UNIT-2

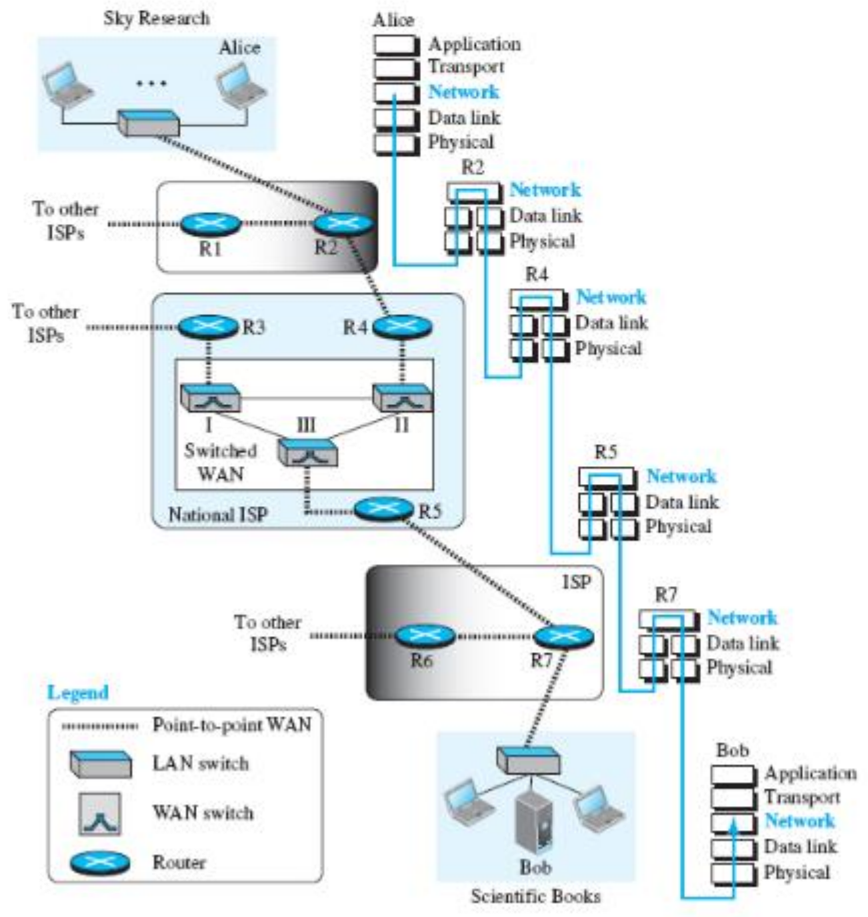
Network Layer: Data Transfer: Services, Packet Switching, Performance, IPv4: IPv4 Addressing, Main and Auxiliary Protocols-IP datagram Packet. IPv6: IPv6 Addressing, The IPv6 Protocol. (Chapter 7)

Network Layer: Routing of Packets: General Idea, Least-Cost Routing, Routing Algorithms: Distance-Vector Routing, Link-State Routing, Path-Vector Routing, OSPF, BGP4, Multicast Routing: Unicasting, Multicasting, Distance Vector Multicast Routing Protocol. IGMP. (Chapter 8)

CHAPTER 7
NETWORK LAYER: DATA TRANSFER

Figure 7.1 shows the communication between Alice and Bob at the network layer. At the source host, the network layer encapsulates data received from the transport layer in a network layer **packet.** At the destination host, the network layer decapsulates data from the network layer and delivers it to the transport layer. The routers do not do any encapsulation or decapsulation unless in some special cases when the packet needs to be fragmented.

Figure 7.1 *Communication at the network layer*



7.1 SERVICES

We briefly discuss the **services provided at the network layer**.

7.1.1 Packetizing

The first duty of the network layer is definitely **packetizing**: encapsulating the payload (data received from the 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 **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.

7.1.2 Routing

The network layer is responsible for routing a network-layer 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.

7.1.3 Error Control

In Chapter 3, 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 the whole datagram. This checksum may prevent any changes or corruptions in the header of the datagram between two hops and from end to end.

7.1.4 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.**

7.1.5 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.

7.1.6 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.

7.1.7 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 13.

7.2 PACKET SWITCHING

From the discussion of routing and forwarding in Section 7.1, 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 communications 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.*

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 next.

7.2.1 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.

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 source or from a different source.

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.

7.2.2 Virtual-Circuit Approach: Connection-Oriented Service

In a connection-oriented service (also called a *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. 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.

7.3 PERFORMANCE

The upper-layer protocols that use the service of the network layer expect to receive an ideal service, but the network layer is not perfect. **The performance of a network can be measured in terms of *delay*, *throughput*, and *packet loss*.** Congestion control is an issue that can improve the performance.

7.3.1 Delay

All of us expect an instantaneous response from a network, but a packet, from its source to its destination, encounters delays. **The delays in a network can be divided into four types: transmission delay, propagation delay, processing delay, and queuing delay**. Let us first discuss each of these delay types and then show how to calculate a packet delay from the source to the destination.

Transmission Delay

A source host or a router cannot send a packet instantaneously. **A sender needs to put the bits in a packet on the line one by one.** If the first bit of the packet is put on the line at time t_1 and the last bit is put on the line at time t_2, transmission delay of the packet is $(t_2 - t_1)$. Definitely, the transmission delay is longer for a longer packet and shorter if the sender can transmit faster. In other words, the transmission delay is

Delay$_{tr}$ = (packet length) / (transmission rate)

Propagation Delay

Propagation delay is **the time it takes for a bit to travel from point A to point B in the transmission media.** The propagation delay for a packet-switched network depends on the propagation delay of each network (LAN or WAN). The propagation delay depends on the propagation speed of the media, which is 3×10^8 m/s in a vacuum and normally much less in a wired medium; it also depends on the distance of the link. In other words, propagation delay is

Delay$_{pg}$ = (distance) / (propagation speed)

Processing Delay

The processing delay is the **time required for a router or a destination host to receive a packet from its input port, remove the header, perform an error-detection procedure, and deliver the packet to the output port (in the case of a router) or deliver the packet to the upper-layer protocol** (in the case of the destination host). The processing delay may be different for each packet, but normally is calculated as an average.

Delay$_{pr}$ = time required to process a packet in a router or a destination host

Queuing Delay

Queuing delay can normally happen in a router. As we discuss in Chapter 8, a router has an input queue connected to each of its input ports to store packets waiting to be processed; the router also has an output queue connected to each of its output ports to store packets waiting to be transmitted. **The queuing delay for a packet in a router is measured as the time a packet waits in the input queue and output queue of a router.** We can compare the situation with a busy airport. Some planes may need to wait to get the landing band (input delay); some planes may need to wait to get the departure band (output delay).

Delay$_{qu}$ = time a packet waits in input and output queues in a router

Total Delay

Assuming equal delays for the sender, routers, and receiver, the total delay (source-to-destination delay) a packet encounters can be calculated if we know the number of routers, n, in the whole path.

Total delay = $(n + 1)$ (delay$_{tr}$ + delay$_{pg}$ + delay$_{pr)}$)+ (n) (delay)$_{qu)}$

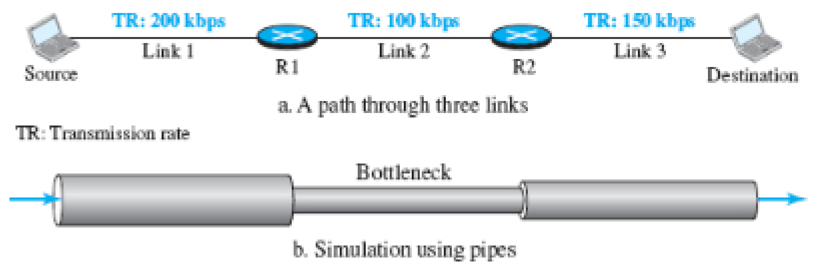
Note that if we have *n* routers, we have (*n* + 1) links. Therefore, we have (*n* + 1) transmission delays related to *n* routers and the source, (*n* + 1) propagation delays related to (*n* + 1) links, (*n* + 1) processing delays related to *n* routers and the destination, and only *n* queuing delays related to *n* routers.

7.3.2 Throughput

Throughput at any point in a network is defined as the number of bits passing through the point in a second, which is actually the **transmission rate** of data at that point. In a path from source to destination, a packet may pass through several links (networks), each with a different transmission rate. How, then, can we determine the throughput of the whole path? To see the situation, assume that we have three links, each with a different transmission rate, as shown in Figure 7.2.

Figure 7.2 *Throughput in a path with three links in a series*

Figure 7.2 *Throughput in a path with three links in a series*



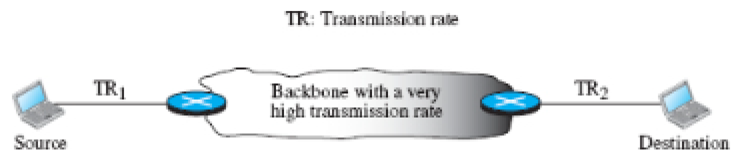
In Figure 7.2, the data can flow at the rate of 200 kbps in link 1. However, when the data arrives at router R1, it cannot pass at this rate. Data need to be queued at the router and sent at 100 kbps. When data arrive at router R2, they could be sent at the rate of 150 kbps, but there is not enough data to be sent. In other words, the average rate of the data flow in link 3 is also 100 kbps. We can conclude that the average data rate for this path is 100 kbps, the minimum of the three different data rates. Figure 7.2 also shows that we can simulate the behavior of each link with pipes of different sizes; the average throughput is determined by the bottleneck, the pipe with the smallest diameter. In general, in a path with *n* links in series, we have

Throughput = minimum {TR_1, TR_2, …, TR_n}

Although the situation in Figure 7.2 shows how to calculate the throughput when the data are passed through several links, the actual situation in the Internet is that the data normally pass through two access networks and the Internet backbone, as shown in Figure 7.3.

Figure 7.3 *A path through the Internet backbone*

Figure 7.3 *A path through the Internet backbone*



7.3.3 Packet Loss

Another issue that severely affects the performance of communication is **the number of packets lost during transmission.** When a router receives a packet while processing another packet, the received packet needs to be stored in the input buffer waiting for its turn. A router, however, has an input buffer with a limited size. A time may come when the buffer is full and the next packet needs to be dropped. The effect of packet loss on the Internet network layer is that the packet needs to be re-sent, which in turn may create overflow and cause more packet loss. A lot of theoretical studies have been done in queuing theory to prevent the overflow of queues and prevent packet loss.

7.4 INTERNET PROTOCOL VERSION 4

The network layer in the Internet has gone through several versions, but only **two versions** have survived: **IP Version 4 (IPv4)** and **IP Version 6 (IPv6).** Although IPv4 is almost depleted, we discuss it because there are still some areas that use this version and also because it is the foundation for IPv6.

7.4.1 IPv4 Addressing

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*.

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**

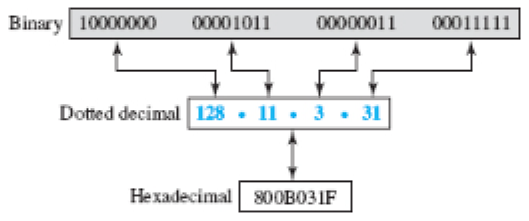
addresses, which means that the address space is 2^{32} or 4,294,967,296 (more than 4 billion). ***If there were no restrictions, more than 4 billion devices could be connected to the Internet.***

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 4 bits**. **This means that a 32-bit address has eight hexadecimal digits**. This notation is often used in network programming. Figure 7.4 shows an IP address in the three discussed notations.

Figure 7.4 *Three different notations in IPv4 addressing*

Figure 7.4 *Three different notations in IPv4 addressing*



Hierarchy in Addressing

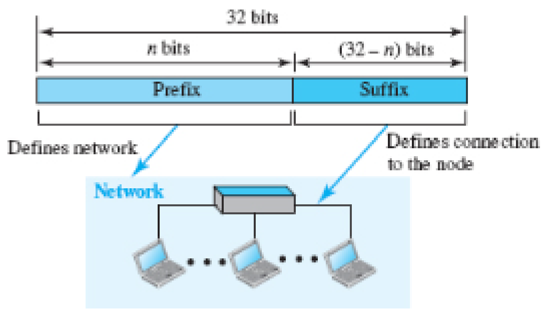
In any communication network that involves delivery, such as a telephone network or a postal network, the **addressing system is hierarchical**. In a postal network, the postal address (mailing address) includes the country, state, city, street, house number, and the name of the mail recipient. Similarly, 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 is 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 7.5 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.

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.

Figure 7.5 *Hierarchy in addressing*

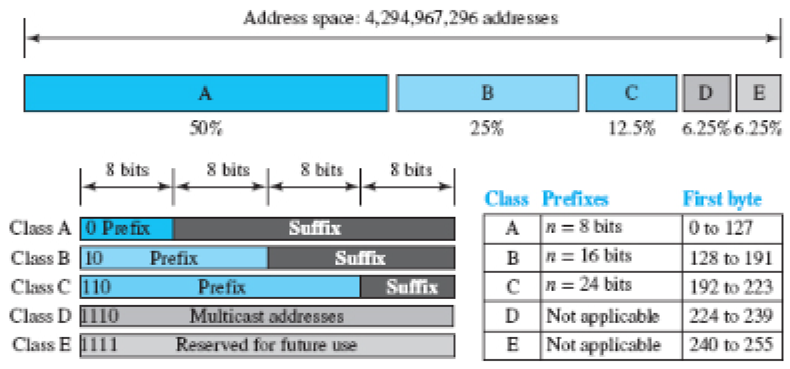
Figure 7.5 *Hierarchy in addressing*



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 (classes A, B, C, D, and E), as shown in Figure 7.6. This scheme is referred to as **classful addressing**. Although classful addressing belongs to the past, it helps us to understand classless addressing.

Figure 7.6 *Occupation of the address space in classful addressing*



Class	Prefixes	First byte
A	n = 8 bits	0 to 127
B	n = 16 bits	128 to 191
C	n = 24 bits	192 to 223
D	Not applicable	224 to 239
E	Not applicable	240 to 255

In class A, the network length is 8 bits, but because the first bit, which is 0, defines the class, we can have only 7 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 because the first 2 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 because 3 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.

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. Because 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 being available for organizations and individuals that needed to have an Internet connection. 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 would need to have one single network (seen by the rest of the world) with 16,777,216 nodes (computers in this single network). Because there were 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 design flaw. 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 E addresses were almost never used, wasting the whole class.

Classless Addressing

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 in Section 7.5), 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.

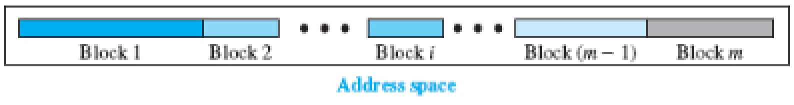
There was another motivation for classless addressing. During the 1990s, Internet Service Providers (ISPs) came into prominence. An ISP is an organization that provides Internet access and services for individuals, small businesses, and midsize organizations that do not want to create an Internet site and become involved in providing Internet services (such as electronic mail) for their employees. An ISP 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, …, 2^{32} addresses. One of the **restrictions** 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 7.7 shows the division of the whole address space into non-overlapping blocks.
Figure 7.7 *Variable-length blocks in classless addressing*

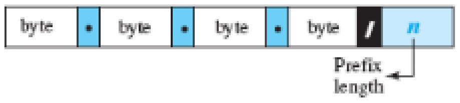
Figure 7.7 *Variable-length blocks in classless addressing*



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. Because 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.** An address in classless addressing can then be represented as shown in Figure 7.8.
Figure 7.8 *Slash notation (CIDR)*

Figure 7.8 *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. Because the value of prefix length, *n*, is given, we can easily find these three pieces of information.

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

Example 7.1

A classless address is given as 167.199.170.82/27. Find Number of addresses in the block, First address(Network address) and Last address

The desired three pieces of information as follows.

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

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 01010010
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* = NOT (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 7.2

We repeat Example 7.1 167.199.170.82/27. using the mask. The mask in dotted-decimal notation is 255.255.255.224. 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* = **NOT** (mask) + 1 = 0.0.0.31 + 1 =32addresses
First address:	First = (address) **AND** (mask) = 167.199.170. 64
Last address:	Last = (address) **OR** (**NOT** mask) =167.199.170. 95

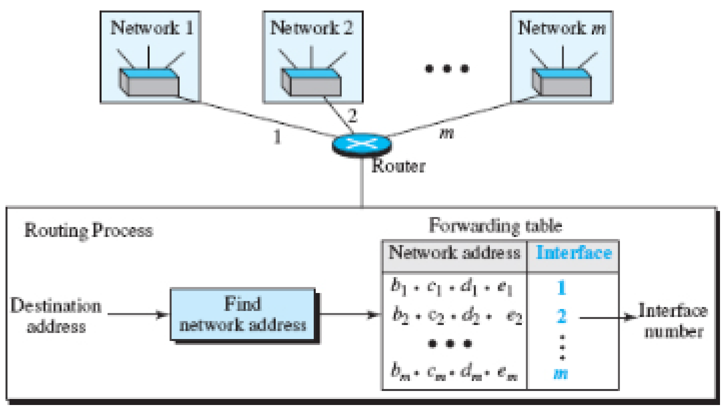
Example 7.3

A classless address is given as 200.10.20.40/28. Find Number of addresses in the block, First address(Network address) and Last address

Network Address

The preceding 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 up 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 and from which interface the packet should be sent out. When the packet arrives at the network, it reaches its destination host using link-layer addressing, which was discussed in Chapter 3 (Section 3.4). Figure 7.9 shows the idea.

Figure 7.9 *Network address*



After the network address has been 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, **three 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 n The requested block needs to be allocated where there are a contiguous number of available addresses in the address space. However, there is a restriction on choosing the first address in the block.**

3. **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**

First address = (prefix in decimal) × 2^{32-n} = (prefix in decimal) × N

Example 7.4

An ISP has requested a block of 1000 addresses. Because 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 $\boldsymbol{N}$, the prefix length is n, the *assigned number of addresses* to each subnetwork is $\boldsymbol{N_{sub}}$, 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:

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

- 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 7.5

An organization is granted a block of addresses with the beginning address 14.24.74.0/24. The organization needs to have three 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.**

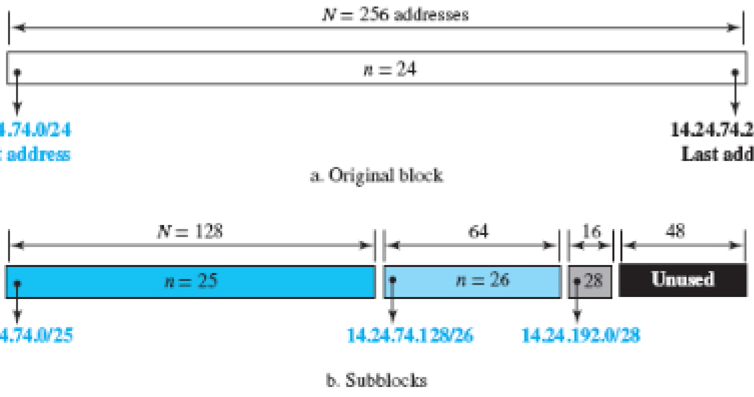
b. **The number of addresses in the second largest subblock, which requires 60 addresses, is not a power of 2 either. We allocate 64 addresses. The subnet mask for this subnet can be found as $n_2 = 32 - \log_2 64 = 26$. The first address in this block is 14.24.74.128/26; the last address is 14.24.74.191/26.**

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 7.10 shows the configuration of blocks. It shows the first address in each block.

Figure 7.10 *Solution to Example 7.5*

Figure 7.10 *Solution to Example 7.5*



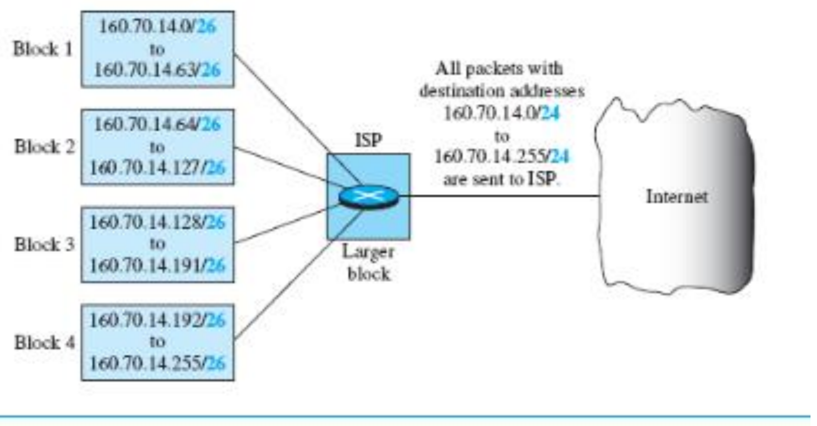
Address Aggregation

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 7.6

Figure 7.11 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 the routing we 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.

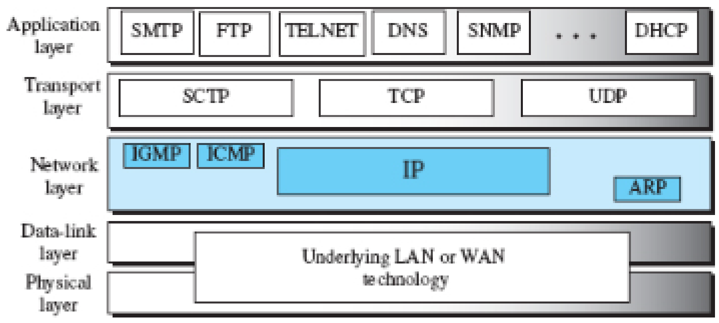
Figure 7.11 *Example of address aggregation*


Block 1
160.70.14.0/26 to 160.70.14.63/26
Block 2
160.70.14.64/26 to 160.70.14.127/26
Block 3
160.70.14.128/26 to 160.70.14.191/26
Block 4
160.70.14.192/26 to 160.70.14.255/26
ISP
Larger block
All packets with destination addresses 160.70.14.0/24 to 160.70.14.255/24 are sent to ISP.
Internet

7.4.2 Main and Auxiliary Protocols

The network layer in version 4 can be thought of as one main protocol and three auxiliary protocols. The main protocol, Internet Protocol version 4 (IPv4), is responsible for packetizing, forwarding, and delivery of a packet at the network layer. The Internet Control Message Protocol version 4 (ICMPv4) helps IPv4 to handle some errors that may occur in the network-layer delivery. The Internet Group Management Protocol (IGMP) is used to help IPv4 in multicasting. The Address Resolution Protocol (ARP) is used to glue the network and data-link layers in mapping network-layer addresses to link-layer addresses. Figure 7.12 shows the positions of these four protocols in the TCP/IP protocol suite.

Figure 7.12 *Position of IP and other network-layer protocols in TCP/IP protocol suite*



We discuss IPv4, ICMPv4, and ARP in this chapter. We will discuss IGMP when we talk about multicasting in Chapter 8.

.

IPv4 is an **unreliable datagram protocol**—a **best-effort delivery** service. The term *best effort* means that IPv4 packets can be corrupted, be lost, arrive out of order, be delayed or create congestion for the network. If reliability is important, IPv4 must be paired with a reliable transport-layer protocol such as TCP. An example of a common best-effort delivery service is the post office. The post office does its best to deliver the regular mail but does not always succeed. If an unregistered letter is lost or damaged in the mail, the would-be recipient will not receive the correspondence and the sender will need to re-create it.

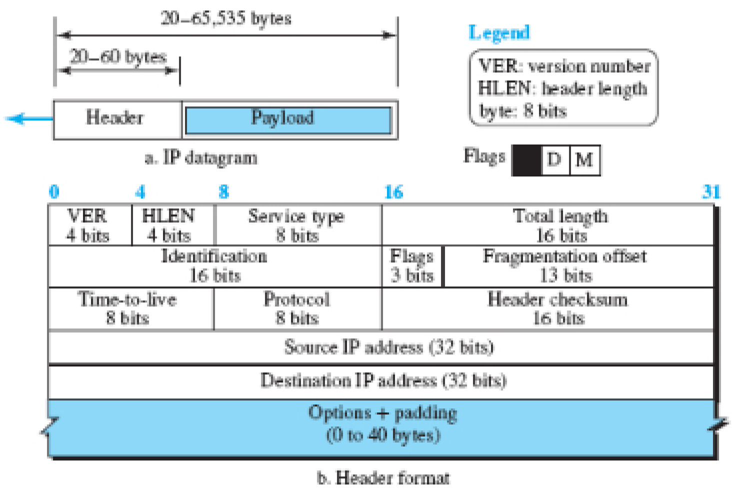
IPv4 is also a connectionless protocol that uses the datagram approach. This means that each datagram is handled independently, and *each datagram can follow a different route to the destination.* This implies that datagrams sent by the same source to the same destination could arrive out of order. Again, IPv4 relies on a higher-level protocol to take care of all these problems.

Datagram Format

In this section, we begin by discussing the first service provided by IPv4, packetizing. We show how IPv4 defines the format of a packet in which the data coming from the upper layer or other protocols are encapsulated. Packets used by the IP are called ***datagrams***. Figure 7.13 shows the IPv4 datagram format.

A datagram is a variable-length packet consisting of **two** parts: **the header** and **payload (data).** The header is 20 to 60 bytes in length and contains information essential to routing and delivery. The first 20 bytes are essential and together are called the main header. The next 40 bytes include options and padding that may or may not be present. It is customary in TCP/IP to show the header in 4-byte sections.

Figure 7.13 *IP datagram*



Discussing the meaning and rationale for the existence of each field is essential to understanding the operation of IPv4; a brief description of each field is in order.

- **Version number.** The 4-bit version number (VER) field defines the version of the IPv4 protocol, which, obviously, has the value of 4.
- **Header length.** The 4-bit header length (HLEN) field defines the total length of the datagram header in 4-byte words. The IPv4 datagram has a variable-length header. When a device receives a datagram, it needs to know when the header stops and the data, which are encapsulated in the packet, start. However, to make the value of the header length (number of bytes) fit in a 4-bit header length, the total length of the header is calculated as 4-byte words. The total length is divided by 4, and the value is inserted in the field. The receiver needs to multiply the value of this field by 4 to find the total length.
- **Service type.** In the original design of the IP header, this field was referred to as type of service (TOS), which defined how the datagram should be handled. In the late 1990s, the Internet Engineering Task Force (IETF) redefined the field to provide *differentiated services* (DiffServ), which divide applications into different classes according to their priority. The use of a 4-byte word for the header length is also logical because the IP header is always needed to be aligned in 4-byte boundaries.
- **Total length.** This 16-bit field defines the total length (header plus data) of the IP datagram in bytes. A 16-bit number can define a total length of up to 65,535 (when all bits are 1s). However, the size of the datagram is normally much less than this. This field helps the receiving device know when the packet has completely arrived. To find the length of the data coming from the upper layer, subtract the header length from the total length. The header length can be found by multiplying the value in the HLEN field by 4.

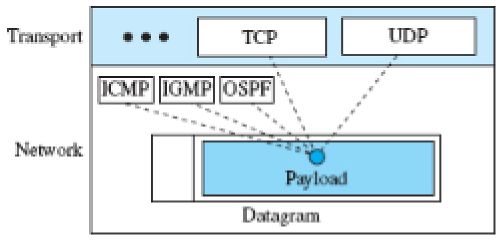
Length of data = total length − (HLEN) × 4

Though a size of 65,535 bytes might seem large, the size of the IPv4 datagram may increase in the near future as the underlying technologies allow even more throughput (greater bandwidth).

One may ask why we need this field anyway. When a machine (router or host) receives a frame, it drops the header and the trailer, leaving the datagram. Why include an extra field that is not needed? The answer is that in many cases we really do not need the value in this field. However, there are occasions in which the datagram is not the only thing encapsulated in a frame; it may be that padding has been added. For example, the Ethernet protocol has a minimum and maximum restriction on the size of data that can be encapsulated in a frame (46 to 1500 bytes). If the size of an IPv4 datagram is less than 46 bytes, some padding will be added to meet this requirement. In this case, when a machine decapsulates the datagram, it needs to check the total length field to determine how much is really data and how much is padding.

- **Identification, flags, and fragmentation offset.** These three fields are related to the fragmentation of the IP datagram when the size of the datagram is larger than the underlying network can carry. We discuss the contents and importance of these fields when we talk about fragmentation later in this section.
- **Time-to-live.** Because of some malfunctioning of routing protocols (discussed in the next bullet point) a datagram may be circulating in the Internet, visiting some networks over and over without reaching the destination. This may create extra traffic in the Internet. The time-to-live (TTL) field is used to control the maximum number of hops (routers) visited by the datagram. When a source host sends the datagram, it stores a number in this field. This value is approximately 2 times the maximum number of routes between any two hosts. Each router that processes the datagram decrements this number by one. If this value, after being decremented, is zero, the router discards the datagram.
- **Protocol.** In TCP/IP, the data section of a packet, called the ***payload***, carries the whole packet from another protocol. A datagram, for example, can carry a packet belonging to any transport-layer protocol such as UDP or TCP. A datagram can also carry a packet from other protocols that directly use the service of the IP, such as some routing protocols or some auxiliary protocols. The Internet authority has given any protocol that uses the service of the IP a unique 8-bit number that is inserted in the protocol field. When the payload is encapsulated in a datagram at the source IP, the corresponding protocol number is inserted in this field; when the datagram arrives at the destination, the value of this field helps to define to which protocol the payload should be delivered. In other words, this field provides multiplexing at the source and demultiplexing at the destination, as shown in Figure 7.14. Note that the protocol fields at the network layer play the same role as the port numbers at the transport layer (Chapter 9). However, we need two port numbers in a transport-layer packet because the port numbers at the source and destination are different, but we need only one protocol field because this value is the same for each protocol no matter whether it is located at the source or the destination.

Figure 7.14 *Multiplexing and demultiplexing using the value of the protocol field*



Some protocol values

ICMP	01
IGMP	02
TCP	06
UDP	17
OSPF	89

- **Header checksum.** IP is not a reliable protocol; it does not check whether the payload carried by a datagram is corrupted during the transmission. IP puts the burden of error checking of the payload on the protocol that owns the payload, such as UDP or TCP. The datagram header, however, is added by IP, and its error checking is the responsibility of IP. Errors in the IP header can be a disaster. For example, if the destination IP address is corrupted, the packet can be delivered to the wrong host. If the protocol field is corrupted, the payload may be delivered to the wrong protocol. If the fields related to the fragmentation are corrupted, the datagram cannot be reassembled correctly at the destination, and so on. For these reasons, IP a **header checksum field to check the header**, but not the payload. We need to remember that, because the value of some fields, such as TTL, which are related to fragmentation and options, may change from router to router, the checksum needs to be recalculated at each router, as we show in Example 7.11.
- **Source and destination addresses.** These 32-bit source and destination address fields define the IP address of the source and destination, respectively. The source host should know its IP address. The destination IP address is either known by the protocol that uses the service of IP or is provided by the DNS as described in Chapter 10. Note that the value of these fields must remain unchanged during the time the IP datagram travels from the source host to the destination host. IP addresses were discussed earlier in this chapter.
- **Options.** A datagram header can have up to 40 bytes of options. Options can be used for network testing and debugging. Although options are not a required part of the IP header, option processing is required of the IP software. This means that all implementations must be able to handle options if they are present in the header. The existence of options in a header creates some burden on the datagram handling; some options can be changed by routers, which forces each router to recalculate the header checksum. There are 1-byte and multibyte options that we will briefly discuss later in the chapter.
- **Payload.** Payload, or data, is the main reason for creating a datagram. Payload is the packet coming from other protocols that use the service of IP. Comparing a datagram to a postal package, payload is the content of the package; the header is only the information written on the package.

Example 7.7

An IPv4 packet has arrived with the first 8 bits as $(01000010)_2$. The receiver discards the packet. Why?

Solution

There is an error in this packet. The 4 leftmost bits $(0100)_2$ show the version, which is correct. The next 4 bits $(0010)_2$ show an invalid header length ($2 \times 4 = 8$). The minimum number of bytes in the header must be 20. The packet has been corrupted in transmission.

Example 7.8

In an IPv4 packet, the value of HLEN is $(1000)_2$. How many bytes of options are being carried by this packet?

Solution

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

Example 7.9

In an IPv4 packet, the value of HLEN is 5, and the value of the total length field is $(0028)_{16}$. How many bytes of data are being carried by this packet?

Solution

The HLEN value is 5, which means the total number of bytes in the header is 5×4, or 20 bytes (no options). The total length is $(0028)_{16}$ or 40 bytes, which means the packet is carrying 20 bytes of data ($40 - 20$).

Example 7.10

An IPv4 packet has arrived with the first few hexadecimal digits as shown.

$(45000028000100000102 \ldots)_{16}$

How many hops can this packet travel before being dropped? To which upper-layer protocol do the data belong?

Solution

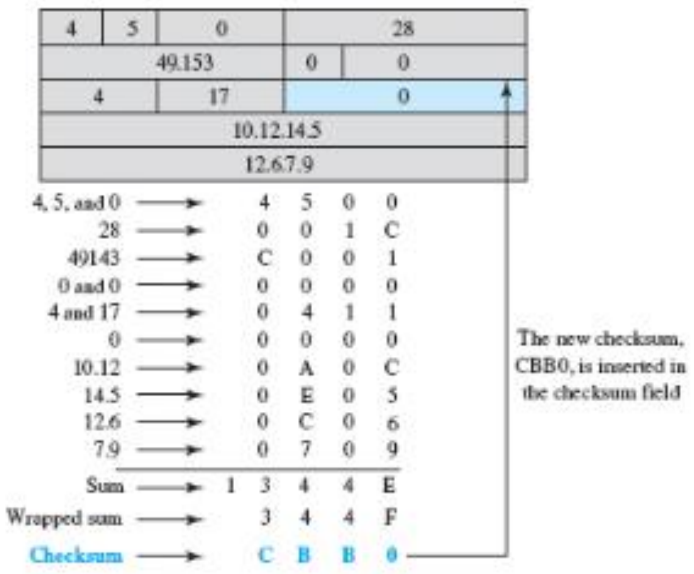
First 4 bytes (8 bits)-Version, HLEN, Service type, total length
Second 4 bytes (8 bits)-Identification bits, flags, fragment offset
Third bytes is about TTL, protocol

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

Example 7.11

Figure 7.15 shows an example of a checksum calculation for an IPv4 header without options. The header is divided into 16-bit sections. All the sections are added, and the sum is complemented after wrapping the leftmost digit. The result is inserted in the checksum field.

Figure 7.15 *Example of checksum calculation in IPv4*



Note that the calculation of wrapped sum and checksum can also be done as follows in hexadecimal:

Wrapped Sum = Sum mod FFFF

Checksum = FFFF − Wrapped Sum

Fragmentation

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

Maximum Transfer Unit (MTU)

Each link-layer protocol has its own frame format. One of the features of each format is the maximum size of the payload that can be encapsulated. In other words, when a datagram is encapsulated in a frame, the total size of the datagram must be less than this maximum size, which is defined by the restrictions imposed by the hardware and software used in the network (see Figure 7.16).

Figure 7.16 *Maximum transfer unit (MTU)*

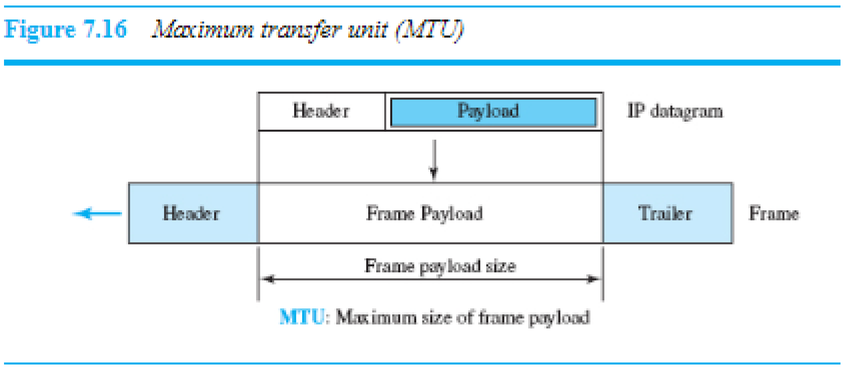


Figure 7.16 Maximum transfer unit (MTU)
Header
Payload
IP datagram
Header
Frame Payload
Trailer
Frame
Frame payload size
MTU: Maximum size of frame payload

The value of the maximum transfer unit (MTU) differs from one physical network protocol to another. For example, the value for a LAN is normally 1500 bytes, but for a WAN it can be larger or smaller.

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

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

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

When we talk about fragmentation, we mean that the payload of the IP datagram is fragmented. However, most parts of the header, with the exception of some options, must be copied by all fragments. The host or router that fragments a datagram must change the values of three fields: flags, fragmentation offset, and total length. The rest of the fields must be copied. Of course, the value of the checksum must be recalculated regardless of fragmentation.

Fields Related to Fragmentation

We mentioned earlier that three fields in an IP datagram are related to fragmentation: *identification, flags,* and *fragmentation offset.* Let us explain these fields now.

The **16-bit *identification field*** identifies a datagram originating from the source host. The combination of the identification and source IP address must uniquely define a datagram as it leaves the source host. To guarantee uniqueness, the IP protocol uses a counter to label the datagrams. The counter is initialized to a positive number. When the IP protocol sends a datagram, it copies the current value of the counter to the identification field and increments the counter by one. As long as the counter is kept in the main memory, uniqueness is guaranteed. When a datagram is fragmented, the value in the identification field is copied into all fragments. In other words, all fragments have the same identification number, which is also the same as the original datagram. The identification number helps the destination in reassembling the datagram. It knows that all fragments having the same identification value should be assembled into one datagram.

The **3-bit *flags field*** defines three flags. The leftmost bit is reserved (not used). The second bit **(D bit) is called the *do not fragment* bit**. If its value is 1, the machine must not fragment the datagram. If it cannot pass the datagram through any available physical network, it discards the datagram and sends an ICMP error message to the source host (discussed in Section 7.4.4). If its value is 0, the datagram can be fragmented if necessary. The third bit (**M bit) is called the *more fragment bit*.** If its value is 1, it means the datagram is not the last fragment; there are more fragments after this one. If its value is 0, it means this is the last or only fragment.

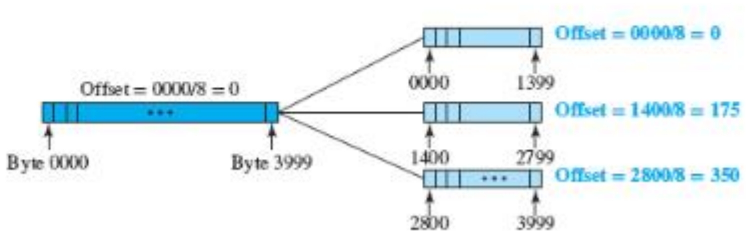
The **13-bit *fragmentation offset field*** shows the relative position of this fragment with respect to the whole datagram. It is the offset of the data in the original datagram measured in units of 8 bytes.

Example

An IP datagram of 4020Bytes (Header 20 bytes and payload 4000 bytes) reached at the router and must be forward to link with MTU of 1420 bytes. How many fragments will be generated and also write MF, offset field and total length value for all the fragments.

Figure 7.17 shows a datagram with a data size of 4000 bytes fragmented into three fragments. The bytes in the original datagram are numbered 0 to 3999. The first fragment carries bytes 0 to 1399. The offset for this datagram is 0/8 = 0. The second fragment carries bytes 1400 to 2799; the offset value for this fragment is 1400/8 = 175. Finally, the third fragment carries bytes 2800 to 3999. The offset value for this fragment is 2800/8 = 350.

Figure 7.17 *Fragmentation example*



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

Figure 7.18 shows an expanded view of the fragments in Figure 7.17. The original packet starts at the client; the fragments are reassembled at the server. The value of the identification field is the same in all fragments, as is the value of the flags field with the more fragment bit (M bit) set for all fragments except the last. Also, the value of the offset field for each fragment is shown. Note that although the fragments arrived out of order at the destination, they can be correctly reassembled.

Figure 7.18 also shows what happens if a fragment itself is fragmented. In this case the value of the offset field is always relative to the original datagram. For example, in the figure, the second fragment is itself fragmented later into two fragments of 800 bytes and 600 bytes, but the offset shows the relative position of the fragments to the original data.

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

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

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

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

d. **Continue the process. The last fragment has its M bit set to 0.**

e. **Continue the process. The last fragment has an M bit value of 0.**

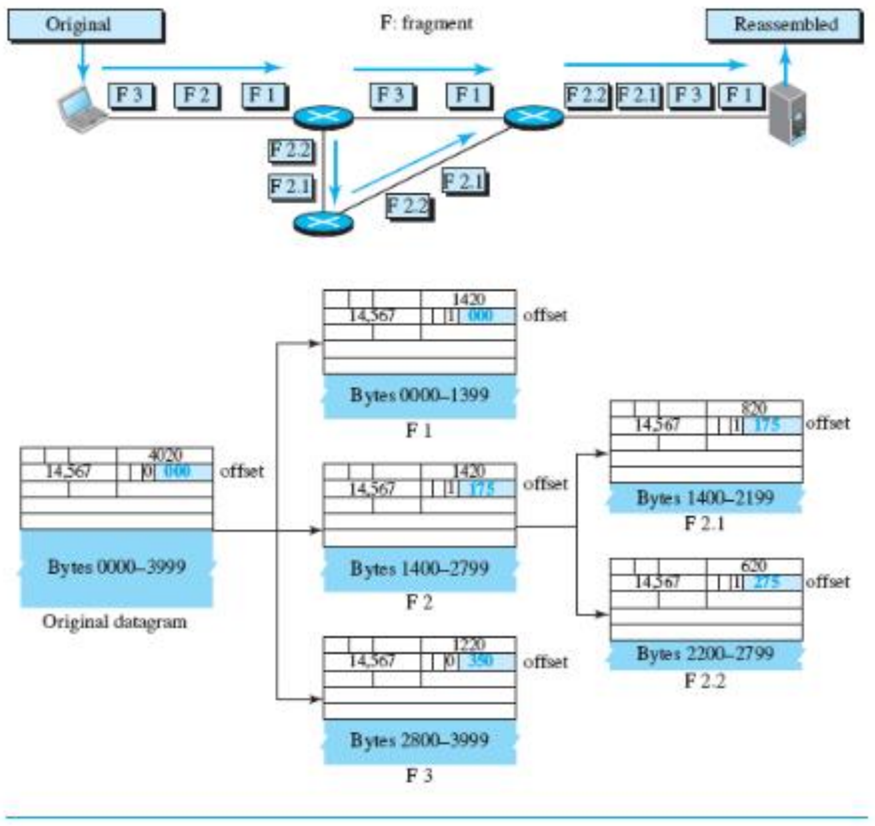
Example 7.12

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

Solution

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

Figure 7.18 *Detailed fragmentation example*



Example 7.13

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

Solution

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

Example 7.14

A packet has arrived with an M bit value of 1 and a fragmentation offset value of 0. Is this the first fragment, the last fragment, or a middle fragment?

Solution

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

Example 7.15

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

Solution

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

Example 7.16

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

Solution

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

7.5 NEXT GENERATION IP (IPV6)

The address depletion of IPv4 and other shortcomings of this protocol prompted a new version of IP protocol in the early 1990s. The new version, which is called **Internet Protocol version 6 (IPv6)** or **IP new generation (IPng)**, was a proposal to augment the address space of IPv4 and at the same time redesign the format of the IP packet and revise some auxiliary protocols such as ICMP. It is interesting to know that IPv5 was a proposal, based on the OSI model, that never materialized.

The main changes needed in the new protocol were as follows: **larger address space, better header format, new options, allowance for extension, support for resource allocation, and support for more security.** The implementation of theses changes made it necessary to create a new version of the ICMP protocol, ICMPv6.

This section has four subsections:

- The first subsection discusses the addressing mechanism in the new generation of the Internet. It first describes the representation and address space. It then shows the allocation in the address space. Finally, it explains autoconfiguration and renumbering, which makes it easy for a host to move from one network to another.
- The second subsection discusses IPv6 protocol. First the new packet format is described. Then it shows how the idea of an extension header can replace the options in version 4.
- The third subsection discusses ICMPv6. It describes how the new protocol replaces several auxiliary protocols in version 4. The subsection also divides the messages in this protocol into four categories and describes them.
- The fourth subsection briefly shows how a smooth transition can be made from the current version to the new one. It explains three strategies that need to be followed for this smooth transition.

7.5.1 IPv6 Addressing

The main reason for migration from IPv4 to IPv6 is the small size of the address space in IPv4. In this section, we show how the huge address space of IPv6 prevents address depletion in the future. We also discuss how the new addressing responds to some problems in the IPv4 addressing mechanism. An **IPv6 address is 128 bits or 16 bytes (octets) long**, 4 times the address length in IPv4.

Representation

A computer normally stores the address in binary, but it is clear that 128 bits cannot easily be handled by humans. Several notations have been proposed to represent IPv6 addresses when they are handled by humans. The following shows two of these notations: binary and colon hexadecimal.

Binary (128 bits)	**1111111011110110 1011 … 1111111100000000**
Colon hexadecimal	**FEF6:BA98:7654:3210:ADEF:BBFF:2922:FF00**

Binary notation is used when the addresses are stored in a computer. The colon **hexadecimal notation** (or colon hex for short) divides the address into eight sections, each made of four hexadecimal digits separated by colons.

Abbreviation

Although an IPv6 address, even in hexadecimal format, is very long, many of the digits are zeros. In this case, we can abbreviate the address. The leading zeros of a section can be omitted. Using this form of abbreviation, 0074 can be written as 74, 000F as F, and 0000 as 0. Note that 3210 cannot be abbreviated. Further abbreviation, often called **zero compression**, can be applied to colon hex notation if there are consecutive sections consisting of zeros only. We can remove all the zeros altogether and replace them with a double semicolon.

FDEC:0:0:0:0:BBFF:0:FFFF	→	**FDEC::BBFF:0:FFFF**

Note that this type of abbreviation is allowed only once per address. If there is more than one run of zero sections, only one of them can be compressed.

Mixed Notation

Sometimes we see a mixed representation of an IPv6 address: colon hex and dotted-decimal notation. This is appropriate during the transition period in which an IPv4 address is embedded in an IPv6 address (as the rightmost 32 bits). We can use the colon hex notation for the leftmost six sections and 4-byte dotted-decimal notation instead of the rightmost two sections. However, this happens when all or most of the leftmost sections of the IPv6 address are zeros. For example, the address (::130.24.24.18) is a legitimate address in IPv6, in which the zero compression shows that all 96 leftmost bits of the address are zeros.

CIDR Notation

As we will see shortly, IPv6 uses hierarchical addressing. For this reason, IPv6 allows slash or CIDR notation. For example, the following shows how we can define a prefix of 60 bits using CIDR. We will later show how an IPv6 address is divided into a prefix and a suffix.

FDEC::BBFF:0:FFFF/60

Three Address Types

In IPv6, a destination address can belong to one of three categories: unicast, anycast, and multicast.

Unicast Address A unicast address defines a single interface (computer or router). The packet sent to a unicast address will be routed to the intended recipient.

Anycast Address An **anycast address** defines a group of computers that all share a single address. A packet with an anycast address is delivered to only one member of the group, the most reachable one. An anycast communication is used, for example, when there are several servers that can respond to an inquiry. The request is sent to the one that is most reachable. The hardware and software generate only one copy of the request; the copy reaches only one of the servers. IPv6 does not designate a block for anycasting; the addresses are assigned from the unicast block.

Multicast Address A multicast address also defines a group of computers. However, there is a difference between anycasting and multicasting. In anycasting, only one copy of the packet is sent to one of the members of the group; in multicasting each member of the group receives a copy. As we will see shortly, IPv6 has designated a block for multicasting from which the same address is assigned to the members of the group. It is interesting that IPv6 does not define broadcasting, even in a limited version. IPv6 considers broadcasting as a special case of multicasting.

7.5.2 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 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.**

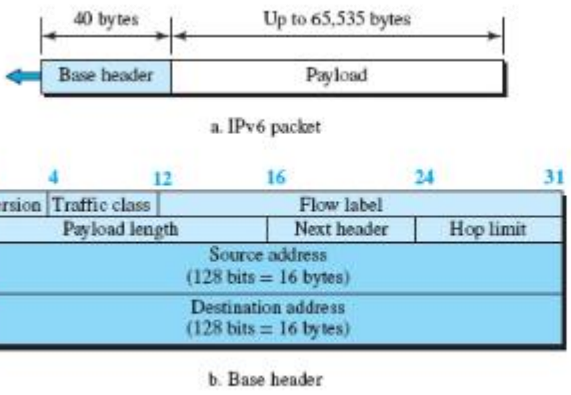
- **Better header format.** IPv6 uses a new header format in which options are separated from the base header and inserted, when needed, between the base header and the data. This simplifies and speeds up the routing process because most of the options do not need to be checked by routers.
- **New options.** 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.

Packet Format

The IPv6 packet is shown in Figure 7.46. Each packet is composed of a base header followed by the payload. The base header occupies 40 bytes, whereas the payload can be up to 65,535 bytes of information. The description of fields follows.

- **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.

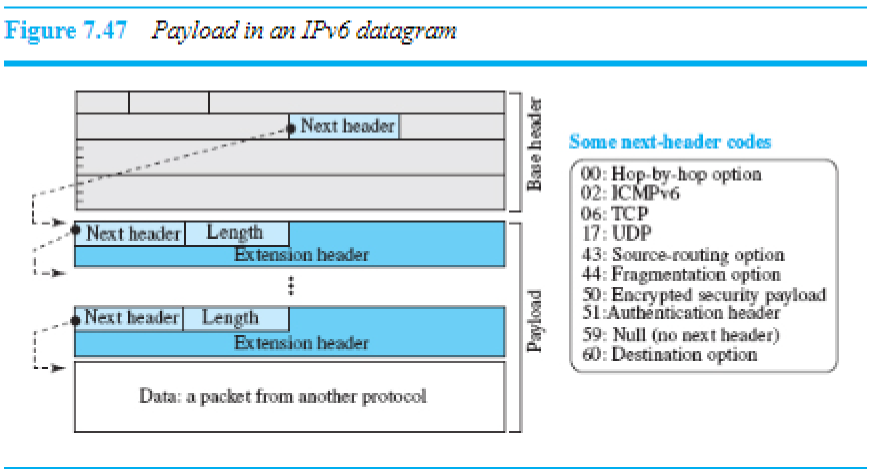
Figure 7.46 *IPv6 datagram*



- **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 in the section.
- **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 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 address.** 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 7.47. 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.

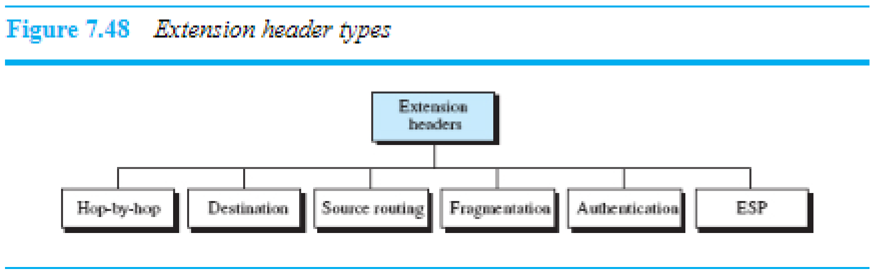
Figure 7.47 *Payload in an IPv6 datagram*



Extension Header

An IPv6 packet is made up 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 7.48).

Figure 7.48 *Extension header types*



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 1 byte, the rest will be filled by 0s.
- **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 are not altered in transition by some hacker. We discuss more about authentication in Chapter 13.

Encrypted Security Payload

The **encrypted security payload (ESP)** is an extension that provides confidentiality and guards against eavesdropping. Again we discuss more about providing **confidentiality for IP packets** in Chapter 13.

Concept of Flow and Priority in IPv6

The IP was originally designed as a connectionless protocol. However, the tendency is to use the IP as a connection-oriented protocol. 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, and having the same kind of security. 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, and long processing time. 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.

Fragmentation and Reassembly

There is 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 in Section 7.5.3) to inform the source.

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.

CHAPTER 8

Network Layer: Routing of Packets

8.1 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 up 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.

8.1.1 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 does not need a forwarding table because it delivers its packet to the default router in its local network. The destination host does not need a 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 preceding 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.

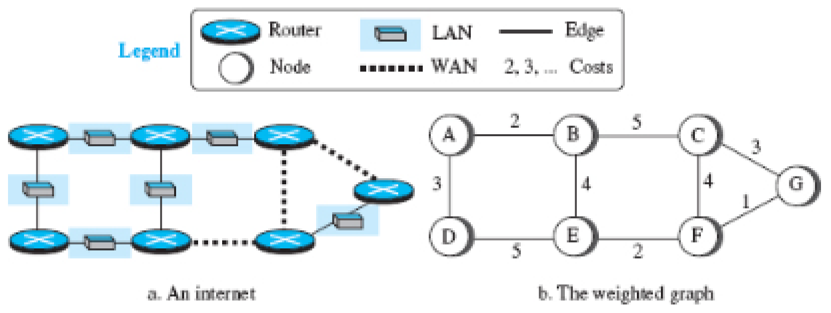
An Internet as a Graph

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 when we discuss that routing protocol. 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 8.1 shows how an internet can be modeled as a graph.

8.1.2 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 8.1, 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 8.1 *An internet and its graphical representation*

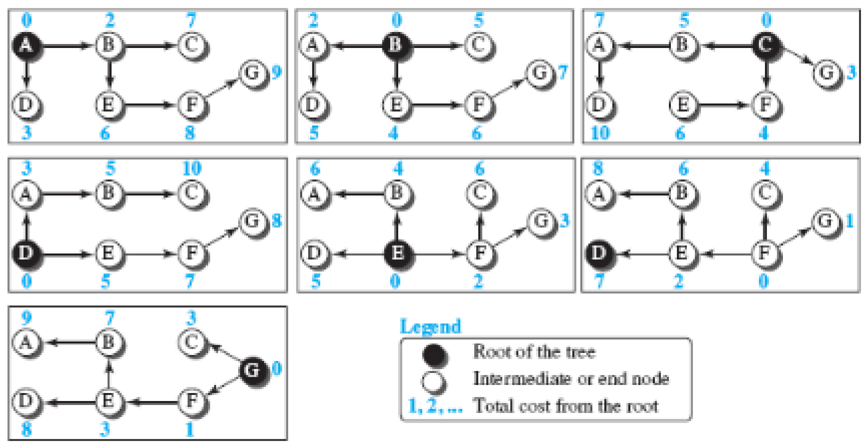


Least-Cost Trees

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 these paths is to combine them into 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 path trees for the whole internet. Figure 8.2 shows the seven least-cost trees for the internet in Figure 8.1. 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 8.2 *Least-cost trees for nodes in the internet of Figure 8.1*



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 8.2, 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 traveling 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 8.2, 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).**

8.2 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 algorithm; later we show how a routing protocol in the Internet implements one of these algorithms.

8.2.1 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 its neighbors about 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,** …) 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***.

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

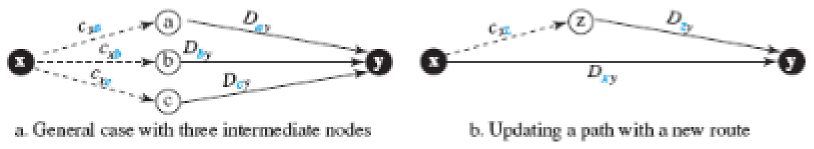
In distance-vector routing, normally we want to update an existing least cost with a least cost through an intermediary node, such as ***z***, if the latter is shorter. In this case, the equation becomes simpler:

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

Figure 8.3 shows the idea graphically for both cases.

Figure 8.3 *Graphical idea behind Bellman-Ford equation*

Figure 8.3 *Graphical idea behind Bellman-Ford equation*


c_{xa}
c_{xb}
c_{xc}
D_{ay}
D_{by}
D_{cy}
x
a
b
c
y
a. General case with three intermediate nodes
c_{xz}
D_{zy}
D_{xy}
z
b. Updating a path with a new route

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 8.3, 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.

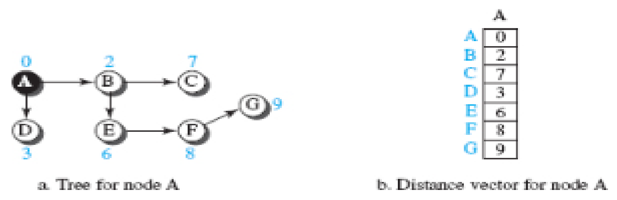
We will shortly show how we use the Bellman-Ford equation and the concept of distance vectors to build least-cost paths for each node in distance-vector routing, but first we need to discuss the concept of a distance vector.

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 8.4 shows the tree for node A in the internet in Figure 8.1 and the corresponding distance vector.

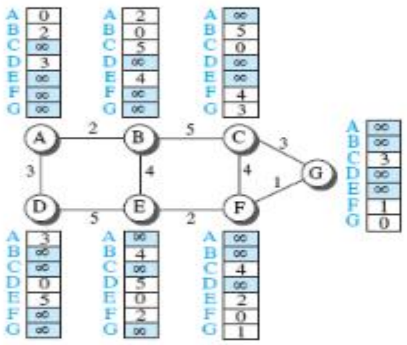
Figure 8.4 *The distance vector corresponding to a tree*

Figure 8.4 *The distance vector corresponding to a tree*



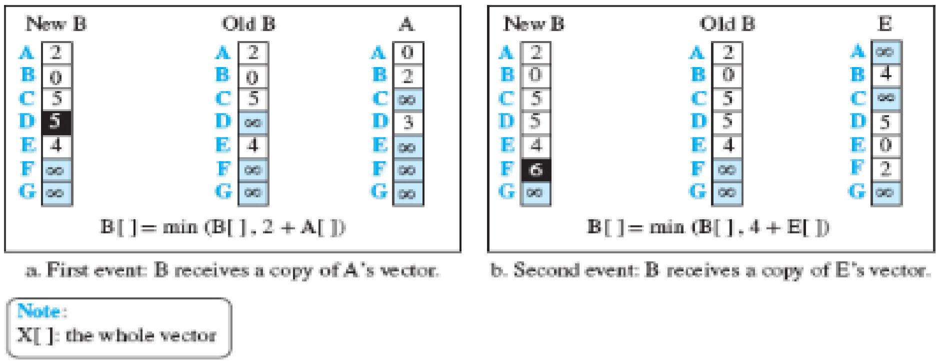
Note that the *name* of the distance vector defines the root, the *indexes* define the destinations, and the *value* of each cell defines the least cost from the root to the destination. A distance vector does not give the path to the destinations as the least-cost tree does; it gives only the least 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 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 8.5 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.



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 must 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. In Figure 8.6, we show the whole vector instead of seven equations for each update. The figure shows two asynchronous events, happening one after another with some time in 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 $c_{EA} = 4$.

Figure 8.6 *Updating distance vectors*

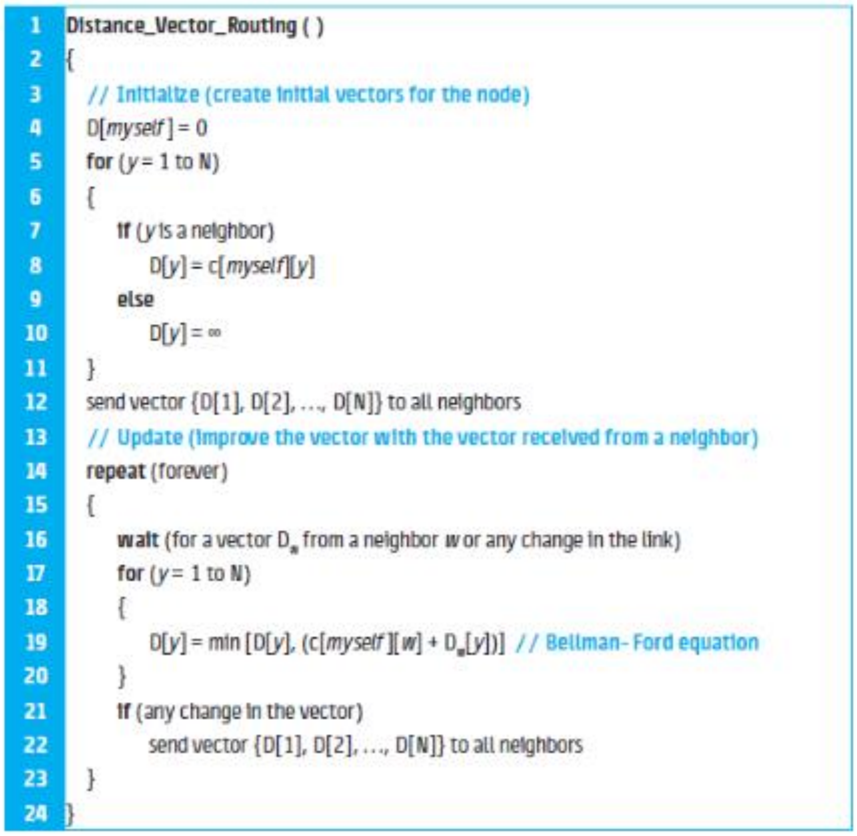

New B
Old B
A
B[ ] = min (B[ ], 2 + A[ ])
a. First event: B receives a copy of A's vector.
New B
Old B
E
B[ ] = min (B[ ], 4 + E[ ])
b. Second event: B receives a copy of E's vector.
Note:
X[ ]: the whole vector

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 8.1. The algorithm is run by its node independently and asynchronously.

Table 8.1 *Distance-vector routing algorithm for a node*



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

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.

8.2.2 Link-State Routing

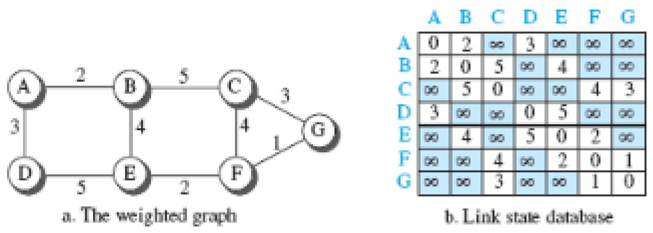
A routing algorithm that directly follows our discussion for creating least-cost trees and forwarding tables is **link-state (LS) routing**. This method uses the term *link state* to define the characteristic of a link (an edge) that represents a network in the internet. In this algorithm the cost associated 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 8.8 shows an example of an LSDB for the graph in Figure 8.1. 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 8.8 *Example of a link-state database*

Figure 8.8 *Example of a link-state database*

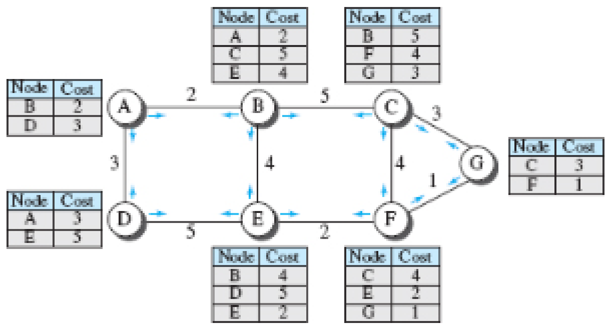


	A	B	C	D	E	F	G
A	0	2	∞	3	∞	∞	∞
B	2	0	5	∞	4	∞	∞
C	∞	5	0	∞	∞	4	3
D	3	∞	∞	0	5	∞	∞
E	∞	4	∞	5	0	2	∞
F	∞	∞	4	∞	2	0	1
G	∞	∞	3	∞	∞	1	0

Now the question is, "How can each node 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 8.9 for our internet in Figure 8.1. 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 8.9. 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 8.9 *LSPs created and sent out by each node to build the LSDB*

Figure 8.9 *LSPs created and sent out by each node to build the 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.

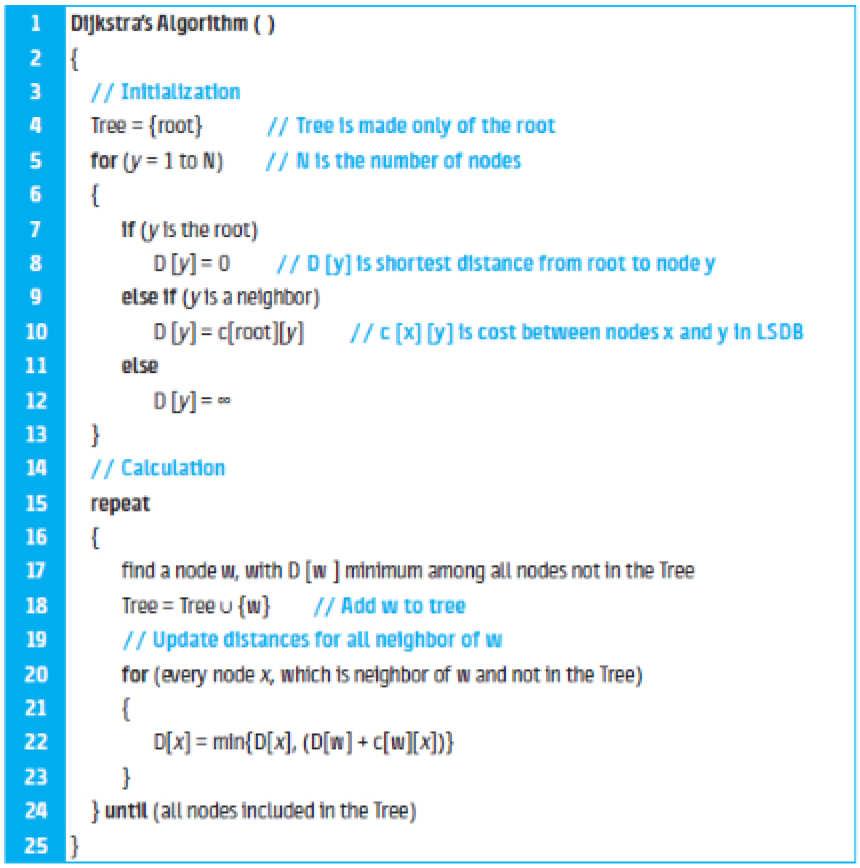
Formation of Least-Cost Trees

To create a least-cost tree for itself, using the shared LSDB, each node needs to run the famous **Dijkstra's 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 these three steps finally create the least-cost tree. Table 8.2 shows a simplified version of Dijkstra's algorithm.

Table 8.2 *Dijkstra's algorithm*

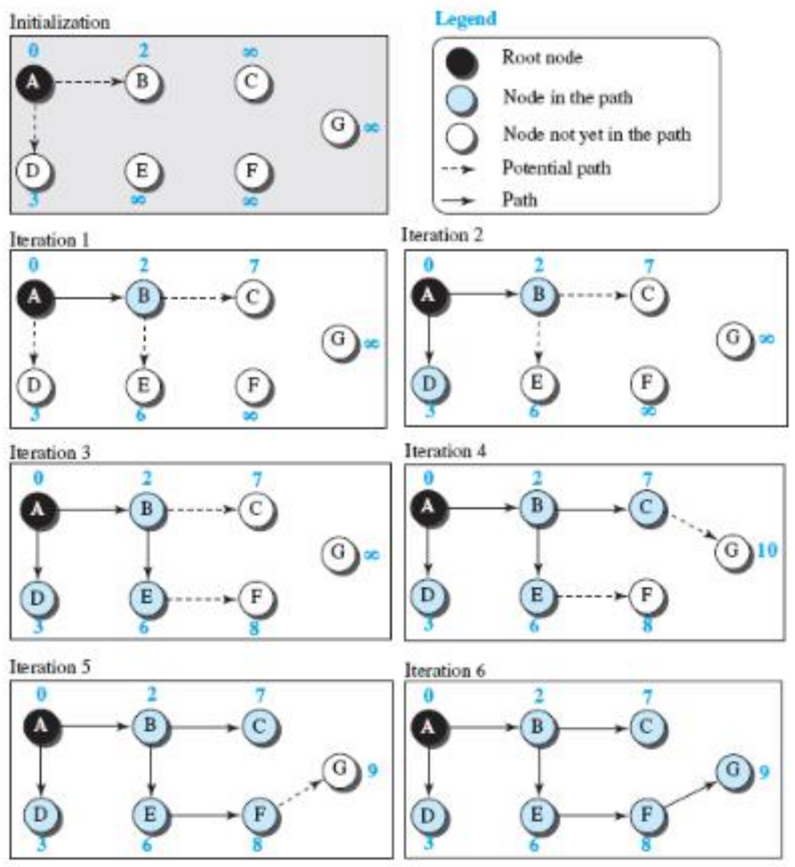


```
1   Dijkstra's Algorithm ( )
2   {
3     // Initialization
4     Tree = {root}            // Tree is made only of the root
5     for (y = 1 to N)         // N is the number of nodes
6     {
7         if (y is the root)
8             D [y] = 0        // D [y] is shortest distance from root to node 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 neighbor of w
20        for (every node x, which is 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  }
```

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 8.10 shows the formation of the least-cost tree for the graph in Figure 8.8 using Dijkstra's algorithm. We need to go through an initialization step and six iterations to find the least-cost tree.

Figure 8.10 *Least-cost tree*



8.2.3 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 that belongs to a commercial rival of the sender who might inspect the packets to obtain 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 in which the mere goal of routing is reachability: to allow the packet to reach its destination more efficiently without assigning costs to the route.

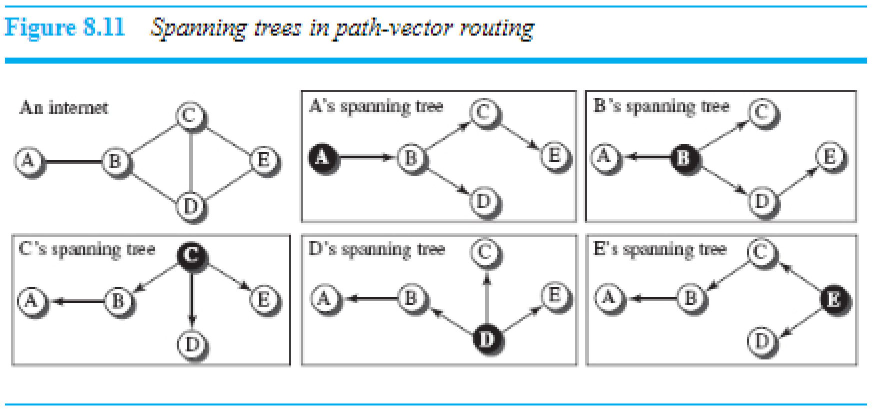
To respond to these demands, a third routing algorithm, called **path-vector (PV) routing**, has been devised. Path-vector routing does not have the drawbacks of LS or DV routing as described previously because it is not based on least-cost routing. The best route is determined by the **source** using 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.

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 8.11 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 8.11 *Spanning trees in path-vector routing*



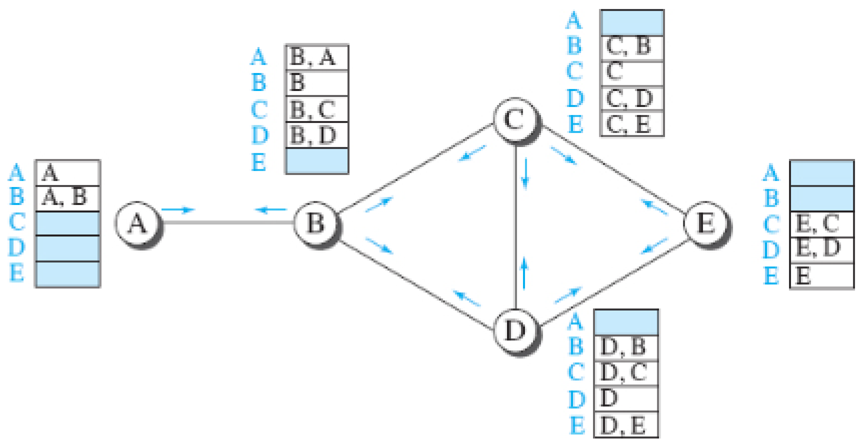
Creation of Spanning Trees

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 8.12 shows all these path vectors for our internet in Figure 8.11. Note, however, that we do not mean that all these tables are created simultaneously; they are created when each node is booted. Figure 8.12 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 equation, but applies its own policy instead of looking for the least cost. We can define this equation as

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

Figure 8.12 *Path vectors made at booting time*


A: A A; B A, B; C; D; E
B: A B, A; B B; C B, C; D B, D; E
C: A; B C, B; C C; D C, D; E C, E
D: A; B D, B; C D, C; D D; E D, E
E: A; B; C E, C; D E, D; E E

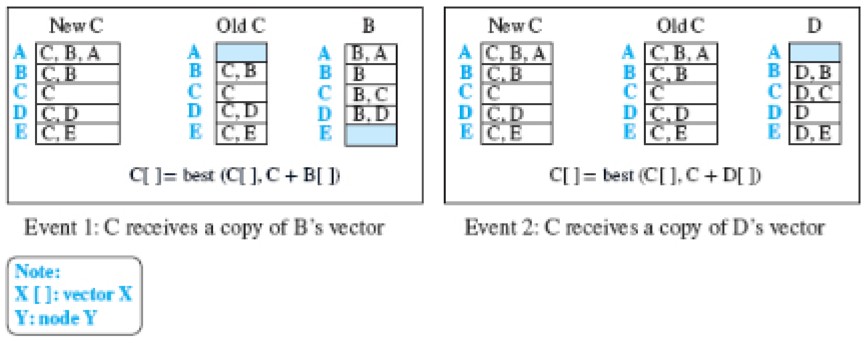
In this equation, the operator (+) means to add *x* to the beginning of the path. We also need to be cautious to avoid adding a node to an empty path because an empty path means one that does not exist.

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 8.13 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 8.13 *Updating path vectors*

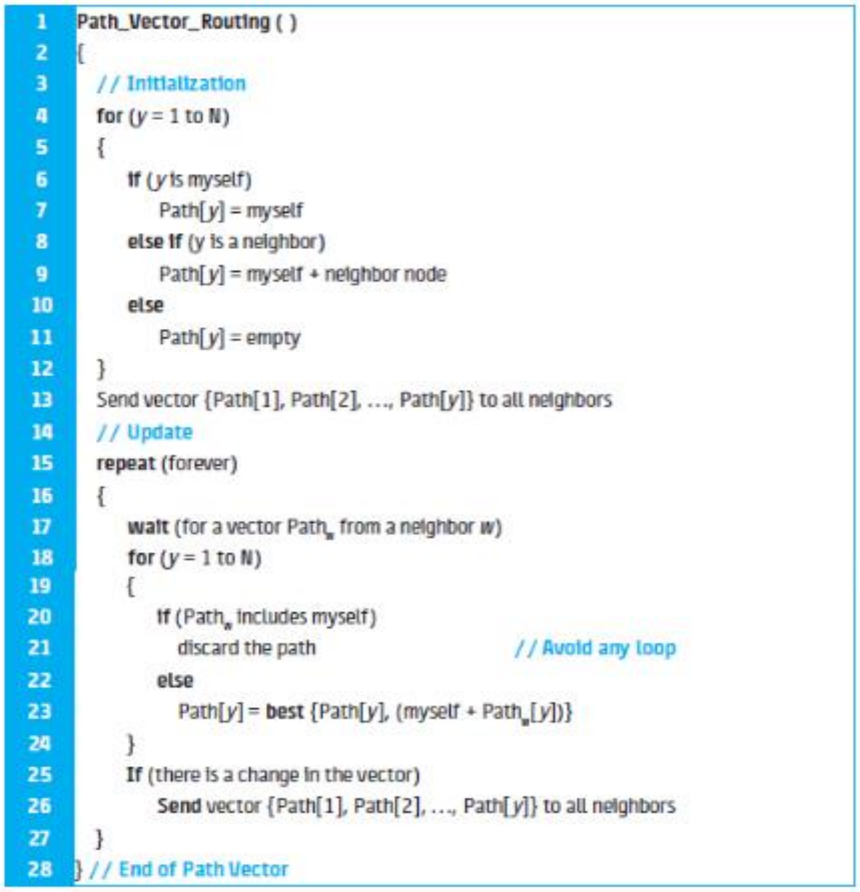
Figure 8.13 *Updating path vectors*


New C
A C, B, A; B C, B; C C; D C, D; E C, E
Old C
A; B C, B; C C; D C, D; E C, E
B
A B, A; B B; C B, C; D B, D; E
C[ ] = best (C[ ], C + B[ ])
Event 1: C receives a copy of B’s vector
New C
A C, B, A; B C, B; C C; D C, D; E C, E
Old C
A C, B, A; B C, B; C C; D C, D; E C, E
D
A; B D, B; C D, C; D D; E D, E
C[ ] = best (C[ ], C + D[ ])
Event 2: C receives a copy of D’s vector
Note:
X [ ]: vector X
Y: node Y

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 8.3.

Table 8.3 *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 Path_w from a neighbor w)
18         for (y = 1 to N)
19         {
20             if (Path_w includes myself)
21                discard the path                    // Avoid any loop
22             else
23                Path[y] = best {Path[y], (myself + Path_w[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.

8.3 UNICAST ROUTING PROTOCOLS

In Section 8.2, 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.***

8.3.3 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 Section 8.2.2. 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 8.19 shows the idea of the cost from a router to the destination host network. We can compare Figure 8.19 with Figure 8.15 for the RIP.

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 Section 8.2.2. Figure 8.20 shows the forwarding tables for the simple AS in Figure 8.19. 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.

Figure 8.19 *Metric in OSPF*

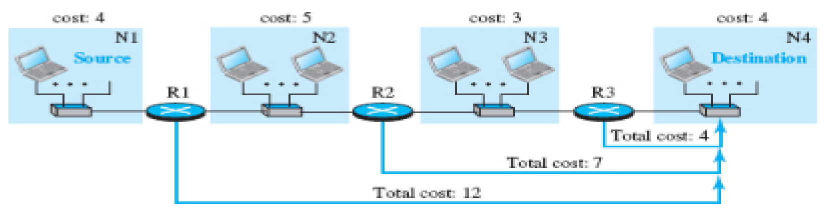
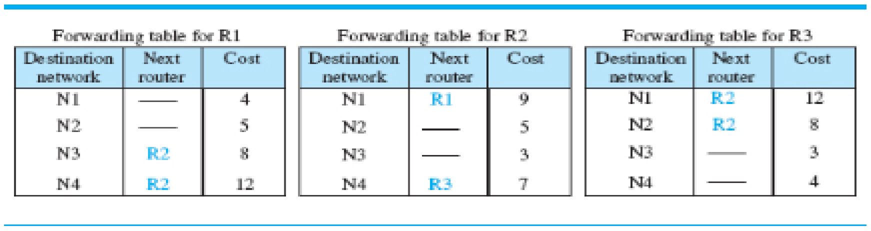


Figure 8.20 *Forwarding tables in OSPF*



Forwarding table for R1

Destination network	Next router	Cost
N1	——	4
N2	——	5
N3	R2	8
N4	R2	12

Forwarding table for R2

Destination network	Next router	Cost
N1	R1	9
N2	——	5
N3	——	3
N4	R3	7

Forwarding table for R3

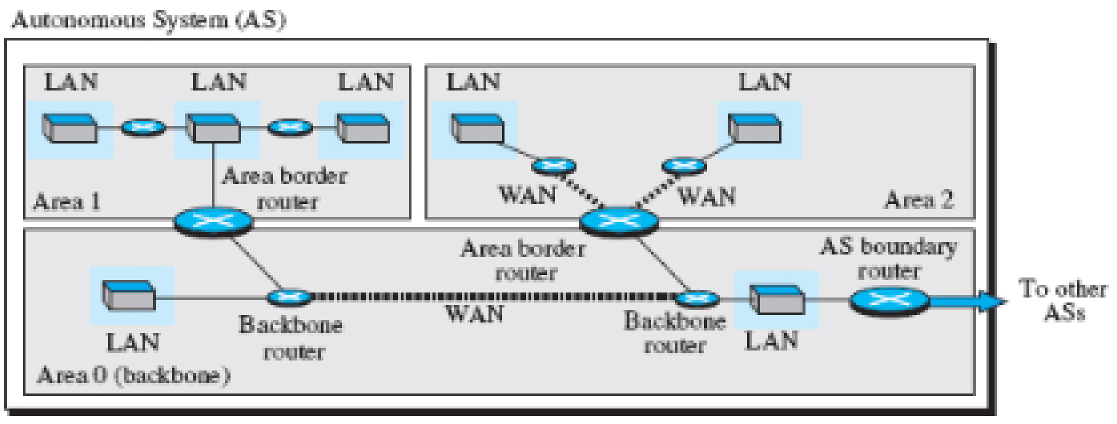
Destination network	Next router	Cost
N1	R2	12
N2	R2	8
N3	——	3
N4	——	4

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.**

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 8.21 shows an autonomous system and its areas.

Figure 8.21 *Areas in an autonomous system*

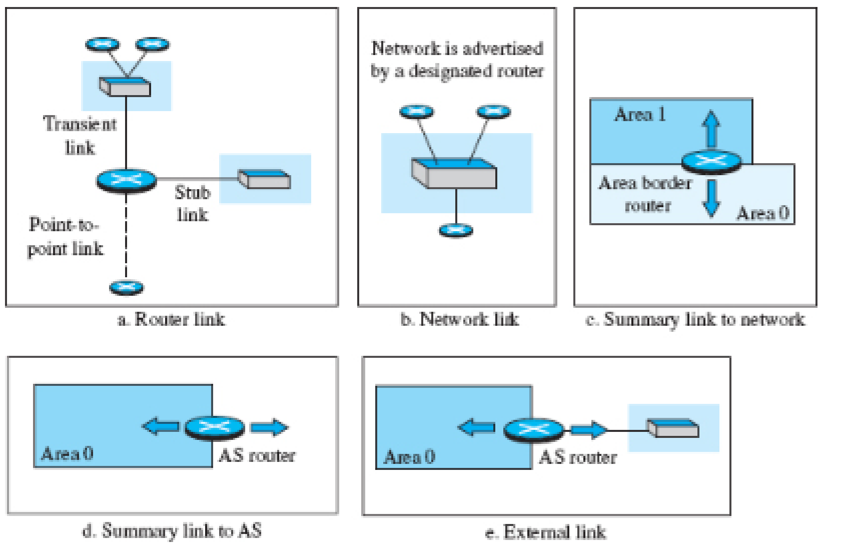


Link-State Advertisement

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 8.22 shows these five advertisements and their uses.

Figure 8.22 *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, because 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.** 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 that uses 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

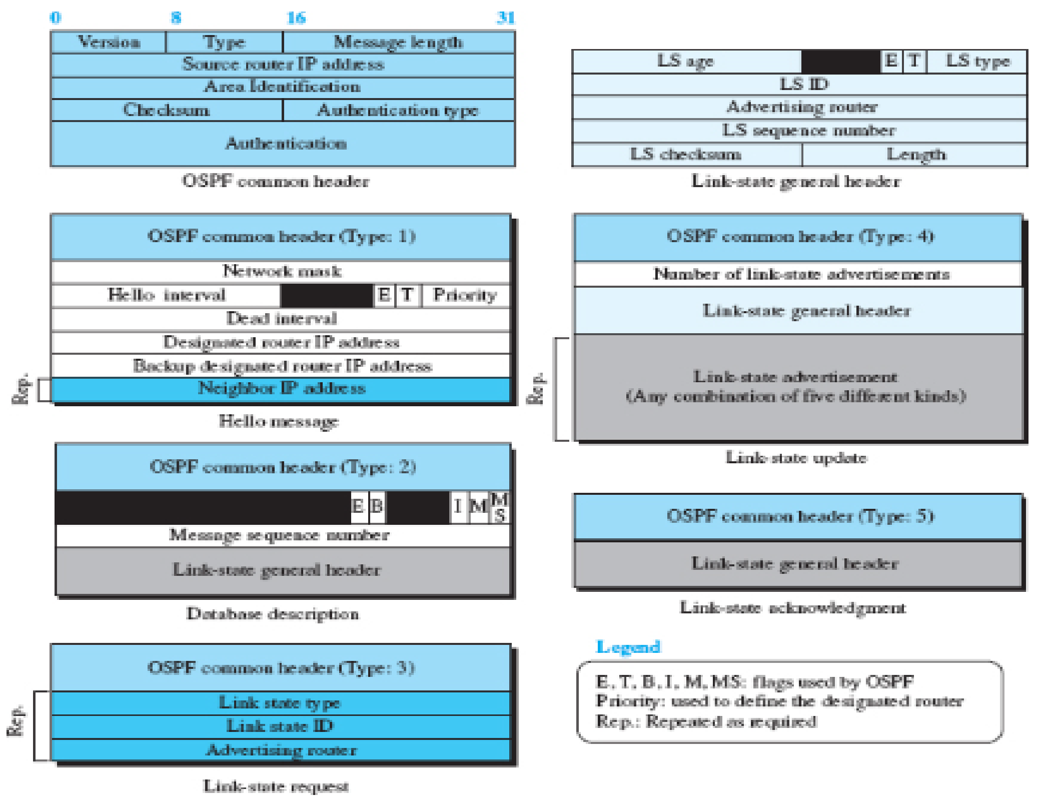
OSPF is a very complex protocol; it uses five different types of messages.

In Figure 8.23, 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 announces 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 in Section 8.2.2.
- 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.

Figure 8.23 *OSPF message formats*


0
8
16
31
Version
Type
Message length
Source router IP address
Area Identification
Checksum
Authentication type
Authentication
OSPF common header
LS age
E T
LS type
LS ID
Advertising router
LS sequence number
LS checksum
Length
Link-state general header
OSPF common header (Type: 1)
Network mask
Hello interval
E T
Priority
Dead interval
Designated router IP address
Backup designated router IP address
Neighbor IP address
Rep.
Hello message
OSPF common header (Type: 4)
Number of link-state advertisements
Link-state general header
Link-state advertisement (Any combination of five different kinds)
Rep.
Link-state update
OSPF common header (Type: 2)
E B
I M MS
Message sequence number
Link-state general header
Database description
OSPF common header (Type: 5)
Link-state general header
Link-state acknowledgment
OSPF common header (Type: 3)
Link state type
Link state ID
Advertising router
Rep.
Link-state request
Legend
E, T, B, I, M, MS: flags used by OSPF
Priority: used to define the designated router
Rep.: Repeated as required

Authentication

As Figure 8.23 shows, the OSPF common header has the provision for authentication of the message sender. As we will discuss in Chapter 13, 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 Section 8.2.2. 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.
- 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 the 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.

8.3.4 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.

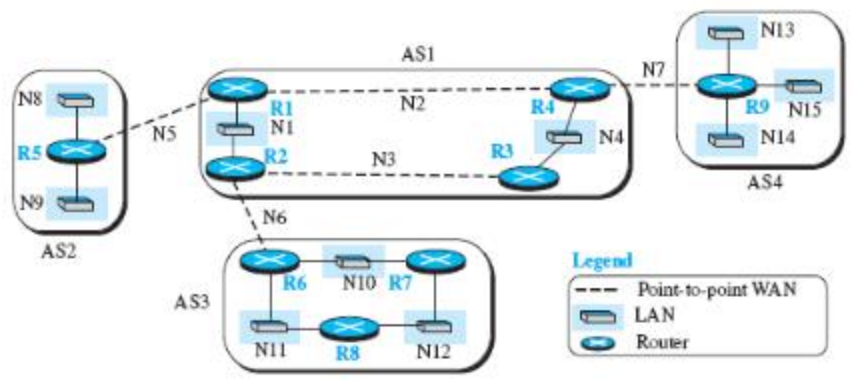
Introduction

Border Gateway Protocol (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 8.24 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

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 that 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.

Figure 8.24 *A sample internet with four ASs*



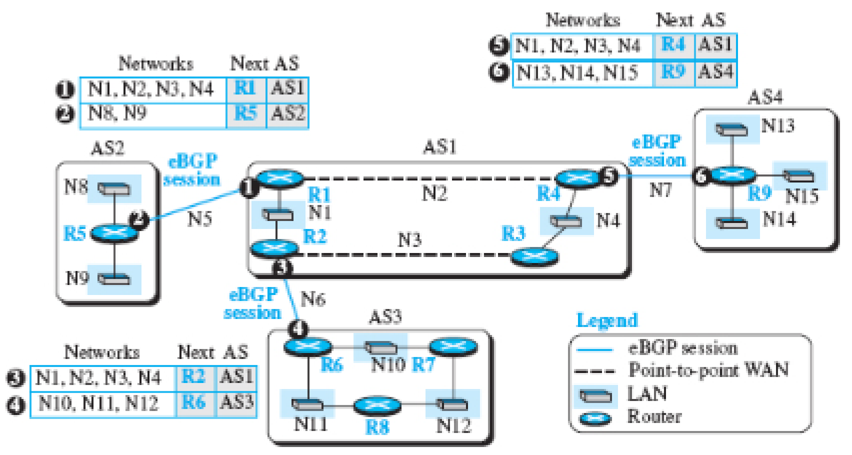
Operation of External BGP (eBGP)

We can say that the GP protocol 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 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 8.24 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 8.25.

Figure 8.25 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.

Figure 8.25 *eBGP operation*



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 non-neighbor 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 AS4, and R9 does not know about networks in AS3.**

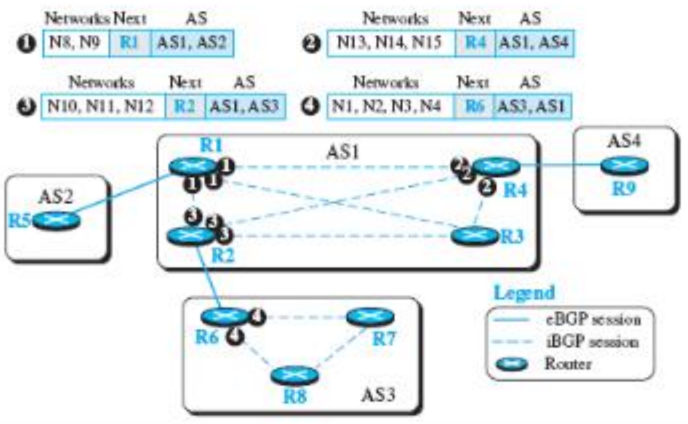
2. **None of the nonborder routers know how to route a packet destined for any networks in other ASs.**

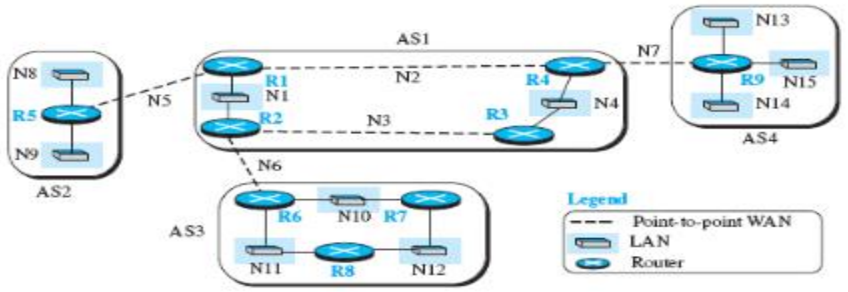
To address these two problems, **we need to allow all pairs of routers** (border or nonborder) to run the second variation of the BGP protocol, **iBGP**.

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* × (*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 8.26 shows the combination of eBGP and iBGP sessions in our internet.

Figure 8.26 *Combination of eBGP and iBGP sessions in our internet*

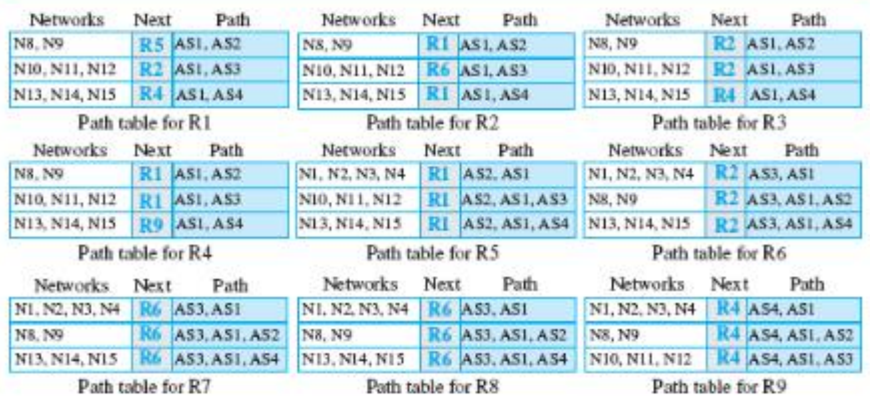




Note that we have not shown the physical networks inside ASs because a session is made on an overlay network (TCP connection), possibly spanning more than one physical network as determined by the route dictated by the intradomain routing protocol. Also note that in this stage only four messages are 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. To demonstrate, we show the path tables in Figure 8.27 for the routers in Figure 8.24. 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 8.27 *Finalized BGP path tables*



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

Path table for R1

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

Path table for R2

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

Path table for R3

Networks	Next	Path
N8, N9	R1	AS1, AS2
N10, N11, N12	R1	AS1, AS3
N13, N14, N15	R9	AS1, AS4

Path table for R4

Networks	Next	Path
N1, N2, N3, N4	R1	AS2, AS1
N10, N11, N12	R1	AS2, AS1, AS3
N13, N14, N15	R1	AS2, AS1, AS4

Path table for R5

Networks	Next	Path
N1, N2, N3, N4	R2	AS3, AS1
N8, N9	R2	AS3, AS1, AS2
N13, N14, N15	R2	AS3, AS1, AS4

Path table for R6

Networks	Next	Path
N1, N2, N3, N4	R6	AS3, AS1
N8, N9	R6	AS3, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4

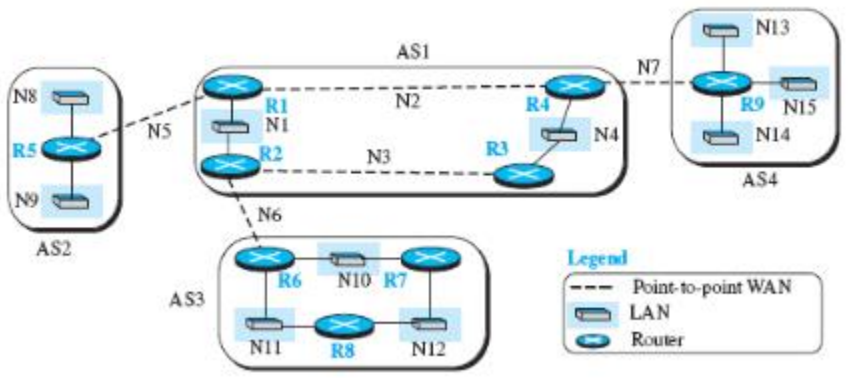
Path table for R7

Networks	Next	Path
N1, N2, N3, N4	R6	AS3, AS1
N8, N9	R6	AS3, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4

Path table for R8

Networks	Next	Path
N1, N2, N3, N4	R4	AS4, AS1
N8, N9	R4	AS4, AS1, AS2
N10, N11, N12	R4	AS4, AS1, AS3

Path table for R9



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 BPG 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.

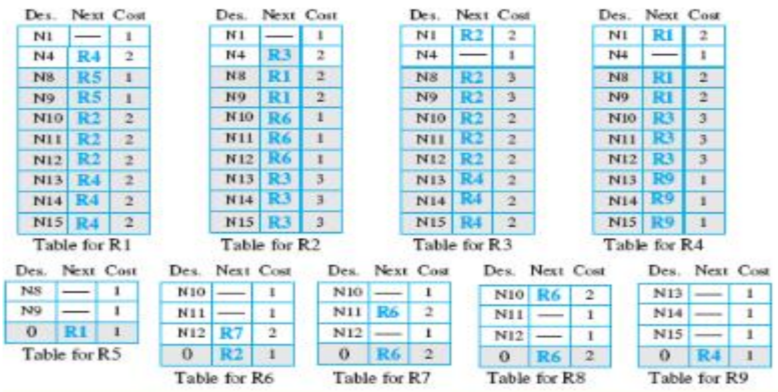
In the case of a stub AS, **the only area border router adds a default entry at the end of its forwarding table and defines the next router to be the speaker router at the end of the eBGP connection.** In Figure 8.24, router R5 in AS2 defines R1 as the default router for all networks other than N8 and 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 routers to be R6. These settings are in accordance with the path tables we describe in Figure 8.27 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**. **Router R1 in AS1 needs to inject the whole contents of the path table for R1** in Figure 8.27 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 8.28 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 8.28 *Forwarding tables after injection from BGP*



Des.	Next	Cost
N1	—	1
N4	R4	2
N8	R5	1
N9	R5	1
N10	R2	2
N11	R2	2
N12	R2	2
N13	R4	2
N14	R4	2
N15	R4	2

Table for R1

Des.	Next	Cost
N1	—	1
N4	R3	2
N8	R1	2
N9	R1	2
N10	R6	1
N11	R6	1
N12	R6	1
N13	R3	3
N14	R3	3
N15	R3	3

Table for R2

Des.	Next	Cost
N1	R2	2
N4	—	1
N8	R2	3
N9	R2	3
N10	R2	2
N11	R2	2
N12	R2	2
N13	R4	2
N14	R4	2
N15	R4	2

Table for R3

Des.	Next	Cost
N1	R1	2
N4	—	1
N8	R1	2
N9	R1	2
N10	R3	3
N11	R3	3
N12	R3	3
N13	R9	1
N14	R9	1
N15	R9	1

Table for R4

Des.	Next	Cost
N8	—	1
N9	—	1
0	R1	1

Table for R5

Des.	Next	Cost
N10	—	1
N11	—	1
N12	R7	2
0	R2	1

Table for R6

Des.	Next	Cost
N10	—	1
N11	R6	2
N12	—	1
0	R6	2

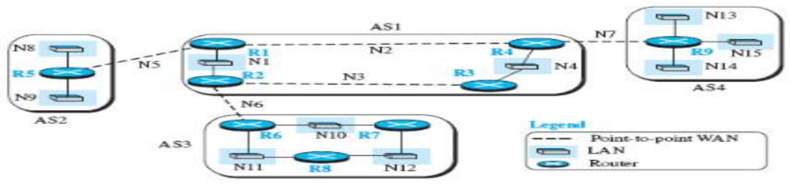
Table for R7

Des.	Next	Cost
N10	R6	2
N11	—	1
N12	—	1
0	R6	2

Table for R8

Des.	Next	Cost
N13	—	1
N14	—	1
N15	—	1
0	R4	1

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. 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 14.18.20.0/26 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.

BGP 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. Interdomain 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 does not. 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. The format for an attribute is shown in Figure 8.29.

Figure 8.29 *Format of a path attribute*

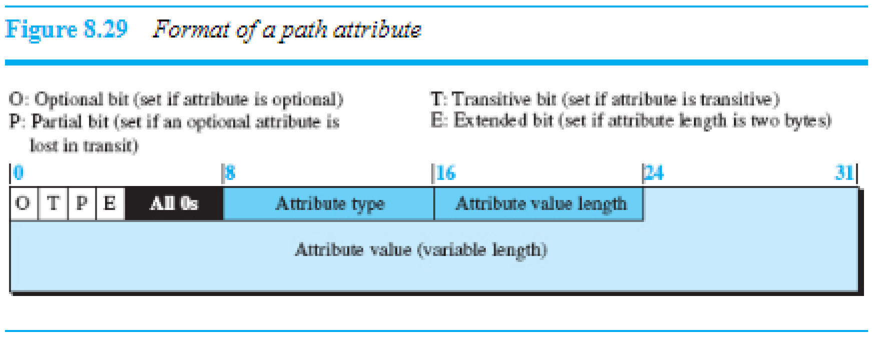


Figure 8.29 Format of a path attribute
O: Optional bit (set if attribute is optional)
P: Partial bit (set if an optional attribute is lost in transit)
T: Transitive bit (set if attribute is transitive)
E: Extended bit (set if attribute length is two bytes)
0
8
16
24
31
O
T
P
E
All 0s
Attribute type
Attribute value length
Attribute value (variable length)

The first byte in each attribute defines the four attribute flags (as shown in Figure 8.29). 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.**

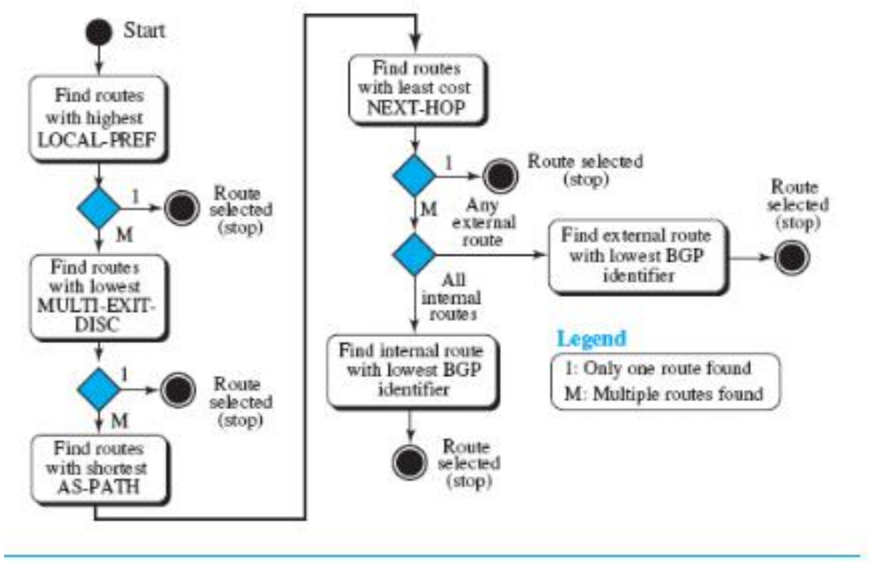
- **AS-PATH (type 2).** This is **a well-known** mandatory attribute, which defines **the list of autonomous systems through which the destination can be reached.** We have used this attribute in our examples. The AS-PATH attribute, as we discussed in path-vector routing in Section 8.2.3, helps prevent 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 nontransitive 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 a 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 nontransitive**, 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 an organization's policy.** The routes the administrator prefers are given a higher local preference value (an attribute value of a 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 8.30 shows the flow diagram used by common implementations.

Figure 8.30 *Flow diagram for route selection*



Start
Find routes with highest LOCAL-PREF
1
Route selected (stop)
M
Find routes with lowest MULTI-EXIT-DISC
1
Route selected (stop)
M
Find routes with shortest AS-PATH
Find routes with least cost NEXT-HOP
1
Route selected (stop)
M
Any external route
Find external route with lowest BGP identifier
Route selected (stop)
All internal routes
Find internal route with lowest BGP identifier
Route selected (stop)
Legend
1: Only one route found
M: Multiple routes found

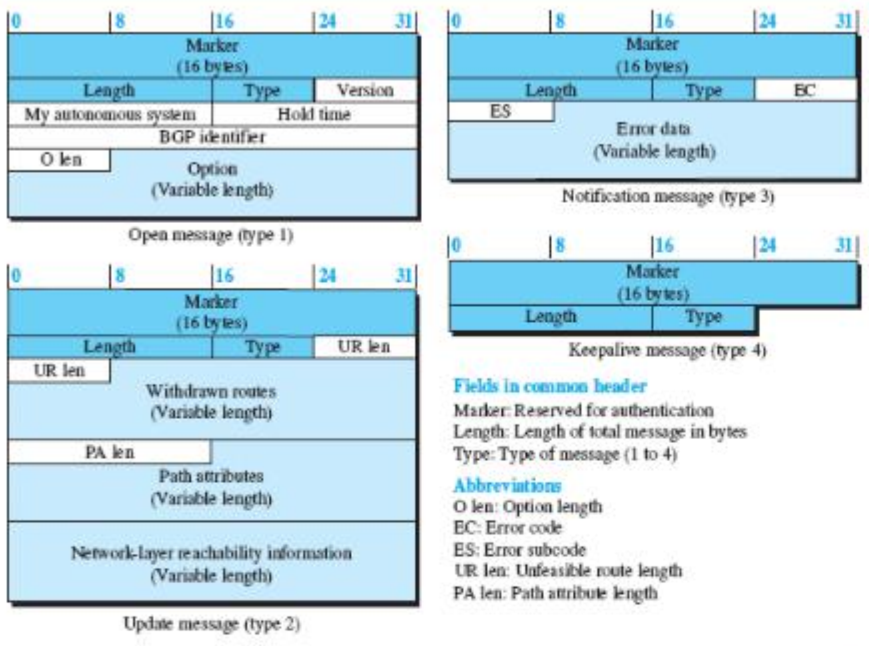
The router extracts the routes that meet the criteria in each step. If only one route is extracted, it is selected and the process stops; otherwise, the process continues with the next step. Note that the first choice is related to the LOCAL-PREF attribute, which reflects the policy imposed by the administration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

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 8.31). All BGP packets share the same common header.

Figure 8.31 *BGP messages*



- **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 keep-alive 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 corruptness also exists in BGP.

8.4 MULTICAST ROUTING

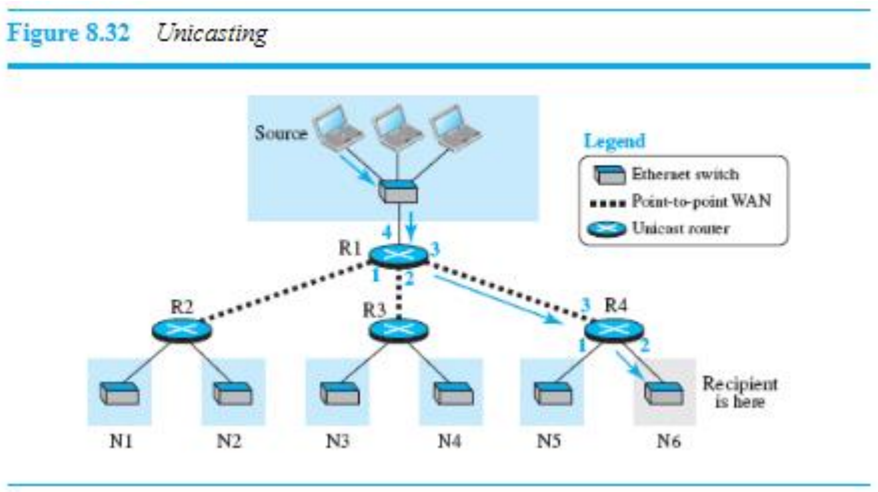
Communication in the Internet today is not only unicasting; multicasting communication is growing fast. In this section, we first discuss the general ideas behind unicasting, multicasting, and broadcasting. We then talk about some basic issues in multicast routing. Finally, we discuss multicasting routing protocols in the Internet.

From the previous chapters, we have learned that forwarding a **datagram by a router is normally based on the prefix of the destination address in the datagram, which defines the network to which the destination host is connected.** Understanding the above forwarding principle, we can now define unicasting, multicasting, and broadcasting. Let us clarify these terms as they relate to the Internet.

8.4.1 Unicasting

In unicasting, there is one source and one destination network. The relationship between the source and the destination network is **one to one**. Each router in the path of the datagram tries to forward the packet to one and only one of its interfaces. Figure 8.32 shows a small internet in which a unicast packet needs to be delivered from a source computer to a destination computer attached to N6. Router R1 is responsible for forwarding the packet only through interface 3; router R4 is responsible for forwarding the packet only through interface 2. When the packet arrives at N6, the delivery to the destination host is the responsibility of the network; either it is broadcast to all hosts, or the Ethernet switch delivers it only to the destination host.

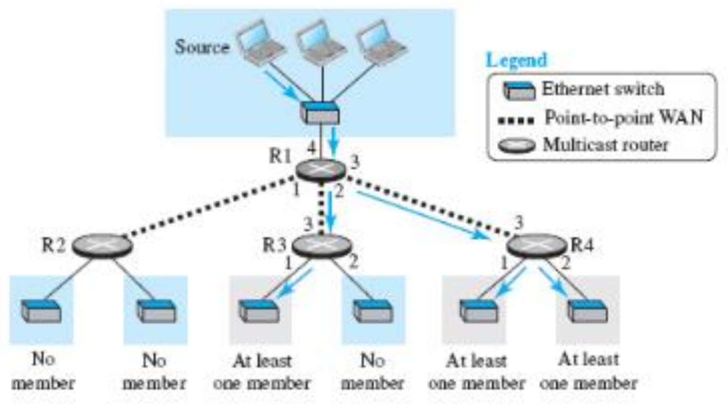
Figure 8.32 *Unicasting*



8.4.2 Multicasting

In multicasting, there is one source and a **group of destinations**. The relationship is one to many. In this type of communication, the source address is a unicast address, but the destination address is a group address, a group of one or more destination networks in which there is at least one member of the group that is interested in receiving the multicast datagram. The group address defines the members of the group. Figure 8.33 shows the same small internet as in Figure 8.32, but the routers have been changed to multicast routers (or previous routers have been configured to do both types of jobs).

Figure 8.33 *Multicasting*



In multicasting, a multicast router may have to send out copies of the same datagram through more than one interface. In Figure 8.33, router R1 needs to send out the datagram through interfaces 2 and 3. Similarly, router R4 needs to send out the datagram through both its interfaces. Router R3, however, knows that there is no member belonging to this group in the area reached by interface 2; it only sends out the datagram through interface 1.

Multicast Applications

Multicasting has many applications today, such as access to distributed databases, information dissemination, teleconferencing, and distance learning.

- **Access to distributed databases.** Most of the large databases today are distributed. That is, the information is stored in more than one location, usually at the time of production. The user who needs to access the database does not know the location of the information. A user's request is multicast to all the database locations, and the location that has the information responds.
- **Information dissemination.** Businesses often need to send information to their customers. If the nature of the information is the same for each customer, it can be multicast. In this way, a business can send one message that can reach many customers. For example, **a software update** can be sent to all purchasers of a particular software package. In a similar manner, news can be easily disseminated through multicasting.
- **Teleconferencing.** Teleconferencing involves multicasting. **The individuals attending a teleconference all need to receive the same information at the same time.** Temporary or permanent groups can be formed for this purpose.
- **Distance learning.** One growing area in the use of multicasting is distance learning**. Lessons taught by one professor can be received by a specific group of students.** This is especially convenient for those students who find it difficult to attend classes on campus.

8.4.3 Distance Vector Multicast Routing Protocol

The **Distance Vector Multicast Routing Protocol (DVMRP)** is the extension of the Routing Information Protocol (RIP) that is used in unicast routing. **It uses the source-based tree approach to multicasting.** It is worth mentioning that **each router in this protocol that receives a multicast packet to be forwarded implicitly creates a source-based multicast tree in three steps**:

1. **The router uses an algorithm called *reverse path forwarding* (RPF) to simulate creation of part of the optimal source-based tree between the source and itself.**
2. **The router uses an algorithm called *reverse path broadcasting* (RPB) to create a broadcast (spanning) tree whose root is the router itself and whose leaves are all networks in the internet.**
3. **The router uses an algorithm called *reverse path multicasting* (RPM) to create a multicast tree by cutting some branches of the tree that end in networks with no member in the group.**

Reverse Path Forwarding (RPF)

The first algorithm, **reverse path forwarding (RPF)**, forces the router to forward a multicast packet from one specific interface: the one which has come through the shortest path from the source to the router. How can a router know which interface is in this path if the router does not have a shortest-path tree rooted at the source? The router uses the first property of the shortest-path tree we discussed in unicast routing, which says that the shortest path from A to B is also the shortest path from B to A. The router does not know the shortest path from the source to itself, but it can find which is the next router in the shortest path from itself to the source (reverse path). The router simply consults **its unicast forwarding table, pretending that it wants to send a packet to the source**; the forwarding table gives the next router and the interface the message that the packet should be sent out from in this reverse direction. The router uses this information to accept a multicast packet only if it arrives from this interface. This is needed to prevent looping. In multicasting, a packet may arrive at the same router that has forwarded it. If the router does not drop all arrived packets except the one, multiple copies of the packet will be circulating in the internet. Of course, the router may add a tag to the packet when it arrives the first time and discard packets that arrive with the same tag, but the RPF strategy is simpler.

Reverse Path Broadcasting (RPB)

The RPF algorithm helps a router to forward only one copy received from a source and drop the rest. However, when we think about broadcasting in the second step, we need to remember that destinations are all the networks (LANs) in the internet. To be efficient, we need to prevent each network from receiving more than one copy of the packet. If a network is connected to more than one router, it may receive a copy of the packet from each router. RPF cannot help here, because a network does not have the intelligence to apply the RPF algorithm; we need to allow only one of the routers attached to a network to pass the packet to the network. One way to do so is to designate only one router as the *parent* of a network related to a specific source. When a router that is not the parent of the attached network receives a multicast packet, it simply drops the packet. There are several ways that the parent of the network related to a network can be selected; one way is to select the router that has the shortest path to the source (using the unicast forwarding table, again in the reverse direction). If there is a tie: in this case, the router with the smaller IP address can be selected. The reader may have noticed that RPB actually creates a broadcast tree from the graph that has been created by the RPF algorithm. RPB has cut those branches of the tree that cause cycles in the graph. If we use the shortest path criteria for choosing the parent router, we have actually created a shortest-path broadcast tree. In other words, after this step, we have a shortest-path tree with the source as the root and all networks (LANs) as the leaves. Every packet started from the source reaches all LANs in the internet traveling the shortest path. Figure 8.34 shows how RPB can avoid duplicate reception in a network by assigning a designated parent router, R1, for network N.

Reverse Path Multicasting (RPM)

As you may have noticed, RPB does not multicast the packet; it broadcasts it. This is not efficient. To increase efficiency, **the multicast packet must reach only those networks that have active members for that particular group.** This is called **reverse path multicasting (RPM)**. To change the broadcast shortest-path tree to a multicast shortest-path tree, each router needs to prune (make inactive) the interfaces that do not reach a network with active members corresponding to a particular source-group combination. This step can be done bottom-up, from the leaves to the root. At the leaf level, the routers connected to the network collect the membership information using the Internet Group Management Protocol (IGMP). The parent router of the network can then disseminate this information upward using the reverse shortest-path tree from the router to the source, the same way as the distance-vector messages are passed from one neighbor to another. When a router receives all these membership-related messages, it knows which interfaces need to be pruned. Of course, because these packets are disseminated periodically, if a new member is added to some networks, all routers are informed and can change the status of their interfaces accordingly. Joining and leaving is continuously applied. Figure 8.35 shows how pruning in RPM lets only networks with group members receive a copy of the packet unless they are in the path to a network with a member.

Figure 8.34 *RPF versus RPB*

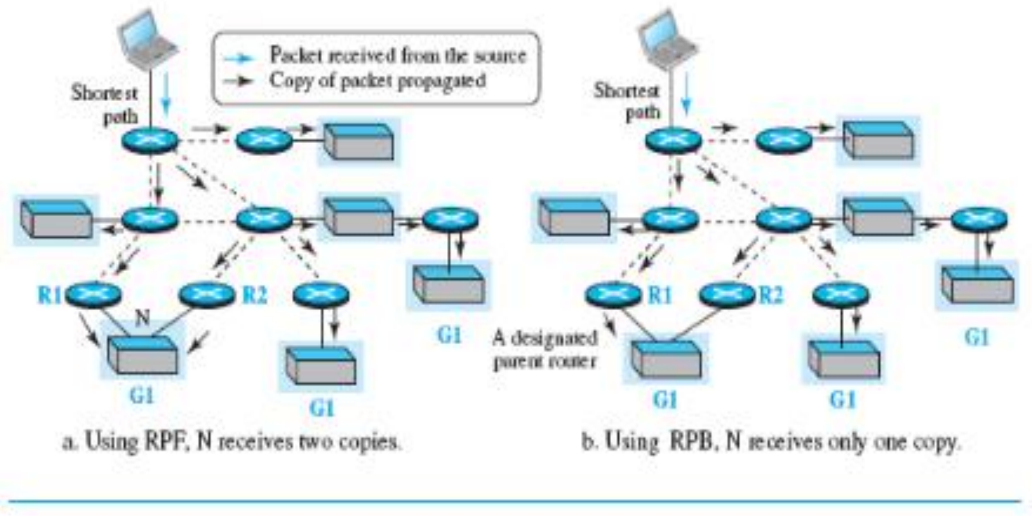
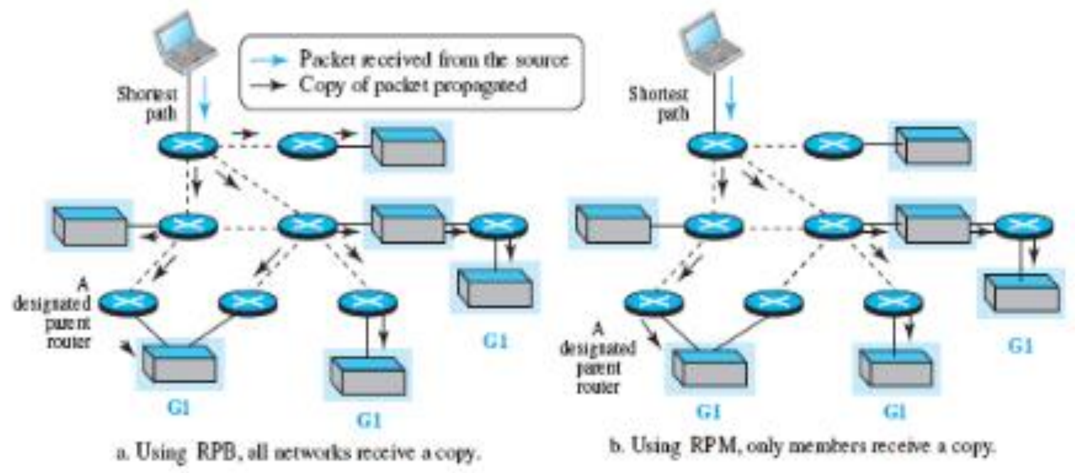

Packet received from the source
Copy of packet propagated
Shortest path
R1
R2
N
G1
G1
G1
a. Using RPF, N receives two copies.
Shortest path
R1
R2
A designated parent router
G1
G1
G1
b. Using RPB, N receives only one copy.

Figure 8.35 *RPB versus RPM*


Packet received from the source
Copy of packet propagated
Shortest path
A designated parent router
G1
a. Using RPB, all networks receive a copy.
b. Using RPM, only members receive a copy.

8.5 IGMP

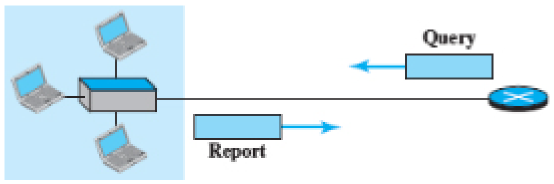
The protocol that is used today for **collecting information about group membership** is the **Internet Group Management Protocol (IGMP)**. IGMP is a protocol defined at the **network layer**; it is one of the auxiliary protocols, like ICMP, that is considered part of the IP. IGMP messages, like ICMP messages, are encapsulated in an IP datagram.

8.5.1 Messages

There are only **two types of messages in IGMP**, version 3: query and report messages, as shown in Figure 8.39. A **query message** is periodically sent by a router to all hosts attached to it to ask them to report their interests about membership in groups. A **report message** is sent by a host as a response to a query message.

Figure 8.39 *IGMP operation*

Figure 8.39 *IGMP operation*



Query Message

The query message **is sent by a router to all hosts** in each interface to collect information about their membership. There are three versions of a query message:

a. **A *general* query message** is sent about membership in any group. It is encapsulated in a datagram with the destination address **224.0.0.1** (all hosts and routers). Note that all routers attached to the same network receive this message to inform them that this message is already sent and that they should refrain from resending it.

b. **A *group-specific* query message** is sent from a router to ask about the membership related to a **specific group.** This is sent when a router does not receive a response about a specific group and wants to be sure that there is no active member of that group in the network. The group identifier (multicast address) is mentioned in the message. The message is encapsulated in a datagram with the destination address set to the corresponding multicast address. Although all hosts receive this message, those not interested drop it.

c. **A *source-and-group-specific* query message** is sent from a router to ask about the membership related to a specific group when the message comes from a specific source or sources. Again the message is sent when the router does not hear about a specific group related to a specific host or hosts. The message is encapsulated in a datagram with the destination address set to the corresponding multicast address. Although all hosts receive this message, those not interested drop it.

Report Message

A report message is sent by a host as a **response to a query message**. The message contains a list of records in which each record gives the identifier of the corresponding group (multicast address) and the addresses of all sources that the host is interested in receiving messages from (inclusion). The record can also mention the source addresses from which the host does not desire to receive a group message (exclusion). The message is encapsulated in a datagram with the multicast address 224.0.0.22 (multicast address assigned to IGMPv3). In IGMPv3, if a host needs to join a group, it waits until it receives a query message and then sends a report message. If a host needs to leave a group, it does not respond to a query message. If no other host responds to the corresponding message, the group is purged from the router database.

8.5.2 Propagation of Membership Information

After a router has collected membership information from the hosts and other routers at its own level in the tree, it can propagate it to the router located in a higher level of the tree. Finally, the router at the tree root can get the membership information to build the multicast tree. The process, however, is more complex than what we can explain in one paragraph. Interested readers can check the book website for the complete description of this protocol.

8.5.3 Encapsulation

The IGMP message is encapsulated in an **IP datagram** with the value of the **protocol** field set to **2** and the **TTL** field set to **1**. The destination IP address of the datagram, however, depends on the type of the message, as shown in Table 8.4.

Table 8.4 *Destination IP Addresses*

Message Type	*IP Address*
General query	224.0.0.1
Other queries	Group address
Report	224.0.0.22