ❑ **Better header format.** IPv6 uses a new header format in which options are separated from the base header and inserted, when needed, between the base header and the data. This simplifies and speeds up the routing process because most of the options do not need to be checked by routers.
- ❑ **New options.** IPv6 has new options to allow for additional functionalities.
- ❑ **Allowance for extension.** IPv6 is designed to allow the extension of the protocol if required by new technologies or applications.
- ❑ **Support for resource allocation.** In IPv6, the type-of-service field has been removed, but two new fields, traffic class and flow label, have been added to enable the source to request special handling of the packet. This mechanism can be used to support traffic such as real-time audio and video.
- ❑ **Support for more security.** The encryption and authentication options in IPv6 provide confidentiality and integrity of the packet.

22.2.1 Packet Format

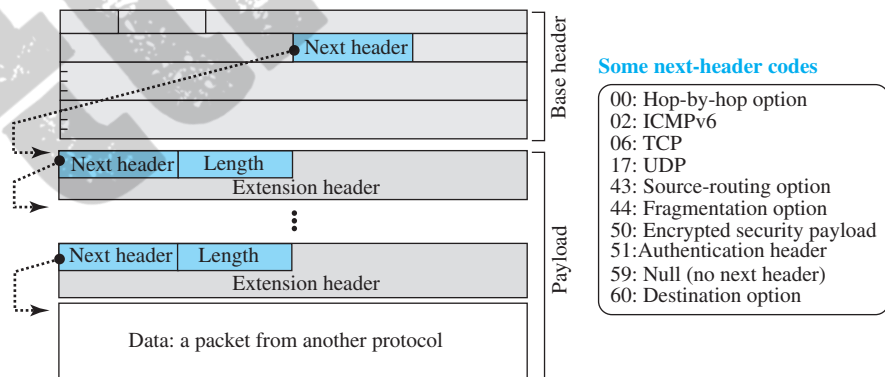
The IPv6 packet is shown in Figure 22.6. Each packet is composed of a base header followed by the payload. The base header occupies 40 bytes, whereas payload can be up to 65,535 bytes of information. The description of fields follows.

Figure 22.6 IPv6 datagram



- ❑ **Version.** The 4-bit version field defines the version number of the IP. For IPv6, the value is 6.
- ❑ **Traffic class.** The 8-bit traffic class field is used to distinguish different payloads with different delivery requirements. It replaces the *type-of-service* field in IPv4.
- ❑ **Flow label.** The flow label is a 20-bit field that is designed to provide special handling for a particular flow of data. We will discuss this field later.
- ❑ **Payload length.** The 2-byte payload length field defines the length of the IP datagram excluding the header. Note that IPv4 defines two fields related to the length: header length and total length. In IPv6, the length of the base header is fixed (40 bytes); only the length of the payload needs to be defined.
- ❑ **Next header.** The **next header** is an 8-bit field defining the type of the first extension header (if present) or the type of the data that follows the base header in the datagram. This field is similar to the protocol field in IPv4, but we talk more about it when we discuss the payload.
- ❑ **Hop limit.** The 8-bit hop limit field serves the same purpose as the TTL field in IPv4.
- ❑ **Source and destination addresses.** The source address field is a 16-byte (128-bit) Internet address that identifies the original source of the datagram. The destination address field is a 16-byte (128-bit) Internet address that identifies the destination of the datagram.
- ❑ **Payload.** Compared to IPv4, the payload field in IPv6 has a different format and meaning, as shown in Figure 22.7.

Figure 22.7 Payload in an IPv6 datagram



The payload in IPv6 means a combination of zero or more extension headers (options) followed by the data from other protocols (UDP, TCP, and so on). In IPv6, options, which are part of the header in IPv4, are designed as extension headers. The payload can have as many extension headers as required by the situation. Each extension header has two mandatory fields, next header and the length,

followed by information related to the particular option. Note that each next header field value (code) defines the type of the next header (hop-by-hop option, source-routing option, . . .); the last next header field defines the protocol (UDP, TCP, . . .) that is carried by the datagram.

Concept of Flow and Priority in IPv6

The IP protocol was originally designed as a connectionless protocol. However, the tendency is to use the IP protocol as a connection-oriented protocol. The MPLS technology described earlier allows us to encapsulate an IPv4 packet in an MPLS header using a label field. In version 6, the flow label has been directly added to the format of the IPv6 datagram to allow us to use IPv6 as a connection-oriented protocol.

To a router, a flow is a sequence of packets that share the same characteristics, such as traveling the same path, using the same resources, having the same kind of security, and so on. A router that supports the handling of flow labels has a flow label table. The table has an entry for each active flow label; each entry defines the services required by the corresponding flow label. When the router receives a packet, it consults its flow label table to find the corresponding entry for the flow label value defined in the packet. It then provides the packet with the services mentioned in the entry. However, note that the flow label itself does not provide the information for the entries of the flow label table; the information is provided by other means, such as the hop-by-hop options or other protocols.

In its simplest form, a flow label can be used to speed up the processing of a packet by a router. When a router receives a packet, instead of consulting the forwarding table and going through a routing algorithm to define the address of the next hop, it can easily look in a flow label table for the next hop.

In its more sophisticated form, a flow label can be used to support the transmission of real-time audio and video. Real-time audio or video, particularly in digital form, requires resources such as high bandwidth, large buffers, long processing time, and so on. A process can make a reservation for these resources beforehand to guarantee that real-time data will not be delayed due to a lack of resources. The use of real-time data and the reservation of these resources require other protocols such as Real-Time Transport Protocol (RTP) and Resource Reservation Protocol (RSVP) in addition to IPv6 (see Chapter 28).

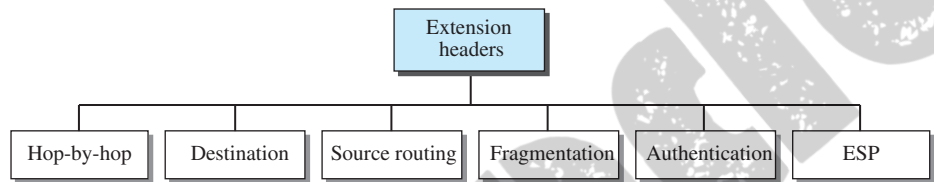
Fragmentation and Reassembly

There are still fragmentation and reassembly of datagrams in the IPv6 protocol, but there is a major difference in this respect. IPv6 datagrams can be fragmented only by the source, not by the routers; the reassembly takes place at the destination. The fragmentation of packets at routers is not allowed to speed up the processing of packets in the router. The fragmentation of a packet in a router needs a lot of processing. The packet needs to be fragmented, all fields related to the fragmentation need to be recalculated. In IPv6, the source can check the size of the packet and make the decision to fragment the packet or not. When a router receives the packet, it can check the size of the packet and drop it if the size is larger than allowed by the MTU of the network ahead. The router then sends a packet-too-big ICMPv6 error message (discussed later) to inform the source.

22.2.2 Extension Header

An IPv6 packet is made of a base header and some extension headers. The length of the base header is fixed at 40 bytes. However, to give more functionality to the IP datagram, the base header can be followed by up to six **extension headers**. Many of these headers are options in IPv4. Six types of extension headers have been defined. These are hop-by-hop option, source routing, fragmentation, authentication, encrypted security payload, and destination option (see Figure 22.8).

Figure 22.8 Extension header types



We briefly describe the extension headers in this section, but the complete description is posted at the book website.

Complete descriptions of extension headers are posted on the book website under Extra Materials for Chapter 22.

Hop-by-Hop Option

The *hop-by-hop option* is used when the source needs to pass information to all routers visited by the datagram. For example, perhaps routers must be informed about certain management, debugging, or control functions. Or, if the length of the datagram is more than the usual 65,535 bytes, routers must have this information. So far, only three hop-by-hop options have been defined: Pad1, PadN, and jumbo payload.

- ❑ **Pad1.** This option is 1 byte long and is designed for alignment purposes. Some options need to start at a specific bit of the 32-bit word. If an option falls short of this requirement by exactly one byte, Pad1 is added.
- ❑ **PadN.** PadN is similar in concept to Pad1. The difference is that PadN is used when 2 or more bytes are needed for alignment.
- ❑ **Jumbo payload.** Recall that the length of the payload in the IP datagram can be a maximum of 65,535 bytes. However, if for any reason a longer payload is required, we can use the jumbo payload option to define this longer length.

Destination Option

The **destination option** is used when the source needs to pass information to the destination only. Intermediate routers are not permitted access to this information. The format of the destination option is the same as the hop-by-hop option. So far, only the Pad1 and PadN options have been defined.

Source Routing

The source routing extension header combines the concepts of the strict source route and the loose source route options of IPv4.

Fragmentation

The concept of **fragmentation** in IPv6 is the same as that in IPv4. However, the place where fragmentation occurs differs. In IPv4, the source or a router is required to fragment if the size of the datagram is larger than the MTU of the network over which the datagram travels. In IPv6, only the original source can fragment. A source must use a **Path MTU Discovery technique** to find the smallest MTU supported by any network on the path. The source then fragments using this knowledge.

If the source does not use a Path MTU Discovery technique, it fragments the datagram to a size of 1280 bytes or smaller. This is the minimum size of MTU required for each network connected to the Internet.

Authentication

The **authentication** extension header has a dual purpose: it validates the message sender and ensures the integrity of data. The former is needed so the receiver can be sure that a message is from the genuine sender and not from an imposter. The latter is needed to check that the data is not altered in transition by some hacker. We discuss more about authentication in Chapters 31 and 32.

Encrypted Security Payload

The **encrypted security payload (ESP)** is an extension that provides confidentiality and guards against eavesdropping. Again, we discuss providing more confidentiality for IP packets in Chapter 32.

Comparison of Options between IPv4 and IPv6

The following shows a quick comparison between the options used in IPv4 and the options used in IPv6 (as extension headers).

- ❑ The no-operation and end-of-option options in IPv4 are replaced by Pad1 and PadN options in IPv6.
- ❑ The record route option is not implemented in IPv6 because it was not used.
- ❑ The timestamp option is not implemented because it was not used.
- ❑ The source route option is called the *source route extension header* in IPv6.
- ❑ The fragmentation fields in the base header section of IPv4 have moved to the fragmentation extension header in IPv6.
- ❑ The authentication extension header is new in IPv6.
- ❑ The encrypted security payload extension header is new in IPv6.

20.1 INTRODUCTION

Unicast routing in the Internet, with a large number of routers and a huge number of hosts, can be done only by using hierarchical routing: routing in several steps using different routing algorithms. In this section, we first discuss the general concept of unicast routing in an *internet*: an internetwork made of networks connected by routers. After the routing concepts and algorithms are understood, we show how we can apply them to the Internet using hierarchical routing.

20.1.1 General Idea

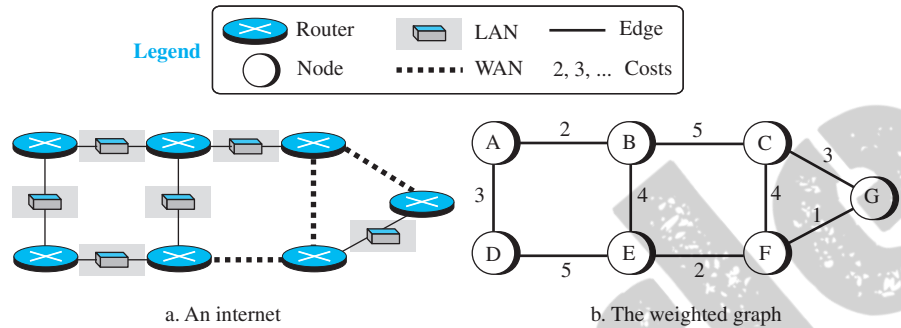
In unicast routing, a packet is routed, hop by hop, from its source to its destination by the help of forwarding tables. The source host needs no forwarding table because it delivers its packet to the default router in its local network. The destination host needs no forwarding table either because it receives the packet from its default router in its local network. This means that only the routers that glue together the networks in the internet need forwarding tables. With the above explanation, routing a packet from its source to its destination means routing the packet from a *source router* (the default router of the source host) to a *destination router* (the router connected to the destination network). Although a packet needs to visit the source and the destination routers, the question is what other routers the packet should visit. In other words, there are several routes that a packet can travel from the source to the destination; what must be determined is which route the packet should take.

An Internet as a Graph

To find the best route, an internet can be modeled as a *graph*. A graph in computer science is a set of *nodes* and *edges* (lines) that connect the nodes. To model an internet as a graph, we can think of each router as a node and each network between a pair of routers as an edge. An internet is, in fact, modeled as a *weighted graph*, in which each edge is associated with a cost. If a weighted graph is used to represent a geographical area, the nodes can be cities and the edges can be roads connecting the cities; the weights, in this case, are distances between cities. In routing, however, the cost of an edge has a different interpretation in different routing protocols, which we discuss in a later section. For the moment, we assume that there is a cost associated with each edge. If there is no edge between the nodes, the cost is infinity. Figure 20.1 shows how an internet can be modeled as a graph.

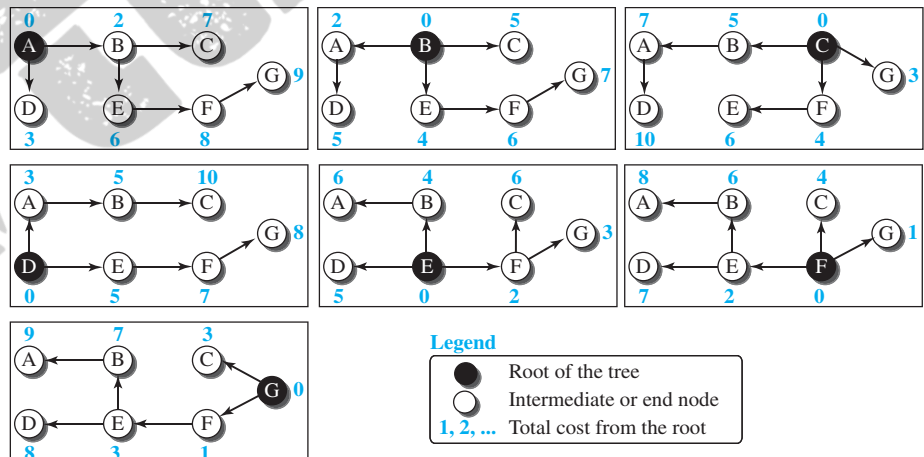
20.1.2 Least-Cost Routing

When an internet is modeled as a weighted graph, one of the ways to interpret the *best* route from the source router to the destination router is to find the *least cost* between the two. In other words, the source router chooses a route to the destination router in such a way that the total cost for the route is the least cost among all possible routes. In Figure 20.1, the best route between A and E is A-B-E, with the cost of 6. This means that each router needs to find the least-cost route between itself and all the other routers to be able to route a packet using this criteria.

Figure 20.1 An internet and its graphical representation

Least-Cost Trees

If there are N routers in an internet, there are $(N - 1)$ least-cost paths from each router to any other router. This means we need $N \times (N - 1)$ least-cost paths for the whole internet. If we have only 10 routers in an internet, we need 90 least-cost paths. A better way to see all of these paths is to combine them in a **least-cost tree**. A least-cost tree is a tree with the source router as the root that spans the whole graph (visits all other nodes) and in which the path between the root and any other node is the shortest. In this way, we can have only one shortest-path tree for each node; we have N least-cost trees for the whole internet. We show how to create a least-cost tree for each node later in this section; for the moment, Figure 20.2 shows the seven least-cost trees for the internet in Figure 20.1.

Figure 20.2 Least-cost trees for nodes in the internet of Figure 20.1

The least-cost trees for a weighted graph can have several properties if they are created using consistent criteria.

1. The least-cost route from X to Y in X's tree is the inverse of the least-cost route from Y to X in Y's tree; the cost in both directions is the same. For example, in Figure 20.2, the route from A to F in A's tree is (A → B → E → F), but the route from F to A in F's tree is (F → E → B → A), which is the inverse of the first route. The cost is 8 in each case.
2. Instead of travelling from X to Z using X's tree, we can travel from X to Y using X's tree and continue from Y to Z using Y's tree. For example, in Figure 20.2, we can go from A to G in A's tree using the route (A → B → E → F → G). We can also go from A to E in A's tree (A → B → E) and then continue in E's tree using the route (E → F → G). The combination of the two routes in the second case is the same route as in the first case. The cost in the first case is 9; the cost in the second case is also 9 (6 + 3).

20.2 ROUTING ALGORITHMS

After discussing the general idea behind least-cost trees and the forwarding tables that can be made from them, now we concentrate on the routing algorithms. Several routing algorithms have been designed in the past. The differences between these methods are in the way they interpret the least cost and the way they create the least-cost tree for each node. In this section, we discuss the common algorithms; later we show how a routing protocol in the Internet implements one of these algorithms.

20.2.1 Distance-Vector Routing

The **distance-vector (DV) routing** uses the goal we discussed in the introduction, to find the best route. In distance-vector routing, the first thing each node creates is its own least-cost tree with the rudimentary information it has about its immediate neighbors. The incomplete trees are exchanged between immediate neighbors to make the trees more and more complete and to represent the whole internet. We can say that in distance-vector routing, a router continuously tells all of its neighbors what it knows about the whole internet (although the knowledge can be incomplete).

Before we show how incomplete least-cost trees can be combined to make complete ones, we need to discuss two important topics: the Bellman-Ford equation and the concept of distance vectors, which we cover next.

Bellman-Ford Equation

The heart of distance-vector routing is the famous **Bellman-Ford** equation. This equation is used to find the least cost (shortest distance) between a source node, x , and a destination node, y , through some intermediary nodes (a, b, c, \dots) when the costs between the source and the intermediary nodes and the least costs between the intermediary nodes and the destination are given. The following shows the general case in which D_{ij} is the shortest distance and c_{ij} is the cost between nodes i and j .

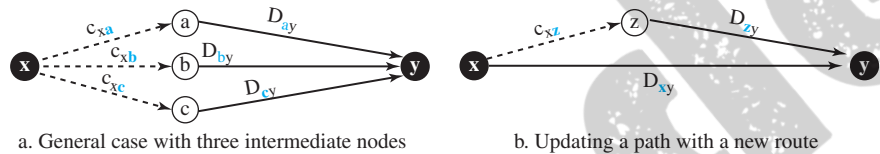
$$D_{xy} = \min \{ (c_{xa} + D_{ay}), (c_{xb} + D_{by}), (c_{xc} + D_{cy}), \dots \}$$

In distance-vector routing, normally we want to update an existing least cost with a least cost through an intermediary node, such as z , if the latter is shorter. In this case, the equation becomes simpler, as shown below:

$$D_{xy} = \min \{ D_{xy}, (c_{xz} + D_{zy}) \}$$

Figure 20.3 shows the idea graphically for both cases.

Figure 20.3 Graphical idea behind Bellman-Ford equation



We can say that the Bellman-Ford equation enables us to build a new least-cost path from previously established least-cost paths. In Figure 20.3, we can think of $(a \rightarrow y)$, $(b \rightarrow y)$, and $(c \rightarrow y)$ as previously established least-cost paths and $(x \rightarrow y)$ as the new least-cost path. We can even think of this equation as the builder of a new least-cost tree from previously established least-cost trees if we use the equation repeatedly. In other words, the use of this equation in distance-vector routing is a witness that this method also uses least-cost trees, but this use may be in the background.

We will shortly show how we use the Bellman-Ford equation and the concept of distance vectors to build least-cost paths for each node in distance-vector routing, but first we need to discuss the concept of a distance vector.

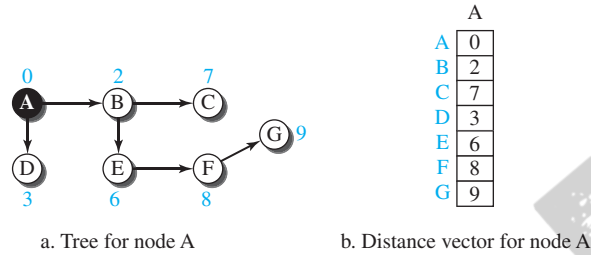
Distance Vectors

The concept of a **distance vector** is the rationale for the name *distance-vector routing*. A least-cost tree is a combination of least-cost paths from the root of the tree to all destinations. These paths are graphically glued together to form the tree. Distance-vector routing unglues these paths and creates a *distance vector*, a one-dimensional array to represent the tree. Figure 20.4 shows the tree for node A in the internet in Figure 20.1 and the corresponding distance vector.

Note that the *name* of the distance vector defines the root, the *indexes* define the destinations, and the *value* of each cell defines the least cost from the root to the destination. A distance vector does not give the path to the destinations as the least-cost tree does; it gives only the least costs to the destinations. Later we show how we can change a distance vector to a forwarding table, but we first need to find all distance vectors for an internet.

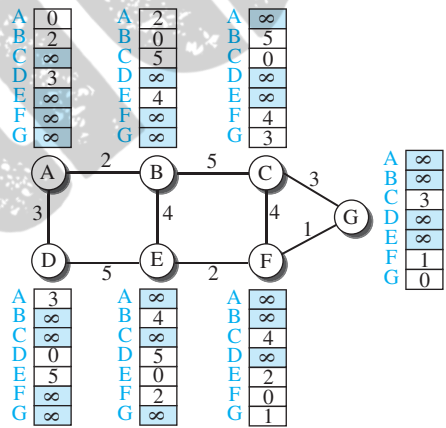
We know that a distance vector can represent least-cost paths in a least-cost tree, but the question is how each node in an internet originally creates the corresponding vector. Each node in an internet, when it is booted, creates a very rudimentary distance vector with the minimum information the node can obtain from its neighborhood. The node sends some greeting messages out of its interfaces and discovers the identity of the immediate neighbors and the distance between itself and each neighbor. It then

Figure 20.4 *The distance vector corresponding to a tree*



makes a simple distance vector by inserting the discovered distances in the corresponding cells and leaves the value of other cells as infinity. Do these distance vectors represent least-cost paths? They do, considering the limited information a node has. When we know only one distance between two nodes, it is the least cost. Figure 20.5 shows all distance vectors for our internet. However, we need to mention that these vectors are made asynchronously, when the corresponding node has been booted; the existence of all of them in a figure does not mean synchronous creation of them.

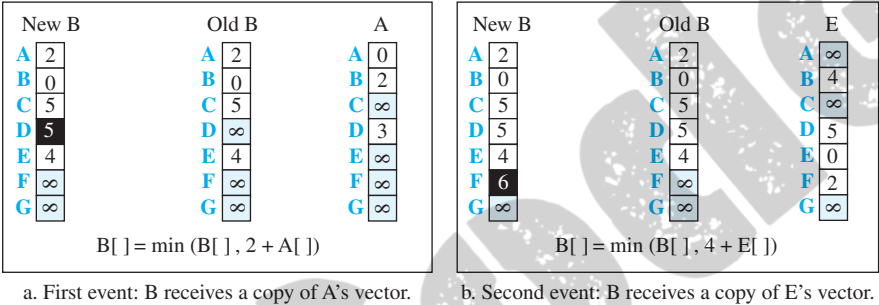
Figure 20.5 *The first distance vector for an internet*



These rudimentary vectors cannot help the internet to effectively forward a packet. For example, node A thinks that it is not connected to node G because the corresponding cell shows the least cost of infinity. To improve these vectors, the nodes in the internet need to help each other by exchanging information. After each node has created its vector, it sends a copy of the vector to all its immediate neighbors. After a node receives a distance vector from a neighbor, it updates its distance vector using the Bellman-Ford equation (second case). However, we need to understand that we need to update, not

only one least cost, but N of them in which N is the number of the nodes in the internet. If we are using a program, we can do this using a loop; if we are showing the concept on paper, we can show the whole vector instead of the N separate equations. We show the whole vector instead of seven equations for each update in Figure 20.6. The figure shows two asynchronous events, happening one after another with some time in

Figure 20.6 Updating distance vectors



Note:
X[] : the whole vector

between. In the first event, node A has sent its vector to node B. Node B updates its vector using the cost $c_{BA} = 2$. In the second event, node E has sent its vector to node B. Node B updates its vector using the cost $c_{EA} = 4$.

After the first event, node B has one improvement in its vector: its least cost to node D has changed from infinity to 5 (via node A). After the second event, node B has one more improvement in its vector; its least cost to node F has changed from infinity to 6 (via node E). We hope that we have convinced the reader that exchanging vectors eventually stabilizes the system and allows all nodes to find the ultimate least cost between themselves and any other node. We need to remember that after updating a node, it immediately sends its updated vector to all neighbors. Even if its neighbors have received the previous vector, the updated one may help more.

Distance-Vector Routing Algorithm

Now we can give a simplified pseudocode for the distance-vector routing algorithm, as shown in Table 20.1. The algorithm is run by its node independently and asynchronously.

Table 20.1 Distance-Vector Routing Algorithm for a Node

1	Distance_Vector_Routing ()
2	{
3	// Initialize (create initial vectors for the node)
4	D[myself] = 0

Table 20.1 Distance-Vector Routing Algorithm for a Node (continued)

```

5   for (y = 1 to N)
6   {
7       if (y is a neighbor)
8           D[y] = c[myself][y]
9       else
10          D[y] = ∞
11  }
12  send vector {D[1], D[2], ..., D[N]} to all neighbors
13  // Update (improve the vector with the vector received from a neighbor)
14  repeat (forever)
15  {
16      wait (for a vector Dw from a neighbor w or any change in the link)
17      for (y = 1 to N)
18      {
19          D[y] = min [D[y], (c[myself][w] + Dw[y])] // Bellman-Ford equation
20      }
21      if (any change in the vector)
22          send vector {D[1], D[2], ..., D[N]} to all neighbors
23  }
24 } // End of Distance Vector

```

Lines 4 to 11 initialize the vector for the node. Lines 14 to 23 show how the vector can be updated after receiving a vector from the immediate neighbor. The *for* loop in lines 17 to 20 allows all entries (cells) in the vector to be updated after receiving a new vector. Note that the node sends its vector in line 12, after being initialized, and in line 22, after it is updated.

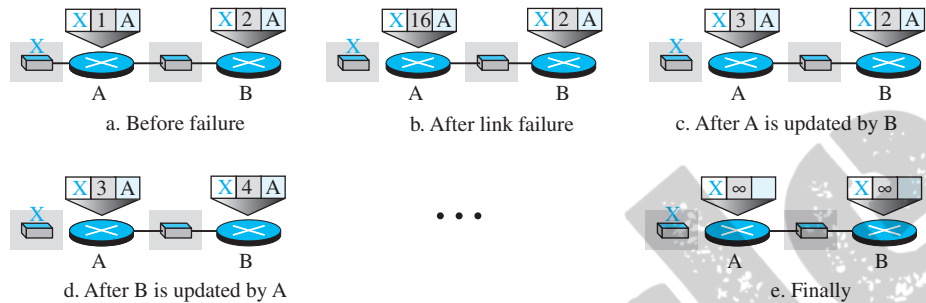
Count to Infinity

A problem with distance-vector routing is that any decrease in cost (good news) propagates quickly, but any increase in cost (bad news) will propagate slowly. For a routing protocol to work properly, if a link is broken (cost becomes infinity), every other router should be aware of it immediately, but in distance-vector routing, this takes some time. The problem is referred to as *count to infinity*. It sometimes takes several updates before the cost for a broken link is recorded as infinity by all routers.

Two-Node Loop

One example of count to infinity is the two-node loop problem. To understand the problem, let us look at the scenario depicted in Figure 20.7.

The figure shows a system with three nodes. We have shown only the portions of the forwarding table needed for our discussion. At the beginning, both nodes A and B

Figure 20.7 Two-node instability

know how to reach node X. But suddenly, the link between A and X fails. Node A changes its table. If A can send its table to B immediately, everything is fine. However, the system becomes unstable if B sends its forwarding table to A before receiving A's forwarding table. Node A receives the update and, assuming that B has found a way to reach X, immediately updates its forwarding table. Now A sends its new update to B. Now B thinks that something has been changed around A and updates its forwarding table. The cost of reaching X increases gradually until it reaches infinity. At this moment, both A and B know that X cannot be reached. However, during this time the system is not stable. Node A thinks that the route to X is via B; node B thinks that the route to X is via A. If A receives a packet destined for X, the packet goes to B and then comes back to A. Similarly, if B receives a packet destined for X, it goes to A and comes back to B. Packets bounce between A and B, creating a two-node loop problem. A few solutions have been proposed for instability of this kind.

Split Horizon

One solution to instability is called **split horizon**. In this strategy, instead of flooding the table through each interface, each node sends only part of its table through each interface. If, according to its table, node B thinks that the optimum route to reach X is via A, it does not need to advertise this piece of information to A; the information has come from A (A already knows). Taking information from node A, modifying it, and sending it back to node A is what creates the confusion. In our scenario, node B eliminates the last line of its forwarding table before it sends it to A. In this case, node A keeps the value of infinity as the distance to X. Later, when node A sends its forwarding table to B, node B also corrects its forwarding table. The system becomes stable after the first update: both node A and node B know that X is not reachable.

Poison Reverse

Using the split-horizon strategy has one drawback. Normally, the corresponding protocol uses a timer, and if there is no news about a route, the node deletes the route from its table. When node B in the previous scenario eliminates the route to X from its advertisement to A, node A cannot guess whether this is due to the split-horizon strategy (the source of information was A) or because B has not received any news about X recently. In the **poison reverse** strategy B can still advertise the value for X, but if the source of

information is A, it can replace the distance with infinity as a warning: “Do not use this value; what I know about this route comes from you.”

Three-Node Instability

The two-node instability can be avoided using split horizon combined with poison reverse. However, if the instability is between three nodes, stability cannot be guaranteed.

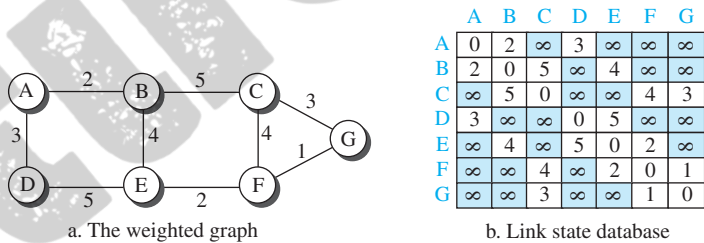
20.2.2 Link-State Routing

A routing algorithm that directly follows our discussion for creating least-cost trees and forwarding tables is **link-state (LS) routing**. This method uses the term *link-state* to define the characteristic of a link (an edge) that represents a network in the internet. In this algorithm the cost associated with an edge defines the state of the link. Links with lower costs are preferred to links with higher costs; if the cost of a link is infinity, it means that the link does not exist or has been broken.

Link-State Database (LSDB)

To create a least-cost tree with this method, each node needs to have a complete *map* of the network, which means it needs to know the state of each link. The collection of states for all links is called the **link-state database (LSDB)**. There is only one LSDB for the whole internet; each node needs to have a duplicate of it to be able to create the least-cost tree. Figure 20.8 shows an example of an LSDB for the graph in Figure 20.1. The LSDB can be represented as a two-dimensional array(matrix) in which the value of each cell defines the cost of the corresponding link.

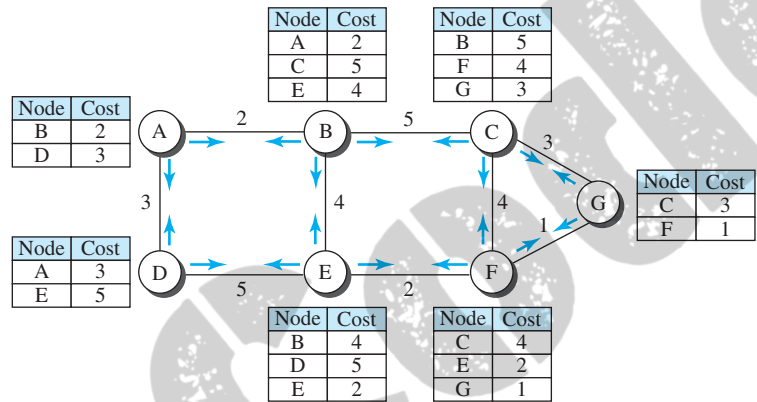
Figure 20.8 Example of a link-state database



Now the question is how each node can create this LSDB that contains information about the whole internet. This can be done by a process called **flooding**. Each node can send some greeting messages to all its immediate neighbors (those nodes to which it is connected directly) to collect two pieces of information for each neighboring node: the identity of the node and the cost of the link. The combination of these two pieces of information is called the *LS packet* (LSP); the LSP is sent out of each interface, as shown in Figure 20.9 for our internet in Figure 20.1. When a node receives an LSP from one of its interfaces, it compares the LSP with the copy it may already have. If the newly arrived LSP is older than the one it has (found by checking the sequence number), it discards the LSP. If it is newer or the first one received, the node discards the old LSP (if there is one) and keeps the received one. It then sends a copy of it out of each

interface except the one from which the packet arrived. This guarantees that flooding stops somewhere in the network (where a node has only one interface). We need to convince ourselves that, after receiving all new LSPs, each node creates the comprehensive LSDB as shown in Figure 20.9. This LSDB is the same for each node and shows the whole map of the internet. In other words, a node can make the whole map if it needs to, using this LSDB.

Figure 20.9 LSPs created and sent out by each node to build LSDB



We can compare the link-state routing algorithm with the distance-vector routing algorithm. In the distance-vector routing algorithm, each router tells its neighbors what it knows about the whole internet; in the link-state routing algorithm, each router tells the whole internet what it knows about its neighbors.

Formation of Least-Cost Trees

To create a least-cost tree for itself, using the shared LSDB, each node needs to run the famous **Dijkstra Algorithm**. This iterative algorithm uses the following steps:

1. The node chooses itself as the root of the tree, creating a tree with a single node, and sets the total cost of each node based on the information in the LSDB.
2. The node selects one node, among all nodes not in the tree, which is closest to the root, and adds this to the tree. After this node is added to the tree, the cost of all other nodes not in the tree needs to be updated because the paths may have been changed.
3. The node repeats step 2 until all nodes are added to the tree.

We need to convince ourselves that the above three steps finally create the least-cost tree. Table 20.2 shows a simplified version of Dijkstra's algorithm.

Table 20.2 Dijkstra's Algorithm

1	Dijkstra's Algorithm ()	
2	{	
3	// Initialization	
4	Tree = {root}	// Tree is made only of the root

Table 20.2 *Dijkstra's Algorithm (continued)*

```

5   for (y = 1 to N)                // N is the number of nodes
6   {
7       if (y is the root)
8           D[y] = 0                // D[y] is shortest distance from root to node y
9       else if (y is a neighbor)
10          D[y] = c[root][y]        // c[x][y] is cost between nodes x and y in LSDB
11      else
12          D[y] = ∞
13  }
14  // Calculation
15  repeat
16  {
17      find a node w, with D[w] minimum among all nodes not in the Tree
18      Tree = Tree ∪ { w }          // Add w to tree
19      // Update distances for all neighbors of w
20      for (every node x, which is a neighbor of w and not in the Tree)
21      {
22          D[x] = min { D[x], (D[w] + c[w][x]) }
23      }
24  } until (all nodes included in the Tree)
25 } // End of Dijkstra

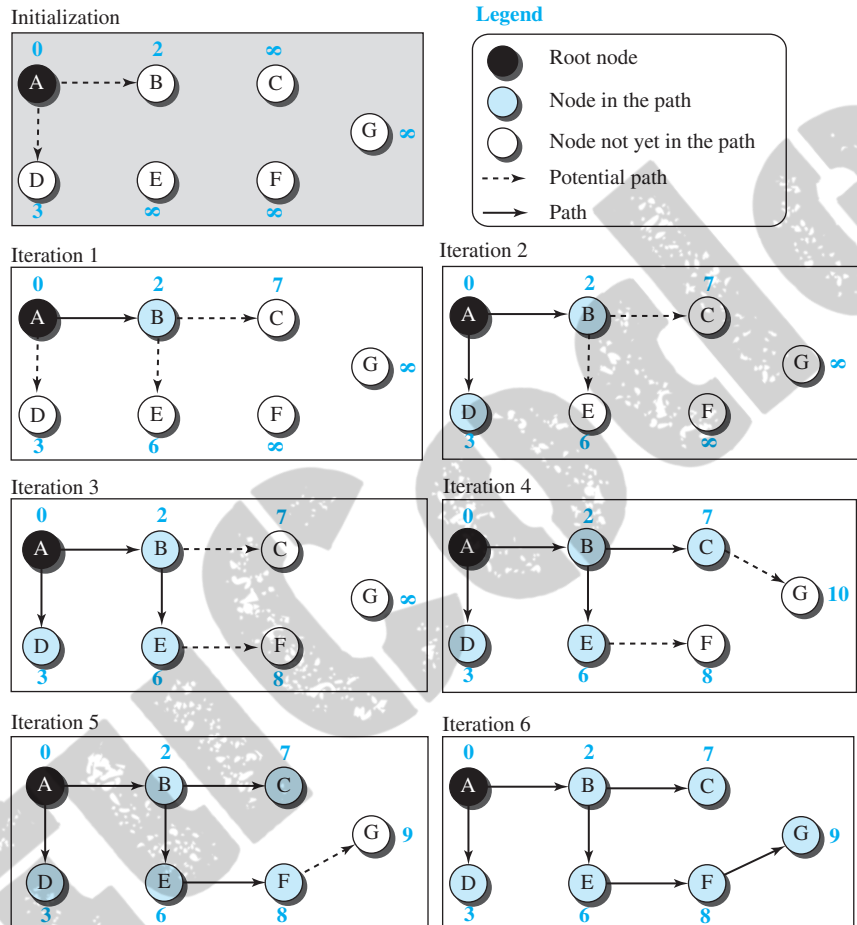
```

Lines 4 to 13 implement step 1 in the algorithm. Lines 16 to 23 implement step 2 in the algorithm. Step 2 is repeated until all nodes are added to the tree.

Figure 20.10 shows the formation of the least-cost tree for the graph in Figure 20.8 using Dijkstra's algorithm. We need to go through an initialization step and six iterations to find the least-cost tree.

20.2.3 Path-Vector Routing

Both link-state and distance-vector routing are based on the least-cost goal. However, there are instances where this goal is not the priority. For example, assume that there are some routers in the internet that a sender wants to prevent its packets from going through. For example, a router may belong to an organization that does not provide enough security or it may belong to a commercial rival of the sender which might inspect the packets for obtaining information. Least-cost routing does not prevent a packet from passing through an area when that area is in the least-cost path. In other words, the least-cost goal, applied by LS or DV routing, does not allow a sender to apply specific policies to the route a packet may take. Aside from safety and security, there are occasions, as discussed in the next section, in which the goal of routing is merely reachability: to allow the packet to reach its destination more efficiently without assigning costs to the route.

Figure 20.10 *Least-cost tree*

To respond to these demands, a third routing algorithm, called **path-vector (PV) routing** has been devised. Path-vector routing does not have the drawbacks of LS or DV routing as described above because it is not based on least-cost routing. The best route is determined by the source using the policy it imposes on the route. In other words, the source can control the path. Although path-vector routing is not actually used in an internet, and is mostly designed to route a packet between ISPs, we discuss the principle of this method in this section as though applied to an internet. In the next section, we show how it is used in the Internet.

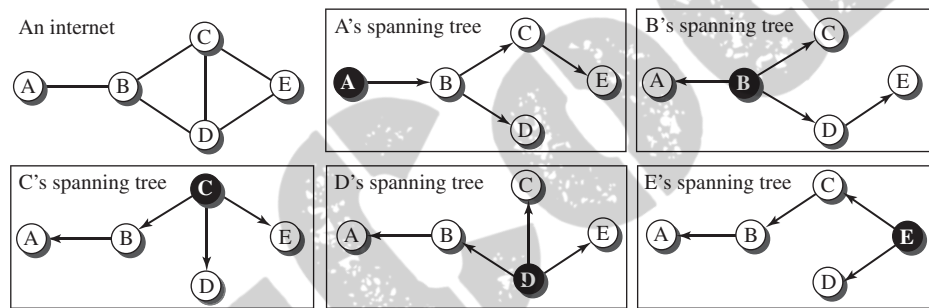
Spanning Trees

In path-vector routing, the path from a source to all destinations is also determined by the *best* spanning tree. The best spanning tree, however, is not the least-cost tree; it is

the tree determined by the source when it imposes its own policy. If there is more than one route to a destination, the source can choose the route that meets its policy best. A source may apply several policies at the same time. One of the common policies uses the minimum number of nodes to be visited (something similar to least-cost). Another common policy is to avoid some nodes as the middle node in a route.

Figure 20.11 shows a small internet with only five nodes. Each source has created its own spanning tree that meets its policy. The policy imposed by all sources is to use the minimum number of nodes to reach a destination. The spanning tree selected by A and E is such that the communication does not pass through D as a middle node. Similarly, the spanning tree selected by B is such that the communication does not pass through C as a middle node.

Figure 20.11 Spanning trees in path-vector routing



Creation of Spanning Trees

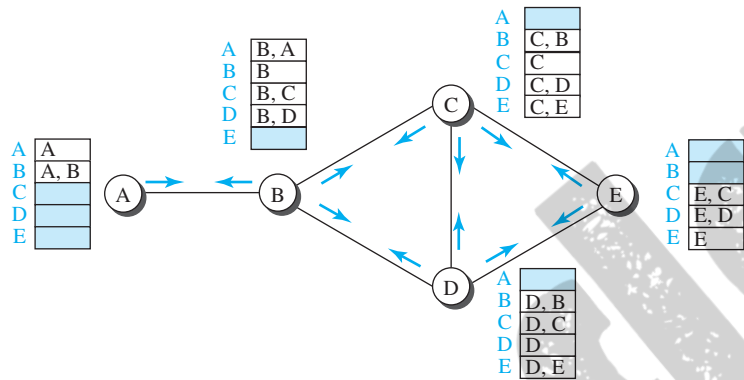
Path-vector routing, like distance-vector routing, is an asynchronous and distributed routing algorithm. The spanning trees are made, gradually and asynchronously, by each node. When a node is booted, it creates a *path vector* based on the information it can obtain about its immediate neighbor. A node sends greeting messages to its immediate neighbors to collect these pieces of information. Figure 20.12 shows all of these path vectors for our internet in Figure 20.11. Note, however, that we do not mean that all of these tables are created simultaneously; they are created when each node is booted. The figure also shows how these path vectors are sent to immediate neighbors after they have been created (arrows).

Each node, after the creation of the initial path vector, sends it to all its immediate neighbors. Each node, when it receives a path vector from a neighbor, updates its path vector using an equation similar to the Bellman-Ford, but applying its own policy instead of looking for the least cost. We can define this equation as

$$\text{Path}(x, y) = \text{best} \{ \text{Path}(x, y), [(x + \text{Path}(\mathbf{v}, y))] \} \quad \text{for all } \mathbf{v}'\text{'s in the internet.}$$

In this equation, the operator (+) means to add x to the beginning of the path. We also need to be cautious to avoid adding a node to an empty path because an empty path means one that does not exist.

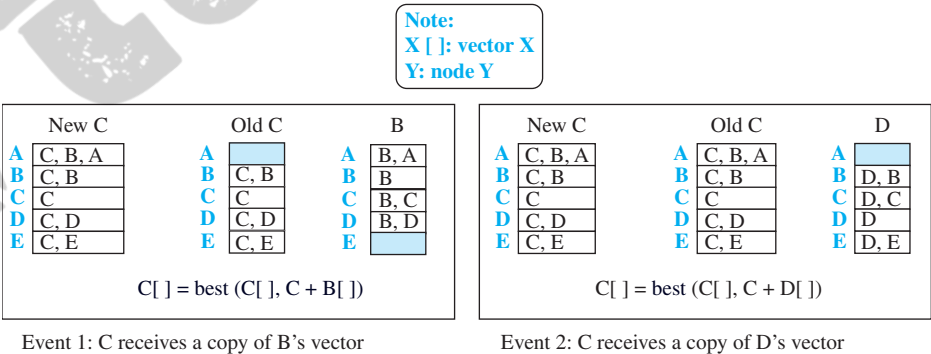
Figure 20.12 Path vectors made at booting time



The policy is defined by selecting the *best* of multiple paths. Path-vector routing also imposes one more condition on this equation: If Path (v, y) includes x , that path is discarded to avoid a loop in the path. In other words, x does not want to visit itself when it selects a path to y .

Figure 20.13 shows the path vector of node C after two events. In the first event, node C receives a copy of B's vector, which improves its vector: now it knows how to reach node A. In the second event, node C receives a copy of D's vector, which does not change its vector. As a matter of fact the vector for node C after the first event is stabilized and serves as its forwarding table.

Figure 20.13 Updating path vectors



Path-Vector Algorithm

Based on the initialization process and the equation used in updating each forwarding table after receiving path vectors from neighbors, we can write a simplified version of the path vector algorithm as shown in Table 20.3.

Table 20.3 Path-vector algorithm for a node

```

1 Path_Vector_Routing ( )
2 {
3     // Initialization
4     for (y = 1 to N)
5     {
6         if (y is myself)
7             Path[y] = myself
8         else if (y is a neighbor)
9             Path[y] = myself + neighbor node
10        else
11            Path[y] = empty
12    }
13    Send vector {Path[1], Path[2], ..., Path[y]} to all neighbors
14    // Update
15    repeat (forever)
16    {
17        wait (for a vector Pathw from a neighbor w)
18        for (y = 1 to N)
19        {
20            if (Pathw includes myself)
21                discard the path // Avoid any loop
22            else
23                Path[y] = best {Path[y], (myself + Pathw[y])}
24        }
25        If (there is a change in the vector)
26            Send vector {Path[1], Path[2], ..., Path[y]} to all neighbors
27    }
28 } // End of Path Vector

```

Lines 4 to 12 show the initialization for the node. Lines 17 to 24 show how the node updates its vector after receiving a vector from the neighbor. The update process is repeated forever. We can see the similarities between this algorithm and the DV algorithm.

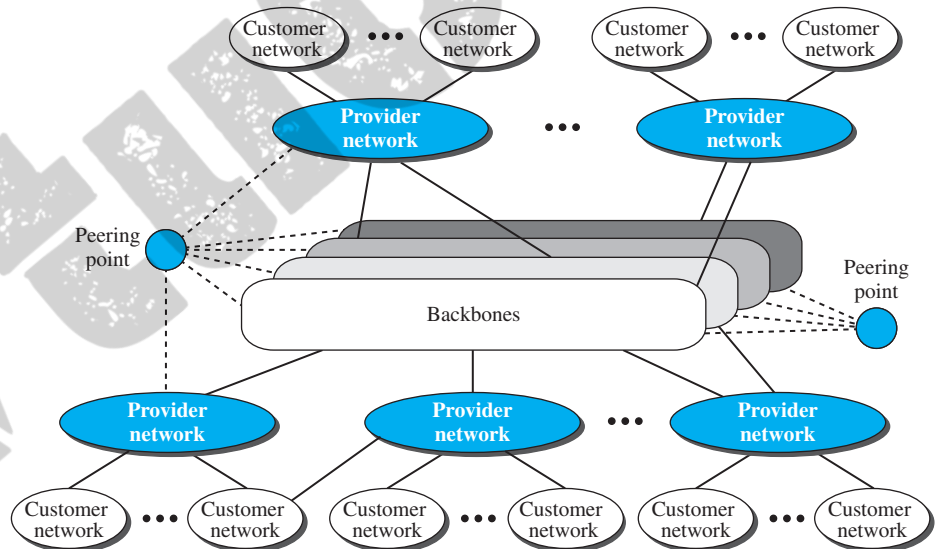
20.3 UNICAST ROUTING PROTOCOLS

In the previous section, we discussed unicast routing algorithms; in this section, we discuss unicast routing protocols used in the Internet. Although three protocols we discuss here are based on the corresponding algorithms we discussed before, a protocol is more than an algorithm. A protocol needs to define its domain of operation, the messages exchanged, communication between routers, and interaction with protocols in other domains. After an introduction, we discuss three common protocols used in the Internet: Routing Information Protocol (RIP), based on the distance-vector algorithm, Open Shortest Path First (OSPF), based on the link-state algorithm, and Border Gateway Protocol (BGP), based on the path-vector algorithm.

20.3.1 Internet Structure

Before discussing unicast routing protocols, we need to understand the structure of today's Internet. The Internet has changed from a tree-like structure, with a single backbone, to a multi-backbone structure run by different private corporations today. Although it is difficult to give a general view of the Internet today, we can say that the Internet has a structure similar to what is shown in Figure 20.14.

Figure 20.14 *Internet structure*



There are several *backbones* run by private communication companies that provide global connectivity. These backbones are connected by some *peering points* that allow connectivity between backbones. At a lower level, there are some *provider networks* that use the backbones for global connectivity but provide services to Internet customers.

Finally, there are some customer *networks* that use the services provided by the provider networks. Any of these three entities (backbone, provider network, or customer network) can be called an Internet Service Provider or ISP. They provide services, but at different levels.

Hierarchical Routing

The Internet today is made of a huge number of networks and routers that connect them. It is obvious that routing in the Internet cannot be done using a single protocol for two reasons: a scalability problem and an administrative issue. *Scalability problem* means that the size of the forwarding tables becomes huge, searching for a destination in a forwarding table becomes time-consuming, and updating creates a huge amount of traffic. The *administrative issue* is related to the Internet structure described in Figure 20.14. As the figure shows, each ISP is run by an administrative authority. The administrator needs to have control in its system. The organization must be able to use as many subnets and routers as it needs, may desire that the routers be from a particular manufacturer, may wish to run a specific routing algorithm to meet the needs of the organization, and may want to impose some policy on the traffic passing through its ISP.

Hierarchical routing means considering each ISP as an **autonomous system (AS)**. Each AS can run a routing protocol that meets its needs, but the global Internet runs a global protocol to glue all ASs together. The routing protocol run in each AS is referred to as *intra-AS routing protocol*, *intradomain routing protocol*, or *interior gateway protocol (IGP)*; the global routing protocol is referred to as *inter-AS routing protocol*, *interdomain routing protocol*, or *exterior gateway protocol (EGP)*. We can have several intradomain routing protocols, and each AS is free to choose one, but it should be clear that we should have only one interdomain protocol that handles routing between these entities. Presently, the two common intradomain routing protocols are RIP and OSPF; the only interdomain routing protocol is BGP. The situation may change when we move to IPv6.

Autonomous Systems

As we said before, each ISP is an autonomous system when it comes to managing networks and routers under its control. Although we may have small, medium-size, and large ASs, each AS is given an autonomous number (ASN) by the ICANN. Each ASN is a 16-bit unsigned integer that uniquely defines an AS. The autonomous systems, however, are not categorized according to their size; they are categorized according to the way they are connected to other ASs. We have stub ASs, multihomed ASs, and transient ASs. The type, as we see will later, affects the operation of the interdomain routing protocol in relation to that AS.

- **Stub AS.** A stub AS has only one connection to another AS. The data traffic can be either initiated or terminated in a stub AS; the data cannot pass through it. A good example of a stub AS is the customer network, which is either the source or the sink of data.
- **Multihomed AS.** A multihomed AS can have more than one connection to other ASs, but it does not allow data traffic to pass through it. A good example of such an AS is some of the customer ASs that may use the services of more than one provider network, but their policy does not allow data to be passed through them.

- ❑ **Transient AS.** A transient AS is connected to more than one other AS and also allows the traffic to pass through. The provider networks and the backbone are good examples of transient ASs.

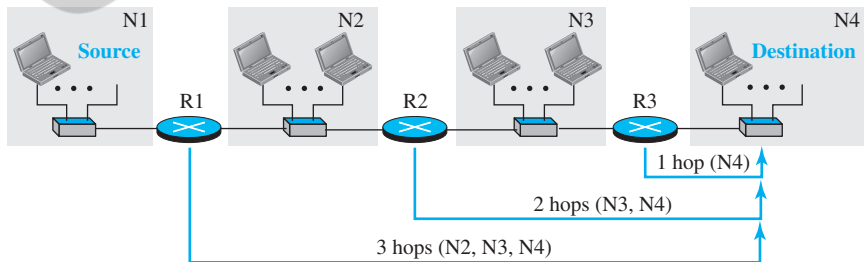
20.3.2 Routing Information Protocol (RIP)

The **Routing Information Protocol (RIP)** is one of the most widely used intradomain routing protocols based on the distance-vector routing algorithm we described earlier. RIP was started as part of the Xerox Network System (XNS), but it was the Berkeley Software Distribution (BSD) version of UNIX that helped make the use of RIP widespread.

Hop Count

A router in this protocol basically implements the distance-vector routing algorithm shown in Table 20.1. However, the algorithm has been modified as described below. First, since a router in an AS needs to know how to forward a packet to different networks (subnets) in an AS, RIP routers advertise the cost of reaching different networks instead of reaching other nodes in a theoretical graph. In other words, the cost is defined between a router and the network in which the destination host is located. Second, to make the implementation of the cost simpler (independent from performance factors of the routers and links, such as delay, bandwidth, and so on), the cost is defined as the number of hops, which means the number of networks (subnets) a packet needs to travel through from the source router to the final destination host. Note that the network in which the source host is connected is not counted in this calculation because the source host does not use a forwarding table; the packet is delivered to the default router. Figure 20.15 shows the concept of hop count advertised by three routers from a source host to a destination host. In RIP, the maximum cost of a path can be 15, which means 16 is considered as infinity (no connection). For this reason, RIP can be used only in autonomous systems in which the diameter of the AS is not more than 15 hops.

Figure 20.15 Hop counts in RIP



Forwarding Tables

Although the distance-vector algorithm we discussed in the previous section is concerned with exchanging distance vectors between neighboring nodes, the routers in an autonomous system need to keep forwarding tables to forward packets to their destination networks. A forwarding table in RIP is a three-column table in which the first column is the address of the destination network, the second column is the address of the next router to which the packet should be forwarded, and the third column is the cost (the number of hops) to reach the destination network. Figure 20.16 shows the three forwarding tables for the routers in Figure 20.15. Note that the first and the third columns together convey the same information as does a distance vector, but the cost shows the number of hops to the destination networks.

Figure 20.16 Forwarding tables

Forwarding table for R1			Forwarding table for R2			Forwarding table for R3		
Destination network	Next router	Cost in hops	Destination network	Next router	Cost in hops	Destination network	Next router	Cost in hops
N1	—	1	N1	R1	2	N1	R2	3
N2	—	1	N2	—	1	N2	R2	2
N3	R2	2	N3	—	1	N3	—	1
N4	R2	3	N4	R3	2	N4	—	1

Although a forwarding table in RIP defines only the next router in the second column, it gives the information about the whole least-cost tree based on the second property of these trees, discussed in the previous section. For example, R1 defines that the next router for the path to N4 is R2; R2 defines that the next router to N4 is R3; R3 defines that there is no next router for this path. The tree is then $R1 \rightarrow R2 \rightarrow R3 \rightarrow N4$.

A question often asked about the forwarding table is what the use of the third column is. The third column is not needed for forwarding the packet, but it is needed for updating the forwarding table when there is a change in the route, as we will see shortly.

RIP Implementation

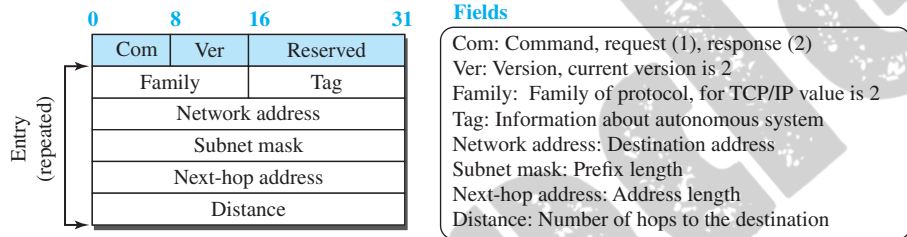
RIP is implemented as a process that uses the service of UDP on the well-known port number 520. In BSD, RIP is a daemon process (a process running in the background), named *routed* (abbreviation for *route daemon* and pronounced *route-dee*). This means that, although RIP is a routing protocol to help IP route its datagrams through the AS, the RIP messages are encapsulated inside UDP user datagrams, which in turn are encapsulated inside IP datagrams. In other words, RIP runs at the application layer, but creates forwarding tables for IP at the network layer.

RIP has gone through two versions: RIP-1 and RIP-2. The second version is backward compatible with the first section; it allows the use of more information in the RIP messages that were set to 0 in the first version. We discuss only RIP-2 in this section.

RIP Messages

Two RIP processes, a client and a server, like any other processes, need to exchange messages. RIP-2 defines the format of the message, as shown in Figure 20.17. Part of the message, which we call *entry*, can be repeated as needed in a message. Each entry carries the information related to one line in the forwarding table of the router that sends the message.

Figure 20.17 RIP message format



RIP has two types of messages: request and response. A request message is sent by a router that has just come up or by a router that has some time-out entries. A request message can ask about specific entries or all entries. A response (or update) message can be either solicited or unsolicited. A solicited response message is sent only in answer to a request message. It contains information about the destination specified in the corresponding request message. An unsolicited response message, on the other hand, is sent periodically, every 30 seconds or when there is a change in the forwarding table.

RIP Algorithm

RIP implements the same algorithm as the distance-vector routing algorithm we discussed in the previous section. However, some changes need to be made to the algorithm to enable a router to update its forwarding table:

- ❑ Instead of sending only distance vectors, a router needs to send the whole contents of its forwarding table in a response message.
- ❑ The receiver adds one hop to each cost and changes the next router field to the address of the sending router. We call each route in the modified forwarding table the *received route* and each route in the old forwarding table the *old route*. The received router selects the old routes as the new ones except in the following three cases:
 1. If the received route does not exist in the old forwarding table, it should be added to the route.
 2. If the cost of the received route is lower than the cost of the old one, the received route should be selected as the new one.
 3. If the cost of the received route is higher than the cost of the old one, but the value of the next router is the same in both routes, the received route should be selected as the new one. This is the case where the route was actually advertised

by the same router in the past, but now the situation has been changed. For example, suppose a neighbor has previously advertised a route to a destination with cost 3, but now there is no path between this neighbor and that destination. The neighbor advertises this destination with cost value infinity (16 in RIP). The receiving router must not ignore this value even though its old route has a lower cost to the same destination.

- ❑ The new forwarding table needs to be sorted according to the destination route (mostly using the longest prefix first).

Example 20.1

Figure 20.18 shows a more realistic example of the operation of RIP in an autonomous system. First, the figure shows all forwarding tables after all routers have been booted. Then we show changes in some tables when some update messages have been exchanged. Finally, we show the stabilized forwarding tables when there is no more change.

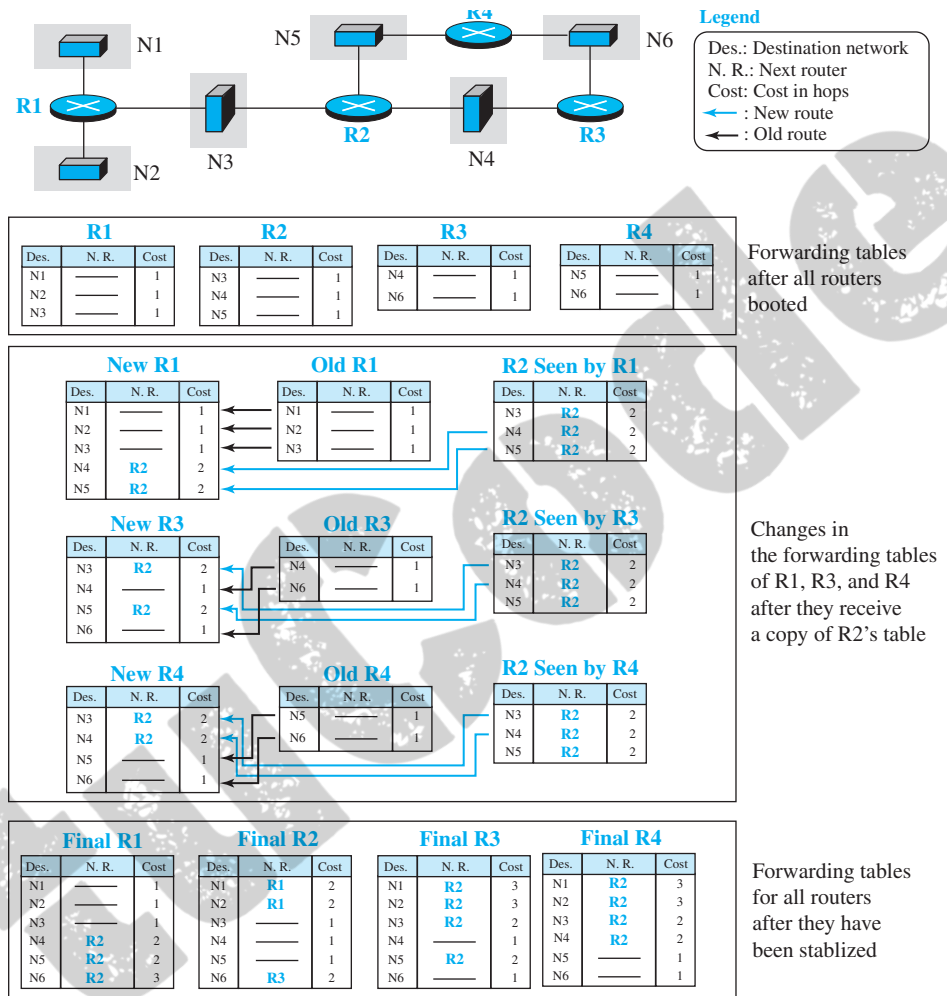
Timers in RIP

RIP uses three timers to support its operation. The *periodic timer* controls the advertising of regular update messages. Each router has one periodic timer that is randomly set to a number between 25 and 35 seconds (to prevent all routers sending their messages at the same time and creating excess traffic). The timer counts down; when zero is reached, the update message is sent, and the timer is randomly set once again. The *expiration timer* governs the validity of a route. When a router receives update information for a route, the expiration timer is set to 180 seconds for that particular route. Every time a new update for the route is received, the timer is reset. If there is a problem on an internet and no update is received within the allotted 180 seconds, the route is considered expired and the hop count of the route is set to 16, which means the destination is unreachable. Every route has its own expiration timer. The *garbage collection timer* is used to purge a route from the forwarding table. When the information about a route becomes invalid, the router does not immediately purge that route from its table. Instead, it continues to advertise the route with a metric value of 16. At the same time, a garbage collection timer is set to 120 seconds for that route. When the count reaches zero, the route is purged from the table. This timer allows neighbors to become aware of the invalidity of a route prior to purging.

Performance

Before ending this section, let us briefly discuss the performance of RIP:

- ❑ **Update Messages.** The update messages in RIP have a very simple format and are sent only to neighbors; they are local. They do not normally create traffic because the routers try to avoid sending them at the same time.
- ❑ **Convergence of Forwarding Tables.** RIP uses the distance-vector algorithm, which can converge slowly if the domain is large, but, since RIP allows only 15 hops in a domain (16 is considered as infinity), there is normally no problem in convergence. The only problems that may slow down convergence are count-to-infinity and loops created in the domain; use of poison-reverse and split-horizon strategies added to the RIP extension may alleviate the situation.

Figure 20.18 Example of an autonomous system using RIP

- ❑ **Robustness.** As we said before, distance-vector routing is based on the concept that each router sends what it knows about the whole domain to its neighbors. This means that the calculation of the forwarding table depends on information received from immediate neighbors, which in turn receive their information from their own neighbors. If there is a failure or corruption in one router, the problem will be propagated to all routers and the forwarding in each router will be affected.

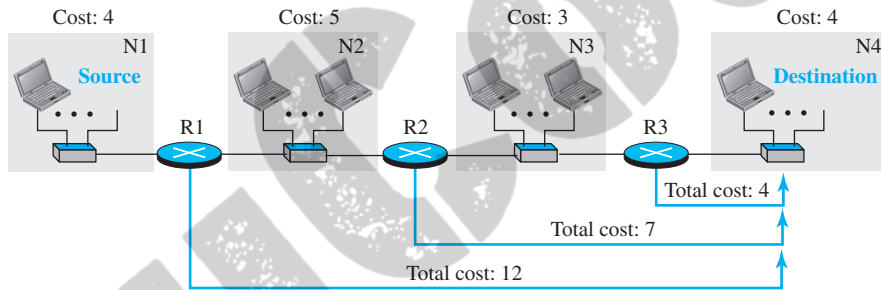
20.3.3 Open Shortest Path First (OSPF)

Open Shortest Path First (OSPF) is also an intradomain routing protocol like RIP, but it is based on the link-state routing protocol we described earlier in the chapter. OSPF is an *open* protocol, which means that the specification is a public document.

Metric

In OSPF, like RIP, the cost of reaching a destination from the host is calculated from the source router to the destination network. However, each link (network) can be assigned a weight based on the throughput, round-trip time, reliability, and so on. An administration can also decide to use the hop count as the cost. An interesting point about the cost in OSPF is that different service types (TOSs) can have different weights as the cost. Figure 20.19 shows the idea of the cost from a router to the destination host network. We can compare the figure with Figure 20.15 for the RIP.

Figure 20.19 *Metric in OSPF*



Forwarding Tables

Each OSPF router can create a forwarding table after finding the shortest-path tree between itself and the destination using Dijkstra's algorithm, described earlier in the chapter. Figure 20.20 shows the forwarding tables for the simple AS in Figure 20.19. Comparing the forwarding tables for the OSPF and RIP in the same AS, we find that the only difference is the cost values. In other words, if we use the hop count for OSPF, the tables will be exactly the same. The reason for this consistency is that both protocols use the shortest-path trees to define the best route from a source to a destination.

Areas

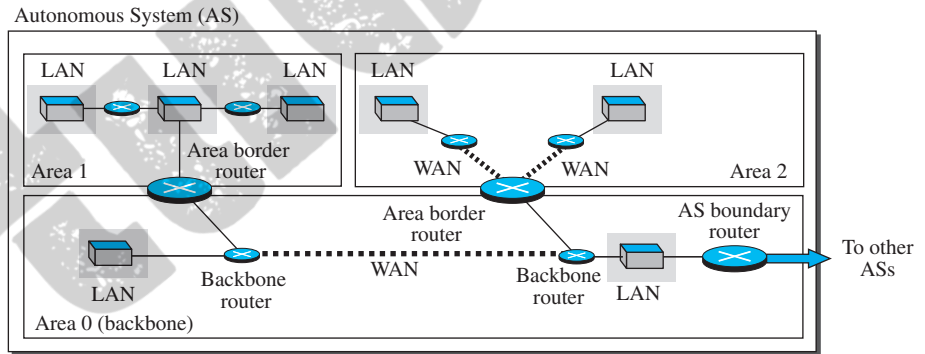
Compared with RIP, which is normally used in small ASs, OSPF was designed to be able to handle routing in a small or large autonomous system. However, the formation of shortest-path trees in OSPF requires that all routers flood the whole AS with their LSPs to create the global LSDB. Although this may not create a problem in a small AS, it may have created a huge volume of traffic in a large AS. To prevent this, the AS needs to be divided into small sections called *areas*. Each area acts as a small independent domain for flooding LSPs. In other words, OSPF uses another level of hierarchy in routing: the first level is the autonomous system, the second is the area.

Figure 20.20 Forwarding tables in OSPF

Forwarding table for R1			Forwarding table for R2			Forwarding table for R3		
Destination network	Next router	Cost	Destination network	Next router	Cost	Destination network	Next router	Cost
N1	—	4	N1	R1	9	N1	R2	12
N2	—	5	N2	—	5	N2	R2	8
N3	R2	8	N3	—	3	N3	—	3
N4	R2	12	N4	R3	7	N4	—	4

However, each router in an area needs to know the information about the link states not only in its area but also in other areas. For this reason, one of the areas in the AS is designated as the *backbone area*, responsible for gluing the areas together. The routers in the backbone area are responsible for passing the information collected by each area to all other areas. In this way, a router in an area can receive all LSPs generated in other areas. For the purpose of communication, each area has an area identification. The area identification of the backbone is zero. Figure 20.21 shows an autonomous system and its areas.

Figure 20.21 Areas in an autonomous system

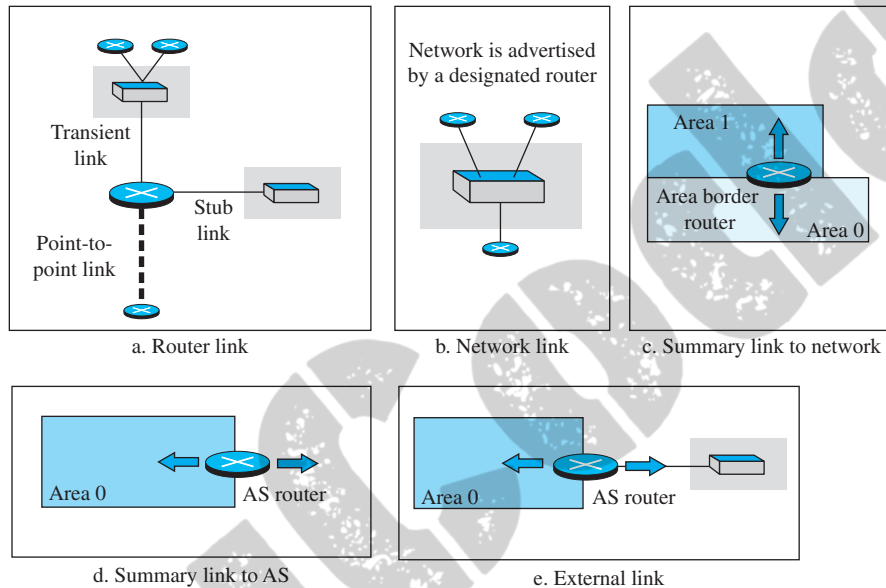


Link-State Advertisement

OSPF is based on the link-state routing algorithm, which requires that a router advertise the state of each link to all neighbors for the formation of the LSDB. When we discussed the link-state algorithm, we used the graph theory and assumed that each router is a node and each network between two routers is an edge. The situation is different in the real world, in which we need to advertise the existence of different entities as nodes, the different types of links that connect each node to its neighbors, and the different types of cost associated with each link. This means we need different types of advertisements, each capable of advertising different situations. We can have five types of

link-state advertisements: *router link*, *network link*, *summary link to network*, *summary link to AS border router*, and *external link*. Figure 20.22 shows these five advertisements and their uses.

Figure 20.22 Five different LSPs



- ❑ **Router link.** A router link advertises the existence of a router as a node. In addition to giving the address of the announcing router, this type of advertisement can define one or more types of links that connect the advertising router to other entities. A *transient link* announces a link to a transient network, a network that is connected to the rest of the networks by one or more routers. This type of advertisement should define the address of the transient network and the cost of the link. A *stub link* advertises a link to a stub network, a network that is not a through network. Again, the advertisement should define the address of the network and the cost. A *point-to-point link* should define the address of the router at the end of the point-to-point line and the cost to get there.
- ❑ **Network link.** A network link advertises the network as a node. However, since a network cannot do announcements itself (it is a passive entity), one of the routers is assigned as the designated router and does the advertising. In addition to the address of the designated router, this type of LSP announces the IP address of all routers (including the designated router as a router and not as speaker of the network), but no cost is advertised because each router announces the cost to the network when it sends a router link advertisement.
- ❑ **Summary link to network.** This is done by an area border router; it advertises the summary of links collected by the backbone to an area or the summary of links

collected by the area to the backbone. As we discussed earlier, this type of information exchange is needed to glue the areas together.

- ❑ **Summary link to AS.** This is done by an AS router that advertises the summary links from other ASs to the backbone area of the current AS, information which later can be disseminated to the areas so that they will know about the networks in other ASs. The need for this type of information exchange is better understood when we discuss inter-AS routing (BGP).
- ❑ **External link.** This is also done by an AS router to announce the existence of a single network outside the AS to the backbone area to be disseminated into the areas.

OSPF Implementation

OSPF is implemented as a program in the network layer, using the service of the IP for propagation. An IP datagram that carries a message from OSPF sets the value of the protocol field to 89. This means that, although OSPF is a routing protocol to help IP to route its datagrams inside an AS, the OSPF messages are encapsulated inside datagrams. OSPF has gone through two versions: version 1 and version 2. Most implementations use version 2.

OSPF Messages

OSPF is a very complex protocol; it uses five different types of messages. In Figure 20.23, we first show the format of the OSPF common header (which is used in all messages) and the link-state general header (which is used in some messages). We then give the outlines of five message types used in OSPF. The *hello* message (type 1) is used by a router to introduce itself to the neighbors and announce all neighbors that it already knows. The *database description* message (type 2) is normally sent in response to the hello message to allow a newly joined router to acquire the full LSDB. The *link-state request* message (type 3) is sent by a router that needs information about a specific LS. The *link-state update* message (type 4) is the main OSPF message used for building the LSDB. This message, in fact, has five different versions (router link, network link, summary link to network, summary link to AS border router, and external link), as we discussed before. The *link-state acknowledgment* message (type 5) is used to create reliability in OSPF; each router that receives a link-state update message needs to acknowledge it.

Authentication

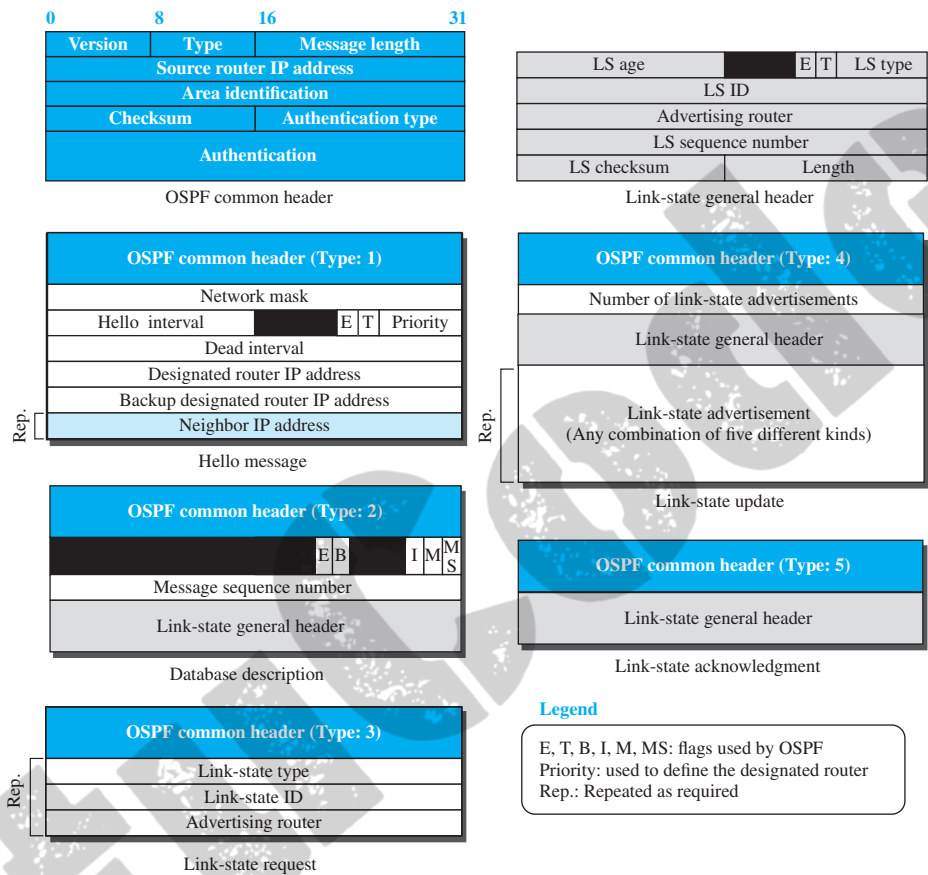
As Figure 20.23 shows, the OSPF common header has the provision for authentication of the message sender. As we will discuss in Chapters 31 and 32, this prevents a malicious entity from sending OSPF messages to a router and causing the router to become part of the routing system to which it actually does not belong.

OSPF Algorithm

OSPF implements the link-state routing algorithm we discussed in the previous section. However, some changes and augmentations need to be added to the algorithm:

- ❑ After each router has created the shortest-path tree, the algorithm needs to use it to create the corresponding routing algorithm.

Figure 20.23 OSPF message formats



Legend

E, T, B, I, M, MS: flags used by OSPF
 Priority: used to define the designated router
 Rep.: Repeated as required

- The algorithm needs to be augmented to handle sending and receiving all five types of messages.

Performance

Before ending this section, let us briefly discuss the performance of OSPF:

- Update Messages.** The link-state messages in OSPF have a somewhat complex format. They also are flooded to the whole area. If the area is large, these messages may create heavy traffic and use a lot of bandwidth.
- Convergence of Forwarding Tables.** When the flooding of LSPs is completed, each router can create its own shortest-path tree and forwarding table; convergence is fairly quick. However, each router needs to run Dijkstra’s algorithm, which may take some time.

- ❑ **Robustness.** The OSPF protocol is more robust than RIP because, after receiving the completed LSDB, each router is independent and does not depend on other routers in the area. Corruption or failure in one router does not affect other routers as seriously as in RIP.

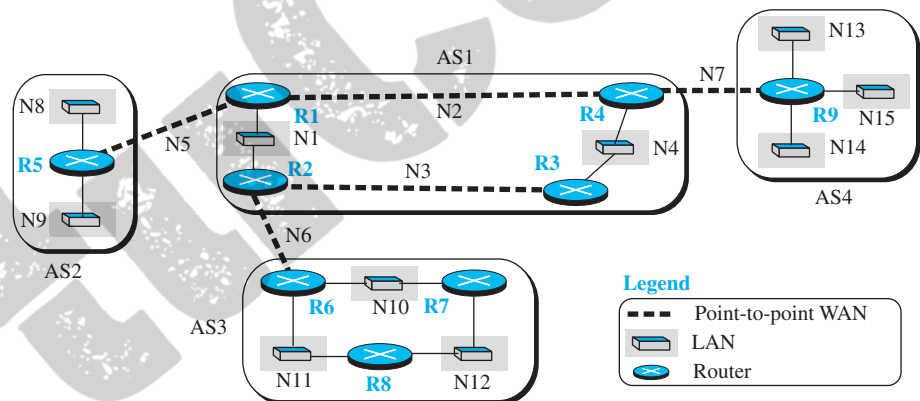
20.3.4 Border Gateway Protocol Version 4 (BGP4)

The **Border Gateway Protocol version 4 (BGP4)** is the only interdomain routing protocol used in the Internet today. BGP4 is based on the path-vector algorithm we described before, but it is tailored to provide information about the reachability of networks in the Internet.

Introduction

BGP, and in particular BGP4, is a complex protocol. In this section, we introduce the basics of BGP and its relationship with intradomain routing protocols (RIP or OSPF). Figure 20.24 shows an example of an internet with four autonomous systems. AS2, AS3, and AS4 are *stub* autonomous systems; AS1 is a *transient* one. In our example, data exchange between AS2, AS3, and AS4 should pass through AS1.

Figure 20.24 A sample internet with four ASs



Each autonomous system in this figure uses one of the two common intradomain protocols, RIP or OSPF. Each router in each AS knows how to reach a network that is in its own AS, but it does not know how to reach a network in another AS.

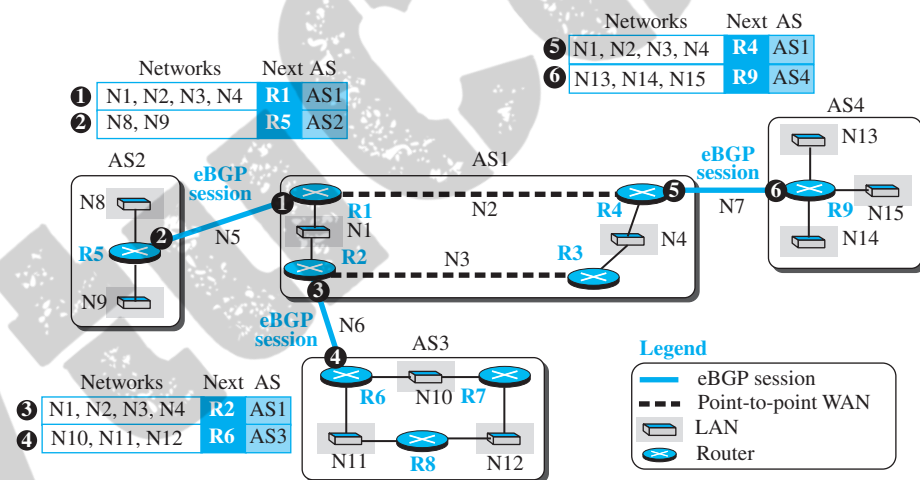
To enable each router to route a packet to any network in the internet, we first install a variation of BGP4, called *external BGP (eBGP)*, on each *border router* (the one at the edge of each AS which is connected to a router at another AS). We then install the second variation of BGP, called *internal BGP (iBGP)*, on all routers. This means that the border routers will be running three routing protocols (intradomain, eBGP, and iBGP), but other routers are running two protocols (intradomain and iBGP). We discuss the effect of each BGP variation separately.

Operation of External BGP (eBGP)

We can say that BGP is a kind of point-to-point protocol. When the software is installed on two routers, they try to create a TCP connection using the well-known port 179. In other words, a pair of client and server processes continuously communicate with each other to exchange messages. The two routers that run the BGP processes are called *BGP peers* or *BGP speakers*. We discuss different types of messages exchanged between two peers, but for the moment we are interested in only the update messages (discussed later) that announce reachability of networks in each AS.

The eBGP variation of BGP allows two physically connected border routers in two different ASs to form pairs of eBGP speakers and exchange messages. The routers that are eligible in our example in Figure 20.24 form three pairs: R1-R5, R2-R6, and R4-R9. The connection between these pairs is established over three physical WANs (N5, N6, and N7). However, there is a need for a logical TCP connection to be created over the physical connection to make the exchange of information possible. Each logical connection in BGP parlance is referred to as a *session*. This means that we need three sessions in our example, as shown in Figure 20.25.

Figure 20.25 eBGP operation



The figure also shows the simplified update messages sent by routers involved in the eBGP sessions. The circled number defines the sending router in each case. For example, message number 1 is sent by router R1 and tells router R5 that N1, N2, N3, and N4 can be reached through router R1 (R1 gets this information from the corresponding intradomain forwarding table). Router R5 can now add these pieces of information at the end of its forwarding table. When R5 receives any packet destined for these four networks, it can use its forwarding table and find that the next router is R1.

The reader may have noticed that the messages exchanged during three eBGP sessions help some routers know how to route packets to some networks in the internet, but

the reachability information is not complete. There are two problems that need to be addressed:

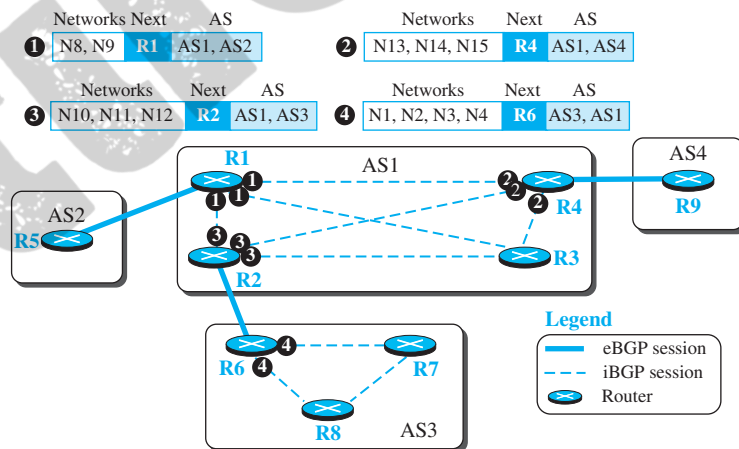
1. Some border routers do not know how to route a packet destined for nonneighbor ASs. For example, R5 does not know how to route packets destined for networks in AS3 and AS4. Routers R6 and R9 are in the same situation as R5: R6 does not know about networks in AS2 and AS4; R9 does not know about networks in AS2 and AS3.
2. None of the nonborder routers know how to route a packet destined for any networks in other ASs.

To address the above two problems, we need to allow all pairs of routers (border or nonborder) to run the second variation of the BGP protocol, iBGP.

Operation of Internal BGP (iBGP)

The iBGP protocol is similar to the eBGP protocol in that it uses the service of TCP on the well-known port 179, but it creates a session between any possible pair of routers inside an autonomous system. However, some points should be made clear. First, if an AS has only one router, there cannot be an iBGP session. For example, we cannot create an iBGP session inside AS2 or AS4 in our internet. Second, if there are n routers in an autonomous system, there should be $[n \times (n - 1) / 2]$ iBGP sessions in that autonomous system (a fully connected mesh) to prevent loops in the system. In other words, each router needs to advertise its own reachability to the peer in the session instead of flooding what it receives from another peer in another session. Figure 20.26 shows the combination of eBGP and iBGP sessions in our internet.

Figure 20.26 Combination of eBGP and iBGP sessions in our internet



Note that we have not shown the physical networks inside ASs because a session is made on an overlay network (TCP connection), possibly spanning more than one physical network as determined by the route dictated by intradomain routing protocol. Also note that in this stage only four messages are exchanged. The first message (numbered 1) is sent by R1 announcing that networks N8 and N9 are reachable through the

path AS1-AS2, but the next router is R1. This message is sent, through separate sessions, to R2, R3, and R4. Routers R2, R4, and R6 do the same thing but send different messages to different destinations. The interesting point is that, at this stage, R3, R7, and R8 create sessions with their peers, but they actually have no message to send.

The updating process does not stop here. For example, after R1 receives the update message from R2, it combines the reachability information about AS3 with the reachability information it already knows about AS1 and sends a new update message to R5. Now R5 knows how to reach networks in AS1 and AS3. The process continues when R1 receives the update message from R4. The point is that we need to make certain that at a point in time there are no changes in the previous updates and that all information is propagated through all ASs. At this time, each router combines the information received from eBGP and iBGP and creates what we may call a path table after applying the criteria for finding the best path, including routing policies that we discuss later. To demonstrate, we show the path tables in Figure 20.27 for the routers in Figure 20.24. For example, router R1 now knows that any packet destined for networks N8 or N9 should go through AS1 and AS2 and the next router to deliver the packet to is router R5. Similarly, router R4 knows that any packet destined for networks N10, N11, or N12 should go through AS1 and AS3 and the next router to deliver this packet to is router R1, and so on.

Figure 20.27 Finalized BGP path tables

Networks	Next	Path	Networks	Next	Path	Networks	Next	Path
N8, N9	R5	AS1, AS2	N8, N9	R1	AS1, AS2	N8, N9	R2	AS1, AS2
N10, N11, N12	R2	AS1, AS3	N10, N11, N12	R6	AS1, AS3	N10, N11, N12	R2	AS1, AS3
N13, N14, N15	R4	AS1, AS4	N13, N14, N15	R1	AS1, AS4	N13, N14, N15	R4	AS1, AS4

Path table for R1			Path table for R2			Path table for R3		
Networks	Next	Path	Networks	Next	Path	Networks	Next	Path
N8, N9	R1	AS1, AS2	N1, N2, N3, N4	R1	AS2, AS1	N1, N2, N3, N4	R2	AS3, AS1
N10, N11, N12	R1	AS1, AS3	N10, N11, N12	R1	AS2, AS1, AS3	N8, N9	R2	AS3, AS1, AS2
N13, N14, N15	R9	AS1, AS4	N13, N14, N15	R1	AS2, AS1, AS4	N13, N14, N15	R2	AS3, AS1, AS4

Path table for R4			Path table for R5			Path table for R6		
Networks	Next	Path	Networks	Next	Path	Networks	Next	Path
N1, N2, N3, N4	R6	AS3, AS1	N1, N2, N3, N4	R6	AS3, AS1	N1, N2, N3, N4	R4	AS4, AS1
N8, N9	R6	AS3, AS1, AS2	N8, N9	R6	AS3, AS1, AS2	N8, N9	R4	AS4, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4	N13, N14, N15	R6	AS3, AS1, AS4	N10, N11, N12	R4	AS4, AS1, AS3

Path table for R7			Path table for R8			Path table for R9		
Networks	Next	Path	Networks	Next	Path	Networks	Next	Path
N1, N2, N3, N4	R6	AS3, AS1	N1, N2, N3, N4	R6	AS3, AS1	N1, N2, N3, N4	R4	AS4, AS1
N8, N9	R6	AS3, AS1, AS2	N8, N9	R6	AS3, AS1, AS2	N8, N9	R4	AS4, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4	N13, N14, N15	R6	AS3, AS1, AS4	N10, N11, N12	R4	AS4, AS1, AS3

Injection of Information into Intradomain Routing

The role of an interdomain routing protocol such as BGP is to help the routers inside the AS to augment their routing information. In other words, the path tables collected and organized by BGP are not used, per se, for routing packets; they are injected into intradomain forwarding tables (RIP or OSPF) for routing packets. This can be done in several ways depending on the type of AS.

In the case of a stub AS, the only area border router adds a default entry at the end of its forwarding table and defines the next router to be the speaker router at the end of the eBGP connection. In Figure 20.24, R5 in AS2 defines R1 as the default router for

all networks other than N8 and N9. The situation is the same for router R9 in AS4 with the default router to be R4. In AS3, R6 set its default router to be R2, but R7 and R8 set their default router to be R6. These settings are in accordance with the path tables we describe in Figure 20.27 for these routers. In other words, the path tables are injected into intradomain forwarding tables by adding only one default entry.

In the case of a transient AS, the situation is more complicated. R1 in AS1 needs to inject the whole contents of the path table for R1 in Figure 20.27 into its intradomain forwarding table. The situation is the same for R2, R3, and R4.

One issue to be resolved is the cost value. We know that RIP and OSPF use different metrics. One solution, which is very common, is to set the cost to the foreign networks at the same cost value as to reach the first AS in the path. For example, the cost for R5 to reach all networks in other ASs is the cost to reach N5. The cost for R1 to reach networks N10 to N12 is the cost to reach N6, and so on. The cost is taken from the intradomain forwarding tables (RIP or OSPF).

Figure 20.28 shows the interdomain forwarding tables. For simplicity, we assume that all ASs are using RIP as the intradomain routing protocol. The shaded areas are the augmentation injected by the BGP protocol; the default destinations are indicated as zero.

Figure 20.28 Forwarding tables after injection from BGP

Des.	Next	Cost	Des.	Next	Cost	Des.	Next	Cost	Des.	Next	Cost
N1	—	1	N1	—	1	N1	R2	2	N1	R1	2
N4	R4	2	N4	R3	2	N4	—	1	N4	—	1
N8	R5	1	N8	R1	2	N8	R2	3	N8	R1	2
N9	R5	1	N9	R1	2	N9	R2	3	N9	R1	2
N10	R2	2	N10	R6	1	N10	R2	2	N10	R3	3
N11	R2	2	N11	R6	1	N11	R2	2	N11	R3	3
N12	R2	2	N12	R6	1	N12	R2	2	N12	R3	3
N13	R4	2	N13	R3	3	N13	R4	2	N13	R9	1
N14	R4	2	N14	R3	3	N14	R4	2	N14	R9	1
N15	R4	2	N15	R3	3	N15	R4	2	N15	R9	1
Table for R1			Table for R2			Table for R3			Table for R4		
Des.	Next	Cost	Des.	Next	Cost	Des.	Next	Cost	Des.	Next	Cost
N8	—	1	N10	—	1	N10	—	1	N10	R6	2
N9	—	1	N11	—	1	N11	R6	2	N11	—	1
0	R1	1	N12	R7	2	N12	—	1	N12	—	1
Table for R5			0	R2	1	0	R6	2	0	R4	1
Table for R6			Table for R7			Table for R8			Table for R9		

Address Aggregation

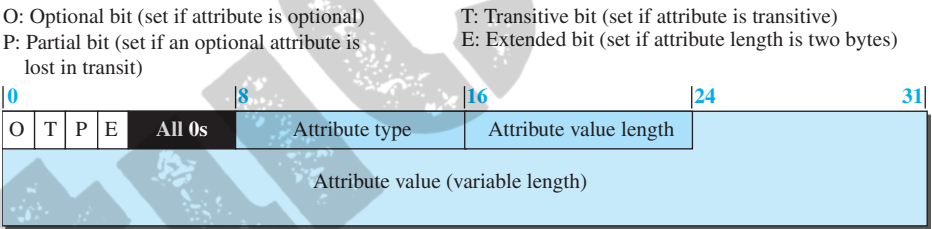
The reader may have realized that intradomain forwarding tables obtained with the help of the BGP4 protocols may become huge in the case of the global Internet because many destination networks may be included in a forwarding table. Fortunately, BGP4 uses the prefixes as destination identifiers and allows the aggregation of these prefixes, as we discussed in Chapter 18. For example, prefixes 14.18.20.0/26, 14.18.20.64/26, 14.18.20.128/26, and 14.18.20.192/26, can be combined into 14.18.20.0/24 if all four

subnets can be reached through one path. Even if one or two of the aggregated prefixes need a separate path, the longest prefix principle we discussed earlier allows us to do so.

Path Attributes

In both intradomain routing protocols (RIP or OSPF), a destination is normally associated with two pieces of information: next hop and cost. The first one shows the address of the next router to deliver the packet; the second defines the cost to the final destination. Inter-domain routing is more involved and naturally needs more information about how to reach the final destination. In BGP these pieces are called *path attributes*. BGP allows a destination to be associated with up to seven path attributes. Path attributes are divided into two broad categories: *well-known* and *optional*. A well-known attribute must be recognized by all routers; an optional attribute need not be. A well-known attribute can be mandatory, which means that it must be present in any BGP update message, or discretionary, which means it does not have to be. An optional attribute can be either transitive, which means it can pass to the next AS, or intransitive, which means it cannot. All attributes are inserted after the corresponding destination prefix in an update message (discussed later). The format for an attribute is shown in Figure 20.29.

Figure 20.29 *Format of path attribute*



The first byte in each attribute defines the four attribute flags (as shown in the figure). The next byte defines the type of attributes assigned by ICANN (only seven types have been assigned, as explained next). The attribute value length defines the length of the attribute value field (not the length of the whole attributes section). The following gives a brief description of each attribute.

- ❑ **ORIGIN (type 1).** This is a well-known mandatory attribute, which defines the source of the routing information. This attribute can be defined by one of the three values: 1, 2, and 3. Value 1 means that the information about the path has been taken from an intradomain protocol (RIP or OSPF). Value 2 means that the information comes from BGP. Value 3 means that it comes from an unknown source.
- ❑ **AS-PATH (type 2).** This is a well-known mandatory attribute, which defines the list of autonomous systems through which the destination can be reached. We have used this attribute in our examples. The AS-PATH attribute, as we discussed in path-vector routing in the last section, helps prevent a loop. Whenever an update

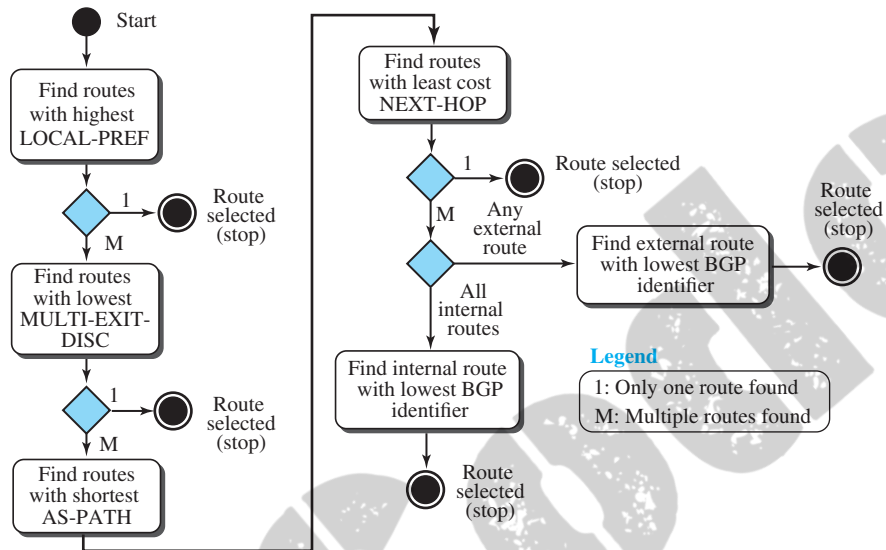
message arrives at a router that lists the current AS as the path, the router drops that path. The AS-PATH can also be used in route selection.

- ❑ **NEXT-HOP (type 3).** This is a well-known mandatory attribute, which defines the next router to which the data packet should be forwarded. We have also used this attribute in our examples. As we have seen, this attribute helps to inject path information collected through the operations of eBGP and iBGP into the intradomain routing protocols such as RIP or OSPF.
- ❑ **MULT-EXIT-DISC (type 4).** The multiple-exit discriminator is an optional intransitive attribute, which discriminates among multiple exit paths to a destination. The value of this attribute is normally defined by the metric in the corresponding intradomain protocol (an attribute value of 4-byte unsigned integer). For example, if a router has multiple paths to the destination with different values related to these attributes, the one with the lowest value is selected. Note that this attribute is intransitive, which means that it is not propagated from one AS to another.
- ❑ **LOCAL-PREF (type 5).** The local preference attribute is a well-known discretionary attribute. It is normally set by the administrator, based on the organization policy. The routes the administrator prefers are given a higher local preference value (an attribute value of 4-byte unsigned integer). For example, in an internet with five ASs, the administrator of AS1 can set the local preference value of 400 to the path AS1 → AS2 → AS5, the value of 300 to AS1 → AS3 → AS5, and the value of 50 to AS1 → AS4 → AS5. This means that the administrator prefers the first path to the second one and prefers the second one to the third one. This may be a case where AS2 is the most secured and AS4 is the least secured AS for the administration of AS1. The last route should be selected if the other two are not available.
- ❑ **ATOMIC-AGGREGATE (type 6).** This is a well-known discretionary attribute, which defines the destination prefix as not aggregate; it only defines a single destination network. This attribute has no value field, which means the value of the length field is zero.
- ❑ **AGGREGATOR (type 7).** This is an optional transitive attribute, which emphasizes that the destination prefix is an aggregate. The attribute value gives the number of the last AS that did the aggregation followed by the IP address of the router that did so.

Route Selection

So far in this section, we have been silent about how a route is selected by a BGP router mostly because our simple example has one route to a destination. In the case where multiple routes are received to a destination, BGP needs to select one among them. The route selection process in BGP is not as easy as the ones in the intradomain routing protocol that is based on the shortest-path tree. A route in BGP has some attributes attached to it and it may come from an eBGP session or an iBGP session. Figure 20.30 shows the flow diagram as used by common implementations.

The router extracts the routes which meet the criteria in each step. If only one route is extracted, it is selected and the process stops; otherwise, the process continues with the next step. Note that the first choice is related to the LOCAL-PREF attribute, which reflects the policy imposed by the administration on the route.

Figure 20.30 Flow diagram for route selection

Messages

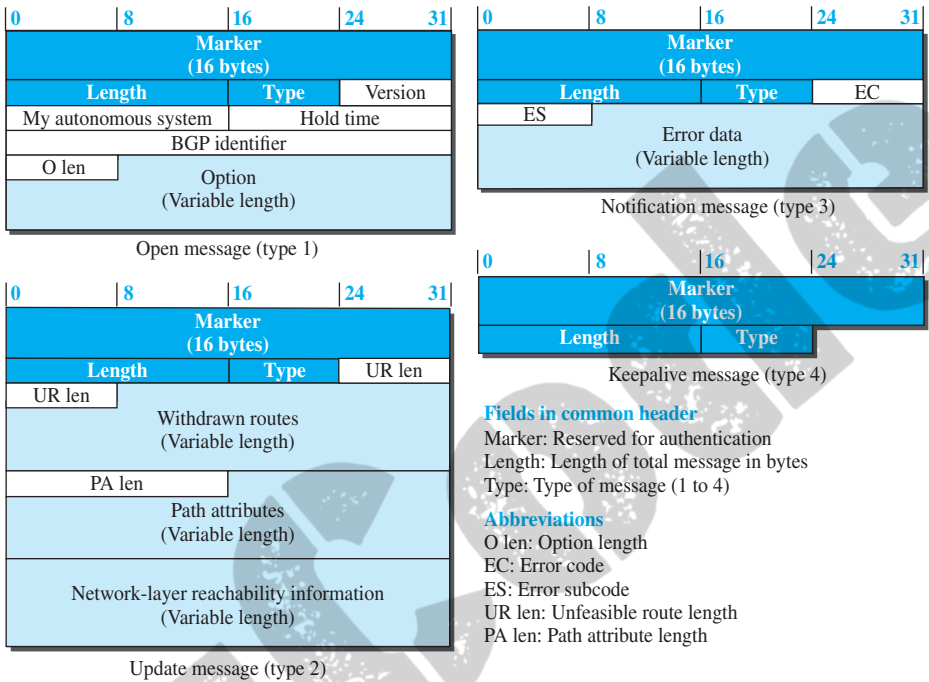
BGP uses four types of messages for communication between the BGP speakers across the ASs and inside an AS: *open*, *update*, *keepalive*, and *notification* (see Figure 20.31). All BGP packets share the same common header.

- ❑ **Open Message.** To create a neighborhood relationship, a router running BGP opens a TCP connection with a neighbor and sends an *open message*.
- ❑ **Update Message.** The *update message* is the heart of the BGP protocol. It is used by a router to withdraw destinations that have been advertised previously, to announce a route to a new destination, or both. Note that BGP can withdraw several destinations that were advertised before, but it can only advertise one new destination (or multiple destinations with the same path attributes) in a single update message.
- ❑ **Keepalive Message.** The BGP peers that are running exchange keepalive messages regularly (before their hold time expires) to tell each other that they are alive.
- ❑ **Notification.** A notification message is sent by a router whenever an error condition is detected or a router wants to close the session.

Performance

BGP performance can be compared with RIP. BGP speakers exchange a lot of messages to create forwarding tables, but BGP is free from loops and count-to-infinity. The same weakness we mention for RIP about propagation of failure and corruption also exists in BGP.

Figure 20.31 BGP messages



21.3.2 Multicast Link State (MOSPF)

Multicast Open Shortest Path First (MOSPF) is the extension of the Open Shortest Path First (OSPF) protocol, which is used in unicast routing. It also uses the source-based tree approach to multicasting. If the internet is running a unicast link-state routing algorithm, the idea can be extended to provide a multicast link-state routing algorithm. Recall that in unicast link-state routing, each router in the internet has a link-state database (LSDB) that can be used to create a shortest-path tree. To extend unicasting to multicasting, each router needs to have another database, as with the case of unicast distance-vector routing, to show which interface has an active member in a particular group. Now a router goes through the following steps to forward a multicast packet received from source S and to be sent to destination G (a group of recipients):

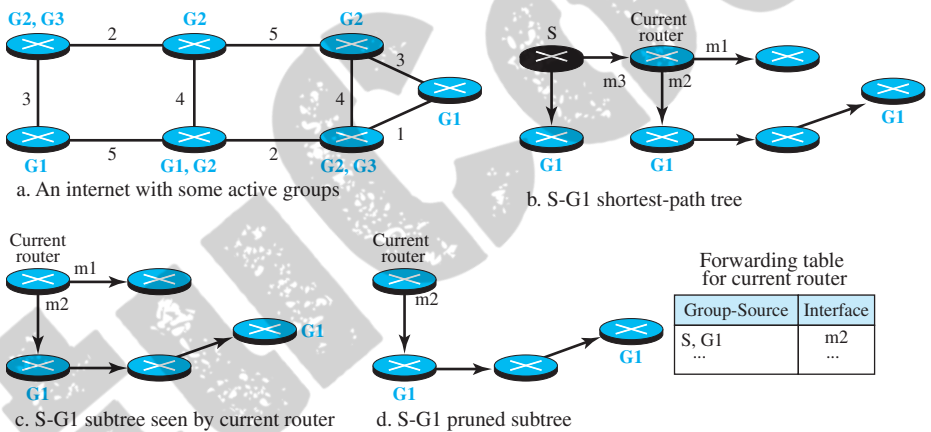
1. The router uses the Dijkstra algorithm to create a shortest-path tree with S as the root and all destinations in the internet as the leaves. Note that this shortest-path tree is different from the one the router normally uses for unicast forwarding, in which the root of the tree is the router itself. In this case, the root of the tree is the source of the packet defined in the source address of the packet. The router is capable of creating this tree because it has the LSDB, the whole topology of the internet; the Dijkstra algorithm can be used to create a tree with any root, no matter which router is using it. The point we need to remember is that the shortest-path tree created this way depends on the specific source. For each source we need to create a different tree.
2. The router finds itself in the shortest-path tree created in the first step. In other words, the router creates a shortest-path subtree with itself as the root of the subtree.
3. The shortest-path subtree is actually a broadcast subtree with the router as the root and all networks as the leaves. The router now uses a strategy similar to the one we

describe in the case of DVMRP to prune the broadcast tree and to change it to a multicast tree. The IGMP protocol is used to find the information at the leaf level. MOSPF has added a new type of link state update packet that floods the membership to all routers. The router can use the information it receives in this way and prune the broadcast tree to make the multicast tree.

- 4. The router can now forward the received packet out of only those interfaces that correspond to the branches of the multicast tree. We need to make certain that a copy of the multicast packet reaches all networks that have active members of the group and that it does not reach those networks that do not.

Figure 21.13 shows an example of using the steps to change a graph to a multicast tree. For simplicity, we have not shown the network, but we added the groups to each router. The figure shows how a source-based tree is made with the source as the root and changed to a multicast subtree with the root at the current router.

Figure 21.13 *Example of tree formation in MOSPF*



21.3.3 Protocol Independent Multicast (PIM)

Protocol Independent Multicast (PIM) is the name given to a common protocol that needs a unicast routing protocol for its operation, but the unicast protocol can be either a distance-vector protocol or a link-state protocol. In other words, PIM needs to use the forwarding table of a unicast routing protocol to find the next router in a path to the destination, but it does not matter how the forwarding table is created. PIM has another interesting feature: it can work in two different modes: dense and sparse. The term *dense* here means that the number of active members of a group in the internet is large; the probability that a router has a member in a group is high. This may happen, for example, in a popular teleconference that has a lot of members. The term *sparse*, on the other hand, means that only a few routers in the internet have active members in the group; the probability that a router has a member of the group is low. This may happen, for example, in a very technical teleconference where a number of members are spread

somewhere in the internet. When the protocol is working in the dense mode, it is referred to as PIM-DM; when it is working in the sparse mode, it is referred to as PIM-SM. We explain both protocols next.

Protocol Independent Multicast-Dense Mode (PIM-DM)

When the number of routers with attached members is large relative to the number of routers in the internet, PIM works in the dense mode and is called **PIM-DM**. In this mode, the protocol uses a source-based tree approach and is similar to DVMRP, but simpler. PIM-DM uses only two strategies described in DVMRP: RPF and RPM. But unlike DVMRP, forwarding of a packet is not suspended awaiting pruning of the first subtree. Let us explain the two steps used in PIM-DM to clear the matter.

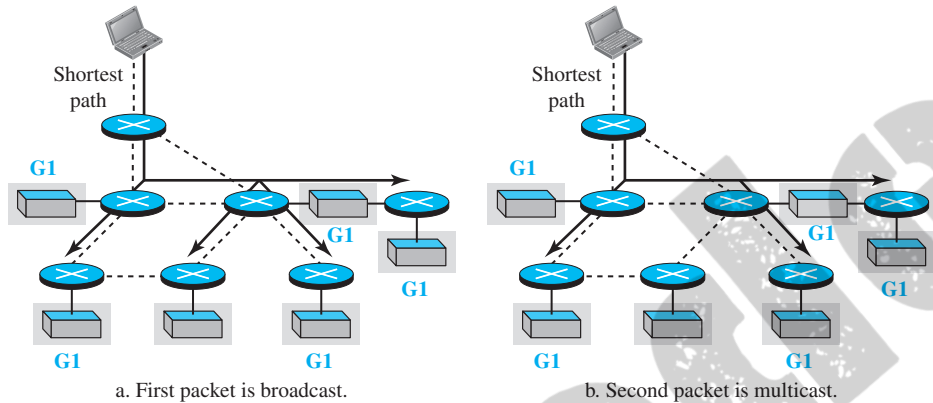
1. A router that has received a multicast packet from the source *S* destined for the group *G* first uses the RPF strategy to avoid receiving a duplicate of the packet. It consults the forwarding table of the underlying unicast protocol to find the next router if it wants to send a message to the source *S* (in the reverse direction). If the packet has not arrived from the next router in the reverse direction, it drops the packet and sends a prune message in that direction to prevent receiving future packets related to (*S*, *G*).
2. If the packet in the first step has arrived from the next router in the reverse direction, the receiving router forwards the packet from all its interfaces except the one from which the packet has arrived and the interface from which it has already received a prune message related to (*S*, *G*). Note that this is actually a broadcasting instead of a multicasting if the packet is the first packet from the source *S* to group *G*. However, each router downstream that receives an unwanted packet sends a prune message to the router upstream, and eventually the broadcasting is changed to multicasting. Note that DVMRP behaves differently: it requires that the prune messages (which are part of DV packets) arrive and the tree is pruned before sending any message through unpruned interfaces. PIM-DM does not care about this precaution because it assumes that most routers have an interest in the group (the idea of the dense mode).

Figure 21.14 shows the idea behind PIM-DM. The first packet is broadcast to all networks, which have or do not have members. After a prune message arrives from a router with no member, the second packet is only multicast.

Protocol Independent Multicast-Sparse Mode (PIM-SM)

When the number of routers with attached members is small relative to the number of routers in the internet, PIM works in the sparse mode and is called **PIM-SM**. In this environment, the use of a protocol that broadcasts the packets until the tree is pruned is not justified; PIM-SM uses a group-shared tree approach to multicasting. The core router in PIM-SM is called the *rendezvous point* (RP). Multicast communication is achieved in two steps. Any router that has a multicast packet to send to a group of destinations first encapsulates the multicast packet in a unicast packet (tunneling) and sends it to the RP. The RP then decapsulates the unicast packet and sends the multicast packet to its destination.

PIM-SM uses a complex algorithm to select one router among all routers in the internet as the RP for a specific group. This means that if we have *m* active groups, we need *m* RPs, although a router may serve more than one group. After the RP for each

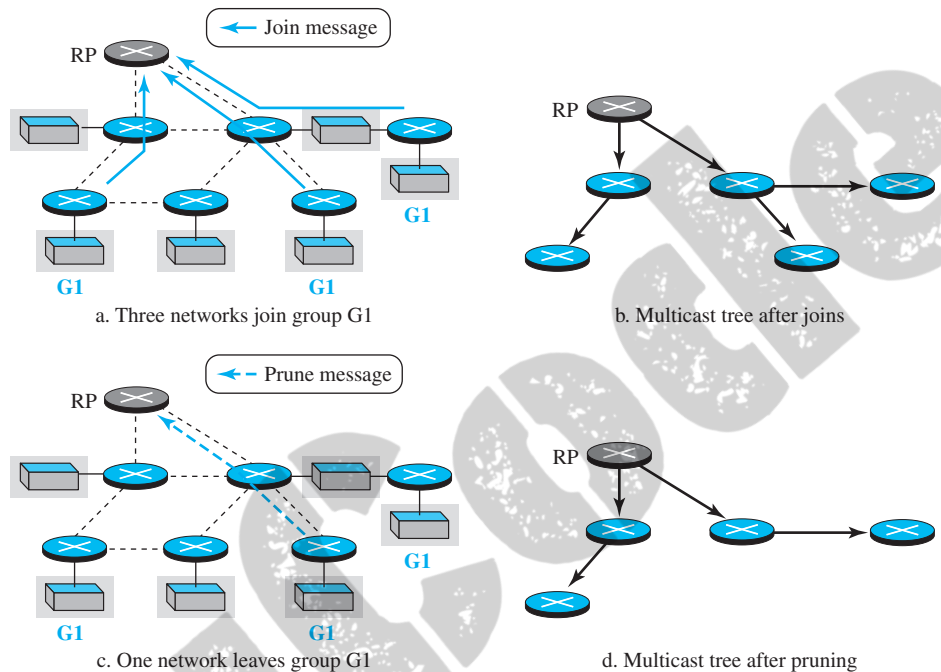
Figure 21.14 Idea behind PIM-DM

group is selected, each router creates a database and stores the group identifier and the IP address of the RP for tunneling multicast packets to it.

PIM-SM uses a spanning multicast tree rooted at the RP with leaves pointing to designated routers connected to each network with an active member. A very interesting point in PIM-SM is the formation of the multicast tree for a group. The idea is that each router helps to create the tree. The router should know the unique interface from which it should accept a multicast packet destined for a group (what was achieved by RPF in DVMRP). The router should also know the interface or interfaces from which it should send out a multicast packet destined for a group (what was achieved by RPM in DVMRP). To avoid delivering more than one copy of the same packet to a network through several routers (what was achieved by RPB in DVMRP), PIM-SM requires that only designated routers send PIM-SM messages, as we will see shortly.

To create a multicast tree rooted at the RP, PIM-SM uses *join* and *prune* messages. Figure 21.15 shows the operation of join and prune messages in PIM-SM. First, three networks join group G1 and form a multicast tree. Later, one of the networks leaves the group and the tree is pruned.

The join message is used to add possible new branches to the tree; the **prune message** is used to cut branches that are not needed. When a designated router finds out that a network has a new member in the corresponding group (via IGMP), it sends a join message in a unicast packet destined for the RP. The packet travels through the unicast shortest-path tree to reach the RP. Any router in the path receives and forwards the packet, but at the same time, the router adds two pieces of information to its multicast forwarding table. The number of the interface through which the join message has arrived is marked (if not already marked) as one of the interfaces through which the multicast packet destined for the group should be sent out in the future. The router also adds a count to the number of join messages received here, as we discuss shortly. The number of the interface through which the join message was sent to the RP is marked (if not already marked) as the only interface through which the multicast packet destined for the same group should be received. In this way, the first join message sent by a

Figure 21.15 *Idea behind PIM-SM*

designated router creates a path from the RP to one of the networks with group members.

To avoid sending multicast packets to networks with no members, PIM-SM uses the prune message. Each designated router that finds out (via IGMP) that there is no active member in its network, sends a prune message to the RP. When a router receives a prune message, it decrements the join count for the interface through which the message has arrived and forwards it to the next router. When the join count for an interface reaches zero, that interface is not part of the multicast tree anymore.