CHAPTER 19

Network-Layer Protocols

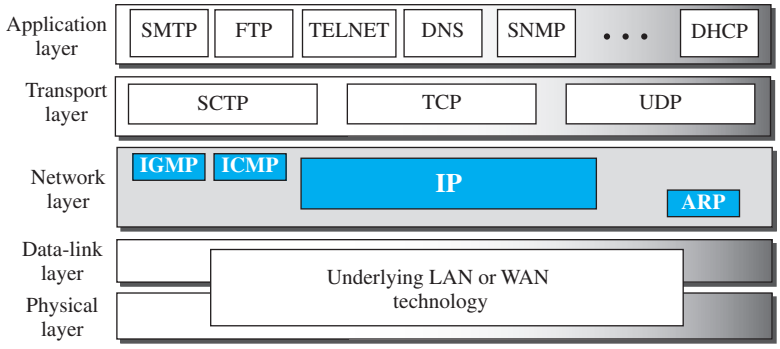
In the previous chapter we introduced the network layer and discussed the services provided by this layer. We also discussed the logical addresses used in this layer. In this chapter, we show how the network layer is implemented in the TCP/IP protocol suite. The protocols in the network layer have gone through a few versions; in this chapter, we concentrate on the current version (v4). The next generation, which is on the horizon, is discussed in Chapter 22.

- The first section discusses the IPv4 protocol. It first describes the IPv4 datagram format. It then explains the purpose of fragmentation in a datagram. The section then briefly discusses options fields and their purpose in a datagram. The section finally mentions some security issues in IPv4, which are addressed in Chapter 32.
- The second section discusses ICMPv4, one of the auxiliary protocols used in the network layer to help IPv4. First, it briefly discusses the purpose of each option. The section then shows how ICMP can be used as a debugging tool. The section finally shows how the checksum is calculated for an ICMPv4 message.
- The third section discusses the mobile IP, whose use is increasing every day when people temporarily move their computers from one place to another. The section first describes the issue of address change in this situation. It then shows the three phases involved in the process. The section finally explains the inefficiency involved in this process and some solutions.

19.1 INTERNET PROTOCOL (IP)

The network layer in version 4 can be thought of as one main protocol and three auxiliary ones. 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 19.1 shows the positions of these four protocols in the TCP/IP protocol suite.

Figure 19.1 *Position of IP and other network-layer protocols in TCP/IP protocol suite*



We will discuss IPv4 and ICMPv4 in this chapter. IGMP will be discussed when we talk about multicasting in Chapter 21. We have discussed ARP in Chapter 9 when we talked about link-layer addresses.

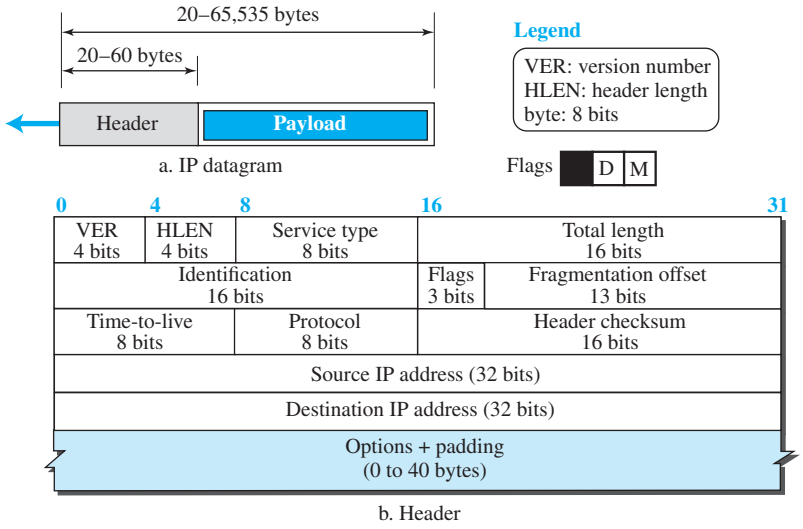
IPv4 is an unreliable datagram protocol—a best-effort delivery service. The term *best-effort* means that IPv4 packets can be corrupted, be lost, arrive out of order, or be delayed, and may create congestion for the network. If reliability is important, IPv4 must be paired with a reliable transport-layer protocol such as TCP. An example of a more commonly understood 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, it is up to the sender or would-be recipient to discover this. The post office itself does not keep track of every letter and cannot notify a sender of loss or damage of one.

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.

19.1.1 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 19.2 shows the IPv4 datagram format. A datagram is a variable-length packet consisting of two parts: header and payload (data). The header is 20 to 60 bytes in length and contains information essential to routing and delivery. It is customary in TCP/IP to show the header in 4-byte sections.

Figure 19.2 *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 is encapsulated in the packet, starts. 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, IETF redefined the field to provide *differentiated services* (DiffServ).

When we discuss differentiated services in Chapter 30, we will be in a better situation to define the bits in this field. The use of 4-byte words for the length header is also logical because the IP header always needs 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 to 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.

$$\text{Length of data} = \text{total length} - (\text{HLEN}) \times 4$$

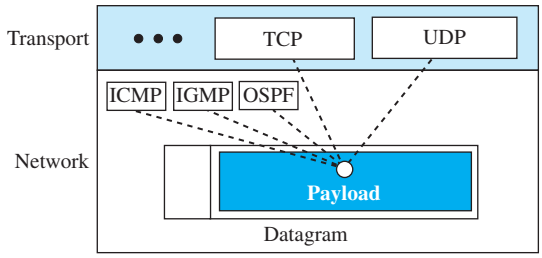
Though a size of 65,535 bytes might seem large, the size of the IPv4 datagram may increase in the near future as the underlying technologies allow even more throughput (greater bandwidth).

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 in the next section.
- ***Time-to-live.*** Due to some malfunctioning of routing protocols (discussed later) 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 two times the maximum number of routers 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 IP a unique 8-bit number which 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 19.3. Note that the protocol fields at the network layer play the same role as the port numbers at the transport layer (Chapters 23 and 24). 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 19.3 *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 adds a header checksum field to check the header, but not the payload. We need to remember that, since 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 discussed in Chapter 10, checksum in the Internet normally uses a 16-bit field, which is the complement of the sum of other fields calculated using 1s complement arithmetic.
- ***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 26. 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 in Chapter 18.

- ***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 one-byte and multi-byte options that we will briefly discuss later in the chapter. The complete discussion is posted at the book website.
- ***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 19.1

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 19.2

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 19.3

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 19.4

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? The data belong to what upper-layer protocol?

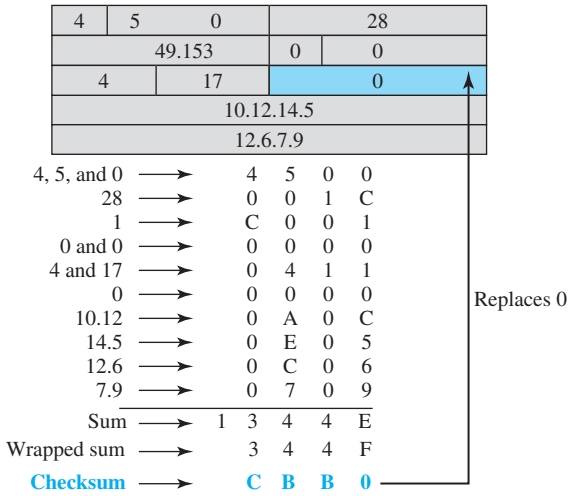
Solution

To find the time-to-live field, we skip 8 bytes (16 hexadecimal digits). The time-to-live field is the ninth byte, which is $(01)_{16}$. 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 19.5

Figure 19.4 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 19.4 *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

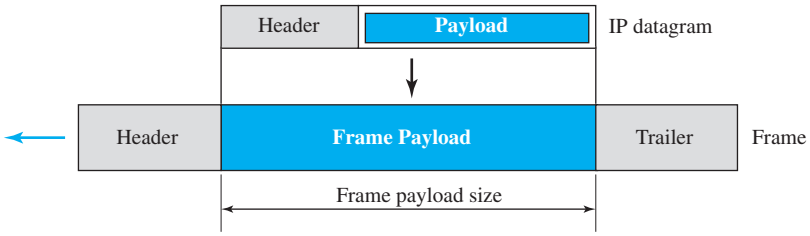
19.1.2 Fragmentation

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 19.5).

Figure 19.5 *Maximum transfer unit (MTU)*



The value of the 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.

In order to make the IP protocol independent of the physical network, the designers decided to make the maximum length of the IP datagram equal to 65,535 bytes. This makes transmission more efficient if 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 of the fragments belonging to the same datagram should finally arrive at the destination host. So it is logical to do the reassembly at the final destination. An even stronger objection for reassembling packets during the transmission is the loss of efficiency it incurs.

When 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

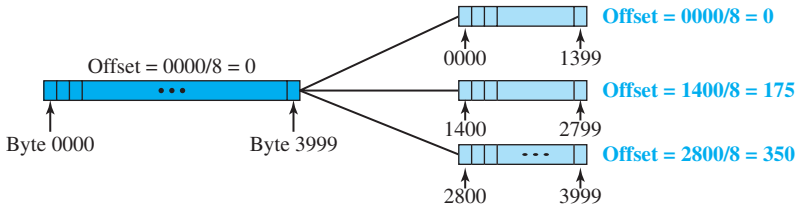
We mentioned before that three fields in an IP datagram are related to fragmentation: *identification, flags,* and *fragmentation offset.* Let us explain these fields now.

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 later). 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. Figure 19.6 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.

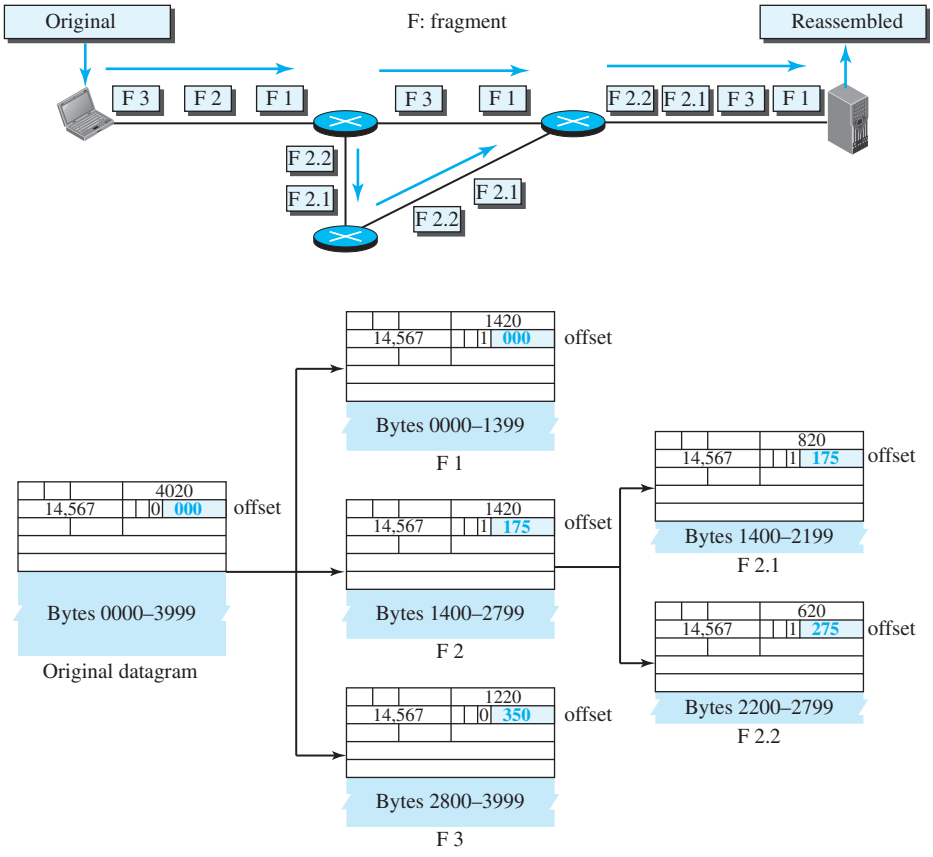
Figure 19.6 *Fragmentation example*



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 19.7 shows an expanded view of the fragments in the previous figure. 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 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 19.7 *Detailed fragmentation example*



The figure also shows what happens if a fragment itself is fragmented. In this case the value of the offset field is always relative to the original datagram. For example, in the figure, the second fragment is itself fragmented later 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 a *more* bit value of 0.

Example 19.6

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

Solution

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

Example 19.7

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 19.8

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 19.9

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 19.10

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.

19.1.3 Options

The header of the IPv4 datagram is made of two parts: a fixed part and a variable part. The fixed part is 20 bytes long and was discussed in the previous section. The variable part comprises the options that can be a maximum of 40 bytes (in multiples of 4-bytes) to preserve the boundary of the header.

Options, as the name implies, are not required for a datagram. They can be used for network testing and debugging. Although options are not a required part of the IPv4 header, option processing is required of the IPv4 software. This means that all implementations must be able to handle options if they are present in the header. Options are divided into two broad categories: single-byte options and multiple-byte options. We give a brief description of options here; for a complete description, see the book website under Extra Materials.

The complete discussion of options in IPv4 is included in the book website under Extra Materials for Chapter 19.

Single-Byte Options

There are two single-byte options.

No Operation

A *no-operation option* is a 1-byte option used as a filler between options.

End of Option

An *end-of-option option* is a 1-byte option used for padding at the end of the option field. It, however, can only be used as the last option.

Multliple-Byte Options

There are four multiple-byte options.

Record Route

A *record route option* is used to record the Internet routers that handle the datagram. It can list up to nine router addresses. It can be used for debugging and management purposes.

Strict Source Route

A *strict source route option* is used by the source to predetermine a route for the datagram as it travels through the Internet. Dictation of a route by the source can be useful for several purposes. The sender can choose a route with a specific type of service, such as minimum delay or maximum throughput. Alternatively, it may choose a route that is

safer or more reliable for the sender's purpose. For example, a sender can choose a route so that its datagram does not travel through a competitor's network.

If a datagram specifies a strict source route, all the routers defined in the option must be visited by the datagram. A router must not be visited if its IPv4 address is not listed in the datagram. If the datagram visits a router that is not on the list, the datagram is discarded and an error message is issued. If the datagram arrives at the destination and some of the entries were not visited, it will also be discarded and an error message issued.

Loose Source Route

A *loose source route option* is similar to the strict source route, but it is less rigid. Each router in the list must be visited, but the datagram can visit other routers as well.

Timestamp

A *timestamp option* is used to record the time of datagram processing by a router. The time is expressed in milliseconds from midnight, Universal time or Greenwich mean time. Knowing the time a datagram is processed can help users and managers track the behavior of the routers in the Internet. We can estimate the time it takes for a datagram to go from one router to another. We say *estimate* because, although all routers may use Universal time, their local clocks may not be synchronized.

19.1.4 Security of IPv4 Datagrams

The IPv4 protocol, as well as the whole Internet, was started when the Internet users trusted each other. No security was provided for the IPv4 protocol. Today, however, the situation is different; the Internet is not secure anymore. Although we will discuss network security in general and IP security in particular in Chapters 31 and 32, here we give a brief idea about the security issues in IP protocol and the solutions. There are three security issues that are particularly applicable to the IP protocol: packet sniffing, packet modification, and IP spoofing.

Packet Sniffing

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

Packet Modification

The second type of attack is to modify the packet. The attacker intercepts the packet, changes its contents, and sends the new packet to the receiver. The receiver believes that the packet is coming from the original sender. This type of attack can be detected using a data integrity mechanism. The receiver, before opening and using the contents of the message, can use this mechanism to make sure that the packet has not been changed during the transmission. We discuss packet integrity in Chapter 32.

IP Spoofing

An attacker can masquerade as somebody else and create an IP packet that carries the source address of another computer. An attacker can send an IP packet to a bank pretending that it is coming from one of the customers. This type of attack can be prevented using an origin authentication mechanism (see Chapter 32).

IPSec

The IP packets today can be protected from the previously mentioned attacks using a protocol called IPSec (IP Security). This protocol, which is used in conjunction with the IP protocol, creates a connection-oriented service between two entities in which they can exchange IP packets without worrying about the three attacks discussed above. We will discuss IPSec in detail in Chapter 32; here it is enough to mention that IPSec provides the following four services:

- ***Defining Algorithms and Keys.*** The two entities that want to create a secure channel between themselves can agree on some available algorithms and keys to be used for security purposes.
- ***Packet Encryption.*** The packets exchanged between two parties can be encrypted for privacy using one of the encryption algorithms and a shared key agreed upon in the first step. This makes the packet sniffing attack useless.
- ***Data Integrity.*** Data integrity guarantees that the packet is not modified during the transmission. If the received packet does not pass the data integrity test, it is discarded. This prevents the second attack, packet modification, described above.
- ***Origin Authentication.*** IPSec can authenticate the origin of the packet to be sure that the packet is not created by an imposter. This can prevent IP spoofing attacks as described above.

19.2 ICMPv4

The IPv4 has no error-reporting or error-correcting mechanism. What happens if something goes wrong? What happens if a router must discard a datagram because it cannot find a route to the final destination, or because the time-to-live field has a zero value? What happens if the final destination host must discard the received fragments of a datagram because it has not received all fragments within a predetermined time limit? These are examples of situations where an error has occurred and the IP protocol has no built-in mechanism to notify the original host.

The IP protocol also lacks a mechanism for host and management queries. A host sometimes needs to determine if a router or another host is alive. And sometimes a network manager needs information from another host or router.

The **Internet Control Message Protocol version 4 (ICMPv4)** has been designed to compensate for the above two deficiencies. It is a companion to the IP protocol. ICMP itself is a network-layer protocol. However, its messages are not passed directly to the data-link layer as would be expected. Instead, the messages are first encapsulated inside IP datagrams before going to the lower layer. When an IP datagram encapsulates

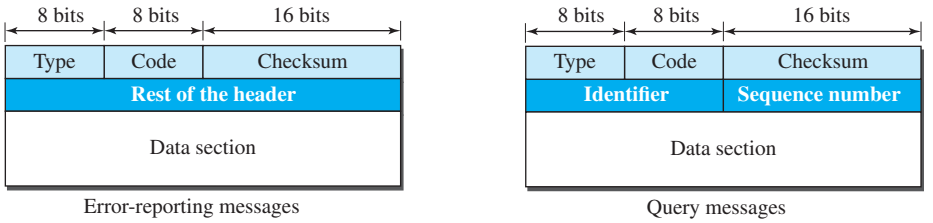
an ICMP message, the value of the protocol field in the IP datagram is set to 1 to indicate that the IP payroll is an ICMP message.

19.2.1 MESSAGES

ICMP messages are divided into two broad categories: *error-reporting messages* and *query messages.* The error-reporting messages report problems that a router or a host (destination) may encounter when it processes an IP packet. The query messages, which occur in pairs, help a host or a network manager get specific information from a router or another host. For example, nodes can discover their neighbors. Also, hosts can discover and learn about routers on their network and routers can help a node redirect its messages.

An ICMP message has an 8-byte header and a variable-size data section. Although the general format of the header is different for each message type, the first 4 bytes are common to all. As Figure 19.8 shows, the first field, ICMP type, defines the type of the message. The code field specifies the reason for the particular message type. The last common field is the checksum field (to be discussed later in the chapter). The rest of the header is specific for each message type.

Figure 19.8 *General format of ICMP messages*



Type and code values

Error-reporting messages	Query messages
03: Destination unreachable (codes 0 to 15)	08 and 00: Echo request and reply (only code 0)
04: Source quench (only code 0)	13 and 14: Timestamp request and reply (only code 0)
05: Redirection (codes 0 to 3)	
11: Time exceeded (codes 0 and 1)	
12: Parameter problem (codes 0 and 1)	

The data section in error messages carries information for finding the original packet that had the error. In query messages, the data section carries extra information based on the type of query.

We give a brief description of the ICMPv4 messages here; for a complete description see the book website under Extra Materials for Chapter 19.

The complete discussion of messages in ICMPv4 is included in the book website under Extra Materials for Chapter 19.

Error Reporting Messages

Since IP is an unreliable protocol, one of the main responsibilities of ICMP is to report some errors that may occur during the processing of the IP datagram. ICMP does not correct errors, it simply reports them. Error correction is left to the higher-level protocols. Error messages are always sent to the original source because the only information available in the datagram about the route is the source and destination IP addresses. ICMP uses the source IP address to send the error message to the source (originator) of the datagram. To make the error-reporting process simple, ICMP follows some rules in reporting messages. First, no error message will be generated for a datagram having a multicast address or special address (such as *this host* or *loopback*). Second, no ICMP error message will be generated in response to a datagram carrying an ICMP error message. Third, no ICMP error message will be generated for a fragmented datagram that is not the first fragment.

Note that all error messages contain a data section that includes the IP header of the original datagram plus the first 8 bytes of data in that datagram. The original datagram header is added to give the original source, which receives the error message, information about the datagram itself. The 8 bytes of data are included because the first 8 bytes provide information about the port numbers (UDP and TCP) and sequence number (TCP). This information is needed so the source can inform the protocols (TCP or UDP) about the error.

The following are important points about ICMP error messages:

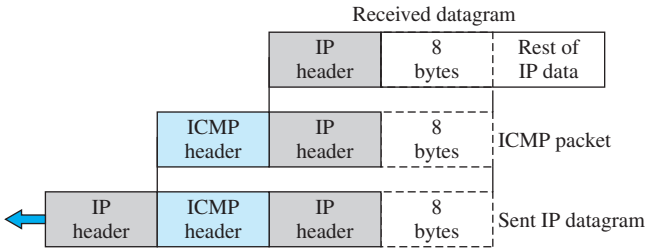
- No ICMP error message will be generated in response to a datagram carrying an ICMP error message.
- No ICMP error message will be generated for a fragmented datagram that is not the first fragment.
- No ICMP error message will be generated for a datagram having a multicast address.
- No ICMP error message will be generated for a datagram having a special address such as 127.0.0.0 or 0.0.0.0.

Note that all error messages contain a data section that includes the IP header of the original datagram plus the first 8 bytes of data in that datagram. The original datagram header is added to give the original source, which receives the error message, information about the datagram itself. The 8 bytes of data are included because, as we will see in Chapter 24 on UDP and TCP protocols, the first 8 bytes provide information about the port numbers (UDP and TCP) and sequence number (TCP). This information is needed so the source can inform the protocols (TCP or UDP) about the error. ICMP forms an error packet, which is then encapsulated in an IP datagram (see Figure 19.9).

Destination Unreachable

The most widely used error message is the destination unreachable (type 3). This message uses different codes (0 to 15) to define the type of error message and the reason why a datagram has not reached its final destination. For example, code 0 tells the

Figure 19.9 *Contents of data field for the error messages*



source that a host is unreachable. This may happen, for example, when we use the HTTP protocol to access a web page, but the server is down. The message "destination host is not reachable" is created and sent back to the source.

Source Quench

Another error message is called the *source quench* (type 4) message, which informs the sender that the network has encountered congestion and the datagram has been dropped; the source needs to slow down sending more datagrams. In other words, ICMP adds a kind of congestion control mechanism to the IP protocol by using this type of message.

Redirection Message

The *redirection message* (type 5) is used when the source uses a wrong router to send out its message. The router redirects the message to the appropriate router, but informs the source that it needs to change its default router in the future. The IP address of the default router is sent in the message.

We discussed the purpose of the *time-to-live* (TTL) field in the IP datagram and explained that it prevents a datagram from being aimlessly circulated in the Internet. When the TTL value becomes 0, the datagram is dropped by the visiting router and a *time exceeded* message (type 11) with code 0 is sent to the source to inform it about the situation. The time-exceeded message (with code 1) can also be sent when not all fragments of a datagram arrive within a predefined period of time.

Parameter Problem

A *parameter problem message* (type 12) can be sent when either there is a problem in the header of a datagram (code 0) or some options are missing or cannot be interpreted (code 1).

Query Messages

Interestingly, query messages in ICMP can be used independently without relation to an IP datagram. Of course, a query message needs to be encapsulated in a datagram, as a carrier. Query messages are used to probe or test the liveliness of hosts or routers in the Internet, find the one-way or the round-trip time for an IP datagram between two devices, or even find out whether the clocks in two devices are synchronized. Naturally, query messages come in pairs: request and reply.

The *echo request* (type 8) and the *echo reply* (type 0) pair of messages are used by a host or a router to test the liveliness of another host or router. A host or router sends

an echo request message to another host or router; if the latter is alive, it responds with an echo reply message. We shortly see the applications of this pair in two debugging tools: *ping* and *traceroute*.

The *timestamp request* (type 13) and the *timestamp reply* (type 14) pair of messages are used to find the round-trip time between two devices or to check whether the clocks in two devices are synchronized. The timestamp request message sends a 32-bit number, which defines the time the message is sent. The timestamp reply resends that number, but also includes two new 32-bit numbers representing the time the request was received and the time the response was sent. If all timestamps represent Universal time, the sender can calculate the one-way and round-trip time.

Deprecated Messages

Three pairs of messages are declared obsolete by IETF:

1. *Information request and replay* messages are not used today because their duties are done by the Address Resolution Protocol (ARP) discussed in Chapter 9.
2. *Address mask request and reply* messages are not used today because their duties are done by the Dynamic Host Configuration Protocol (DHCP), discussed in Chapter 18.
3. *Router solicitation and advertisement* messages are not used today because their duties are done by the Dynamic Host Configuration Protocol (DHCP), discussed in Chapter 18.

19.2.2 Debugging Tools

There are several tools that can be used in the Internet for debugging. We can determine the viability of a host or router. We can trace the route of a packet. We introduce two tools that use ICMP for debugging: *ping* and *traceroute*.

Ping

We can use the *ping* program to find if a host is alive and responding. We use *ping* here to see how it uses ICMP packets. The source host sends ICMP echo-request messages; the destination, if alive, responds with ICMP echo-reply messages. The *ping* program sets the identifier field in the echo-request and echo-reply message and starts the sequence number from 0; this number is incremented by 1 each time a new message is sent. Note that *ping* can calculate the round-trip time. It inserts the sending time in the data section of the message. When the packet arrives, it subtracts the arrival time from the departure time to get the round-trip time (RTT).

Example 19.11

The following shows how we send a *ping* message to the auniversity.edu site. We set the identifier field in the echo request and reply message and start the sequence number from 0; this number is incremented by one each time a new message is sent. Note that *ping* can calculate the round-trip time. It inserts the sending time in the data section of the message. When the packet arrives, it subtracts the arrival time from the departure time to get the *round-trip time* (rtt).

```
$ ping auniversity.edu
PING auniversity.edu (152.181.8.3)  56 (84)  bytes of data.
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=0    ttl=62    time=1.91 ms
```

```
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=1    ttl=62    time=2.04 ms
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=2    ttl=62    time=1.90 ms
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=3    ttl=62    time=1.97 ms
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=4    ttl=62    time=1.93 ms
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=5    ttl=62    time=2.00 ms
--- auniversity.edu statistics ---
6 packets transmitted, 6 received, 0% packet loss
rtt min/avg/max = 1.90/1.95/2.04 ms
```

Traceroute or Tracert

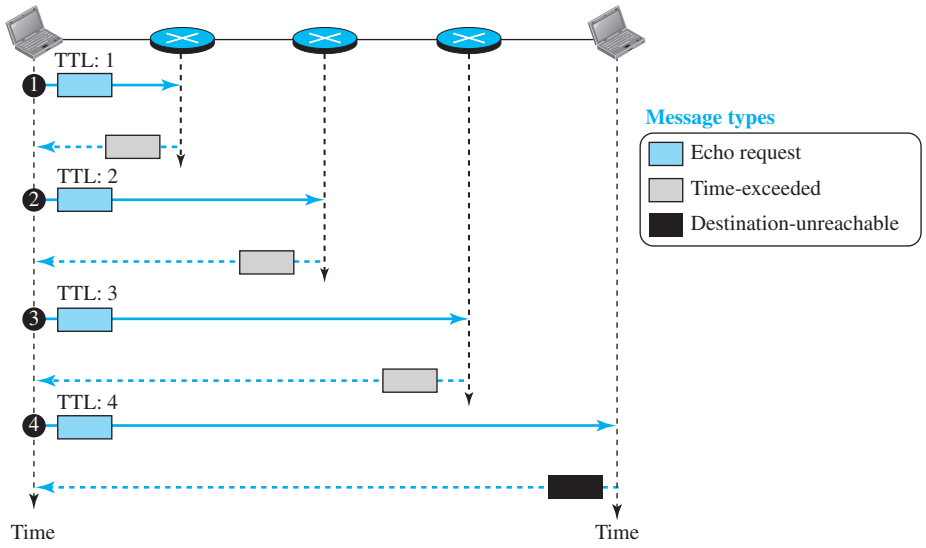
The *traceroute* program in UNIX or *tracert* in Windows can be used to trace the path of a packet from a source to the destination. It can find the IP addresses of all the routers that are visited along the path. The program is usually set to check for the maximum of 30 hops (routers) to be visited. The number of hops in the Internet is normally less than this. Since these two programs behave differently in Unix and Windows, we explain them separately.

Traceroute

The *traceroute* program is different from the *ping* program. The *ping* program gets help from two query messages; the *traceroute* program gets help from two error-reporting messages: time-exceeded and destination-unreachable. The *traceroute* is an application-layer program, but only the client program is needed, because, as we can see, the client program never reaches the application layer in the destination host. In other words, there is no *traceroute* server program. The *traceroute* application program is encapsulated in a UDP user datagram, but *traceroute* intentionally uses a port number that is not available at the destination. If there are n routers in the path, the *traceroute* program sends $(n + 1)$ messages. The first n messages are discarded by the n routers, one by each router; the last message is discarded by the destination host. The *traceroute* client program uses the $(n + 1)$ ICMP error-reporting messages received to find the path between the routers. We will show shortly that the *traceroute* program does not need to know the value of n; it is found automatically. In Figure 19.10, the value of n is 3.

The first *traceroute* message is sent with time-to-live (TTL) value set to 1; the message is discarded at the first router and a time-exceeded ICMP error message is sent, from which the *traceroute* program can find the IP address of the first router (the source IP address of the error message) and the router name (in the data section of the message). The second *traceroute* message is sent with TTL set to 2, which can find the IP address and the name of the second router. Similarly, the third message can find the information about router 3. The fourth message, however, reaches the destination host. This host is also dropped, but for another reason. The destination host cannot find the port number specified in the UDP user datagram. This time ICMP sends a different message, the destination-unreachable message with code 3 to show the port number is not found. After receiving this different ICMP message, the *traceroute* program knows that the final destination is reached. It uses the information in the received message to find the IP address and the name of the final destination.

Figure 19.10 *Use of ICMPv4 in traceroute*



The *traceroute* program also sets a timer to find the round-trip time for each router and the destination. Most *traceroute* programs send three messages to each device, with the same TTL value, to be able to find a better estimate for the round-trip time. The following shows an example of a *traceroute* program, which uses three probes for each device and gets three RTTs.

$ *traceroute* printers.com

***traceroute* to printers.com (13.1.69.93), 30 hops max, 38-byte packets**				
1 route.front.edu	**(153.18.31.254)**	**0.622 ms**	**0.891 ms**	**0.875 ms**
2 ceneric.net	**(137.164.32.140)**	**3.069 ms**	**2.875 ms**	**2.930 ms**
3 satire.net	**(132.16.132.20)**	**3.071 ms**	**2.876 ms**	**2.929 ms**
4 alpha.printers.com	**(13.1.69.93)**	**5.922 ms**	**5.048 ms**	**4.922 ms**

Tracert

The *tracert* program in windows behaves differently. The *tracert* messages are encapsulated directly in IP datagrams. The *tracert,* like *traceroute,* sends echo-request messages. However, when the last echo request reaches the destination host, an echo-replay message is issued.

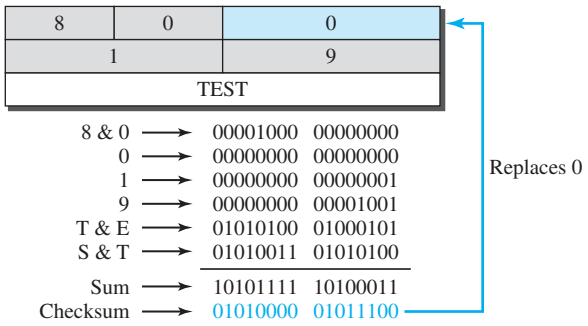
19.2.3 ICMP Checksum

In ICMP the checksum is calculated over the entire message (header and data).

Example 19.12

Figure 19.11 shows an example of checksum calculation for a simple echo-request message. We randomly chose the identifier to be 1 and the sequence number to be 9. The message is divided

Figure 19.11 *Example of checksum calculation*



into 16-bit (2-byte) words. The words are added and the sum is complemented. Now the sender can put this value in the checksum field.

19.3 MOBILE IP

In the last section of this chapter, we discuss mobile IP. As mobile and personal computers such as notebooks become increasingly popular, we need to think about mobile IP, the extension of IP protocol that allows mobile computers to be connected to the Internet at any location where the connection is possible. In this section, we discuss this issue.

19.3.1 Addressing

The main problem that must be solved in providing mobile communication using the IP protocol is addressing.

Stationary Hosts

The original IP addressing was based on the assumption that a host is stationary, attached to one specific network. A router uses an IP address to route an IP datagram. As we learned in Chapter 18, an IP address has two parts: a prefix and a suffix. The prefix associates a host with a network. For example, the IP address 10.3.4.24/8 defines a host attached to the network 10.0.0.0/8. This implies that a host in the Internet does not have an address that it can carry with itself from one place to another. The address is valid only when the host is attached to the network. If the network changes, the address is no longer valid. Routers use this association to route a packet; they use the prefix to deliver the packet to the network to which the host is attached. This scheme works perfectly with **stationary hosts.**

The IP addresses are designed to work with stationary hosts because part of the address defines the network to which the host is attached.

Mobile Hosts

When a host moves from one network to another, the IP addressing structure needs to be modified. Several solutions have been proposed.

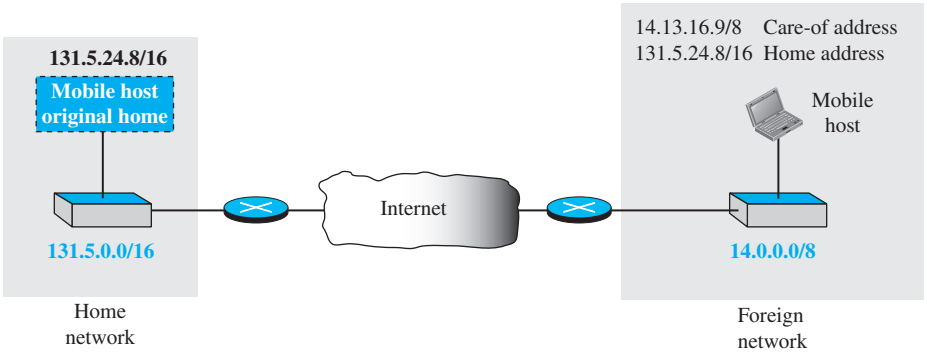
Changing the Address

One simple solution is to let the **mobile host** change its address as it goes to the new network. The host can use DHCP (see Chapter 18) to obtain a new address to associate it with the new network. This approach has several drawbacks. First, the configuration files would need to be changed. Second, each time the computer moves from one network to another, it must be rebooted. Third, the DNS tables (see Chapter 26) need to be revised so that every other host in the Internet is aware of the change. Fourth, if the host roams from one network to another during a transmission, the data exchange will be interrupted. This is because the ports and IP addresses of the client and the server must remain constant for the duration of the connection.

Two Addresses

The approach that is more feasible is the use of two addresses. The host has its original address, called the **home address,** and a temporary address, called the **care-of address.** The home address is permanent; it associates the host with its **home network,** the network that is the permanent home of the host. The care-of address is temporary. When a host moves from one network to another, the care-of address changes; it is associated with the **foreign network,** the network to which the host moves. Figure 19.12 shows the concept.

Figure 19.12 *Home address and care-of address*



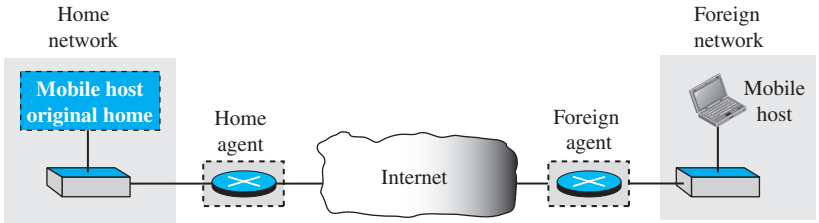
Mobile IP has two addresses for a mobile host: one home address and one care-of address. The home address is permanent; the care-of address changes as the mobile host moves from one network to another.

When a mobile host visits a foreign network, it receives its care-of address during the agent discovery and registration phase, described later.

19.3.2 Agents

To make the change of address transparent to the rest of the Internet requires a **home agent** and a **foreign agent.** Figure 19.13 shows the position of a home agent relative to the home network and a foreign agent relative to the foreign network.

Figure 19.13 *Home agent and foreign agent*



We have shown the home and the foreign agents as routers, but we need to emphasize that their specific function as an agent is performed in the application layer. In other words, they are both routers and hosts.

Home Agent

The home agent is usually a router attached to the home network of the mobile host. The home agent acts on behalf of the mobile host when a remote host sends a packet to the mobile host. The home agent receives the packet and sends it to the foreign agent.

Foreign Agent

The foreign agent is usually a router attached to the foreign network. The foreign agent receives and delivers packets sent by the home agent to the mobile host.

The mobile host can also act as a foreign agent. In other words, the mobile host and the foreign agent can be the same. However, to do this, a mobile host must be able to receive a care-of address by itself, which can be done through the use of DHCP. In addition, the mobile host needs the necessary software to allow it to communicate with the home agent and to have two addresses: its home address and its care-of address. This dual addressing must be transparent to the application programs.

When the mobile host acts as a foreign agent, the care-of address is called a **collocated care-of address.**

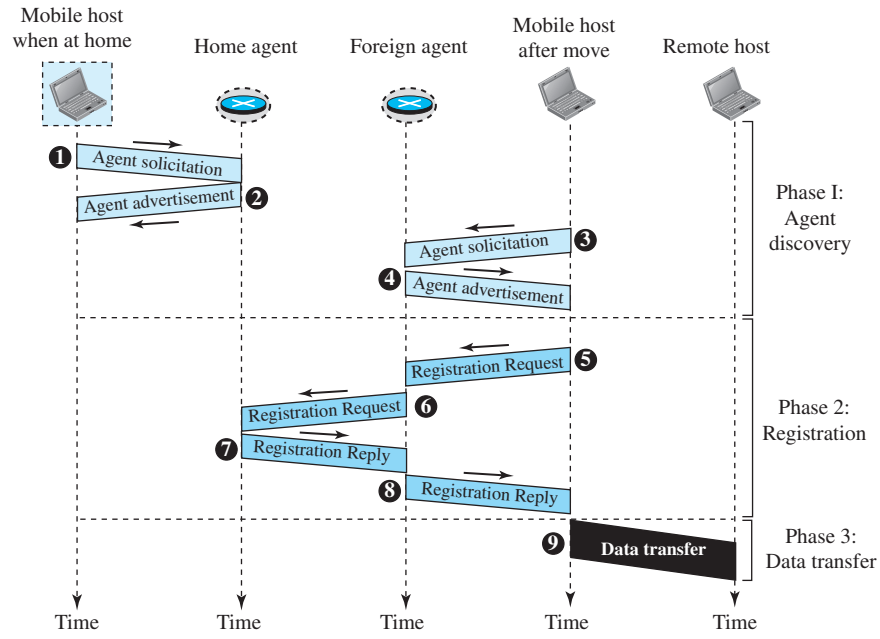
When the mobile host and the foreign agent are the same, the care-of address is called a collocated care-of address.

The advantage of using a collocated care-of address is that the mobile host can move to any network without worrying about the availability of a foreign agent. The disadvantage is that the mobile host needs extra software to act as its own foreign agent.

19.3.3 Three Phases

To communicate with a remote host, a mobile host goes through three phases: agent discovery, registration, and data transfer, as shown in Figure 19.14.

Figure 19.14 *Remote host and mobile host communication*



The first phase, agent discovery, involves the mobile host, the foreign agent, and the home agent. The second phase, registration, also involves the mobile host and the two agents. Finally, in the third phase, the remote host is also involved. We discuss each phase separately.

Agent Discovery

The first phase in mobile communication, *agent discovery,* consists of two subphases. A mobile host must discover (learn the address of) a home agent before it leaves its home network. A mobile host must also discover a foreign agent after it has moved to a foreign network. This discovery consists of learning the care-of address as well as the foreign agent's address. The discovery involves two types of messages: advertisement and solicitation.

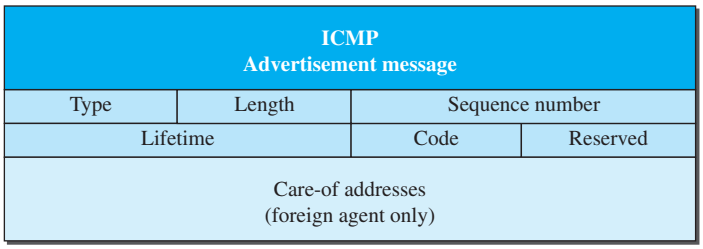
Agent Advertisement

When a router advertises its presence on a network using an ICMP router advertisement, it can append an *agent advertisement* to the packet if it acts as an agent.

Figure 19.15 shows how an agent advertisement is piggybacked to the router advertisement packet.

Mobile IP does not use a new packet type for agent advertisement; it uses the router advertisement packet of ICMP, and appends an agent advertisement message.

Figure 19.15 *Agent advertisement*



The field descriptions are as follows:

- ***Type.*** The 8-bit type field is set to 16.
- ***Length.*** The 8-bit length field defines the total length of the extension message (not the length of the ICMP advertisement message).
- ***Sequence number.*** The 16-bit sequence number field holds the message number. The recipient can use the sequence number to determine if a message is lost.
- ***Lifetime.*** The lifetime field defines the number of seconds that the agent will accept requests. If the value is a string of 1s, the lifetime is infinite.
- ***Code.*** The code field is an 8-bit flag in which each bit is set (1) or unset (0). The meanings of the bits are shown in Table 19.1.

Table 19.1 *Code Bits*

Bit	*Meaning*
0	Registration required. No collocated care-of address.
1	Agent is busy and does not accept registration at this moment.
2	Agent acts as a home agent.
3	Agent acts as a foreign agent.
4	Agent uses minimal encapsulation.
5	Agent uses generic routing encapsulation (GRE).
6	Agent supports header compression.
7	Unused (0).

- ***Care-of Addresses.*** This field contains a list of addresses available for use as care-of addresses. The mobile host can choose one of these addresses. The selection of this care-of address is announced in the registration request. Note that this field is used only by a foreign agent.

Agent Solicitation

When a mobile host has moved to a new network and has not received agent advertisements, it can initiate an *agent solicitation.* It can use the ICMP solicitation message to inform an agent that it needs assistance.

Mobile IP does not use a new packet type for agent solicitation; it uses the router solicitation packet of ICMP.

Registration

The second phase in mobile communication is *registration.* After a mobile host has moved to a foreign network and discovered the foreign agent, it must register. There are four aspects of registration:

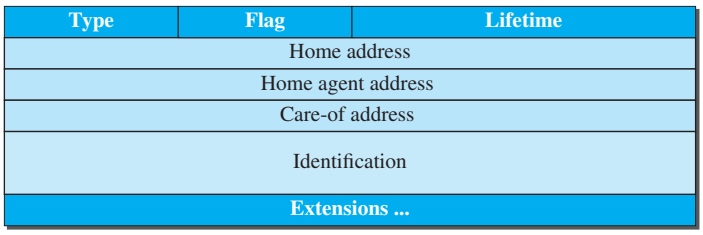
1. The mobile host must register itself with the foreign agent.
2. The mobile host must register itself with its home agent. This is normally done by the foreign agent on behalf of the mobile host.
3. The mobile host must renew registration if it has expired.
4. The mobile host must cancel its registration (deregistration) when it returns home.

Request and Reply

To register with the foreign agent and the home agent, the mobile host uses a *registration request* and a registration reply as shown in Figure 19.14.

Registration Request A registration request is sent from the mobile host to the foreign agent to register its care-of address and also to announce its home address and home agent address. The foreign agent, after receiving and registering the request, relays the message to the home agent. Note that the home agent now knows the address of the foreign agent because the IP packet that is used for relaying has the IP address of the foreign agent as the source address. Figure 19.16 shows the format of the registration request.

Figure 19.16 *Registration request format*



The field descriptions are as follows:

- ***Type.*** The 8-bit type field defines the type of message. For a request message the value of this field is 1.
- ***Flag.*** The 8-bit flag field defines forwarding information. The value of each bit can be set or unset. The meaning of each bit is given in Table 19.2.

Table 19.2 *Registration request flag field bits*

Bit	*Meaning*
0	Mobile host requests that home agent retain its prior care-of address.
1	Mobile host requests that home agent tunnel any broadcast message.
2	Mobile host is using collocated care-of address.
3	Mobile host requests that home agent use minimal encapsulation.
4	Mobile host requests generic routing encapsulation (GRE).
5	Mobile host requests header compression.
6–7	Reserved bits.

- ***Lifetime.*** This field defines the number of seconds the registration is valid. If the field is a string of 0s, the request message is asking for deregistration. If the field is a string of 1s, the lifetime is infinite.
- ***Home address.*** This field contains the permanent (first) address of the mobile host.
- ***Home agent address.*** This field contains the address of the home agent.
- ***Care-of address.*** This field is the temporary (second) address of the mobile host.
- ***Identification.*** This field contains a 64-bit number that is inserted into the request by the mobile host and repeated in the reply message. It matches a request with a reply.
- ***Extensions.*** Variable length extensions are used for authentication. They allow a home agent to authenticate the mobile agent. We discuss authentication in Chapter 31.

Registration Reply A registration reply is sent from the home agent to the foreign agent and then relayed to the mobile host. The reply confirms or denies the registration request. Figure 19.17 shows the format of the registration reply.

The fields are similar to those of the registration request with the following exceptions. The value of the type field is 3. The code field replaces the flag field and shows the result of the registration request (acceptance or denial). The care-of address field is not needed.

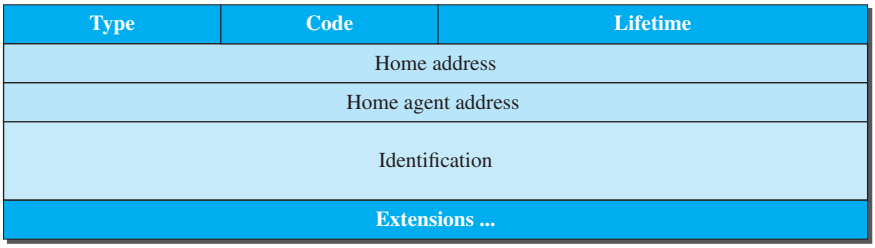
Encapsulation

Registration messages are encapsulated in a UDP user datagram. An agent uses the well-known port 434; a mobile host uses an ephemeral port.

Data Transfer

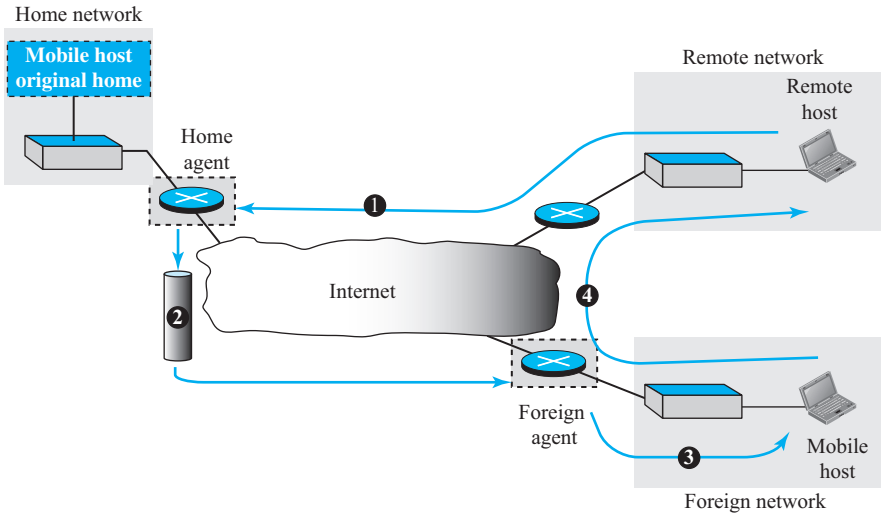
After agent discovery and registration, a mobile host can communicate with a remote host. Figure 19.18 shows the idea.

Figure 19.17 *Registration reply format*



A registration request or reply is sent by UDP using the well-known port 434.

Figure 19.18 *Data transfer*



From Remote Host to Home Agent

When a remote host wants to send a packet to the mobile host, it uses its address as the source address and the home address of the mobile host as the destination address. In other words, the remote host sends a packet as though the mobile host is at its home network. The packet, however, is intercepted by the home agent, which pretends it is the mobile host. This is done using the proxy ARP technique discussed in Chapter 9. Path 1 of Figure 19.18 shows this step.

From Home Agent to Foreign Agent

After receiving the packet, the home agent sends the packet to the foreign agent, using the tunneling concept discuss in Chapter 21. The home agent encapsulates the whole IP packet inside another IP packet using its address as the source and the foreign agent's address as the destination. Path 2 of Figure 19.18 shows this step.

From Foreign Agent to Mobile Host

When the foreign agent receives the packet, it removes the original packet. However, since the destination address is the home address of the mobile host, the foreign agent consults a registry table to find the care-of address of the mobile host. (Otherwise, the packet would just be sent back to the home network.) The packet is then sent to the care-of address. Path 3 of Figure 19.18 shows this step.

From Mobile Host to Remote Host

When a mobile host wants to send a packet to a remote host (for example, a response to the packet it has received), it sends as it does normally. The mobile host prepares a packet with its home address as the source, and the address of the remote host as the destination. Although the packet comes from the foreign network, it has the home address of the mobile host. Path 4 of Figure 19.18 shows this step.

Transparency

In this data transfer process, the remote host is unaware of any movement by the mobile host. The remote host sends packets using the home address of the mobile host as the destination address; it receives packets that have the home address of the mobile host as the source address. The movement is totally transparent. The rest of the Internet is not aware of the movement of the mobile host.

The movement of the mobile host is transparent to the rest of the Internet.

19.3.4 Inefficiency in Mobile IP

Communication involving mobile IP can be inefficient. The inefficiency can be severe or moderate. The severe case is called *double crossing* or *2X*. The moderate case is called *triangle routing* or *dog-leg routing*.

Double Crossing

Double crossing occurs when a remote host communicates with a mobile host that has moved to the same network (or site) as the remote host (see Figure 19.19).

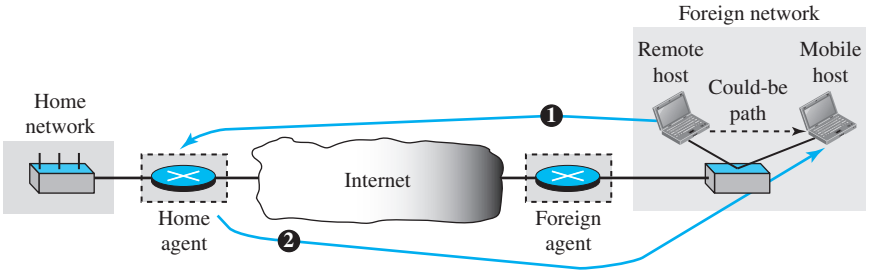
When the mobile host sends a packet to the remote host, there is no inefficiency; the communication is local. However, when the remote host sends a packet to the mobile host, the packet crosses the Internet twice.

Since a computer usually communicates with other local computers (principle of locality), the inefficiency from double crossing is significant.

Triangle Routing

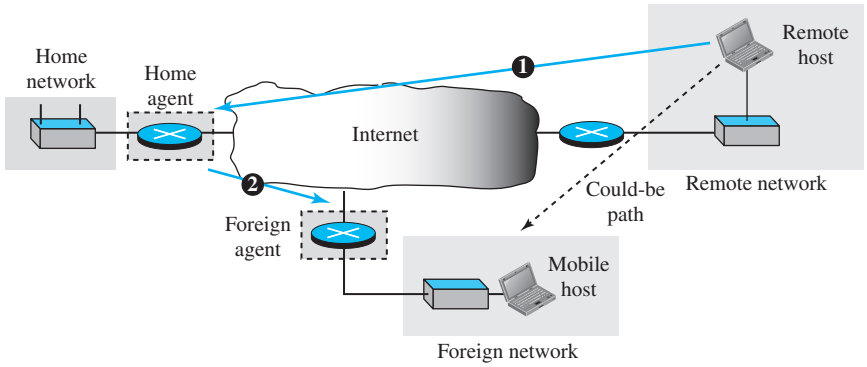
Triangle routing, the less severe case, occurs when the remote host communicates with a mobile host that is not attached to the same network (or site) as the mobile host. When the mobile host sends a packet to the remote host, there is no inefficiency.

Figure 19.19 *Double crossing*



However, when the remote host sends a packet to the mobile host, the packet goes from the remote host to the home agent and then to the mobile host. The packet travels the two sides of a triangle, instead of just one side (see Figure 19.20).

Figure 19.20 *Triangle routing*



Solution

One solution to inefficiency is for the remote host to bind the care-of address to the home address of a mobile host. For example, when a home agent receives the first packet for a mobile host, it forwards the packet to the foreign agent; it could also send an *update binding packet* to the remote host so that future packets to this host could be sent to the care-of address. The remote host can keep this information in a cache.

The problem with this strategy is that the cache entry becomes outdated once the mobile host moves. In this case the home agent needs to send a *warning packet* to the remote host to inform it of the change.

19.4 END-CHAPTER MATERIALS

19.4.1 Recommended Reading

Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], [Koz 05], [Ste 95], [GW 04], [Per 00], [Kes 02], [Moy 98], [WZ 01], and [Los 04].

RFCs

IPv4 protocol is discussed in RFCs 791, 815, 894, 1122, 2474, and 2475. ICMP is discussed in RFCs 792, 950, 956, 957, 1016, 1122, 1256, 1305, and 1987.

19.4.2 Key Terms

care-of address
collocated care-of address
double crossing
foreign agent
foreign network
fragmentation
home address
home agent
home network
Internet Control Message Protocol version 4 (ICMPv4)
mobile host
stationary host
triangle routing

19.4.3 Summary

IPv4 is an unreliable connectionless protocol responsible for source-to-destination delivery. Packets in the IP layer are called datagrams. An IPv4 datagram is made of a header, of size 20 to 60 bytes, and a payload. The total size of an IPv4 datagram can be up to 65,535 bytes. An IPv4 datagram can be fragmented, one or more times, during its path from the source to the destination; reassembly of the fragments, however, should be done at the destination. The checksum for a datagram is calculated only for the header.

The Internet Control Message Protocol version 4 (ICMPv4) supports the unreliable and connectionless Internet Protocol (IP). ICMPv4 messages are encapsulated in IP datagrams. There are two categories of ICMPv4 messages: error-reporting and query messages. The error-reporting messages report problems that a router or a host (destination) may encounter when it processes an IP packet. The query messages, which occur in pairs, help a host or a network manager get specific information from a router or another host.

Mobile IP, designed for mobile communication, is an enhanced version of the Internet Protocol (IP). A mobile host has a home address on its home network and a care-of address on its foreign network. When the mobile host is on a foreign network, a home agent relays messages (for the mobile host) to a foreign agent. A foreign agent sends relayed messages to a mobile host.

20.1 INTRODUCTION

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

20.1.1 General Idea

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

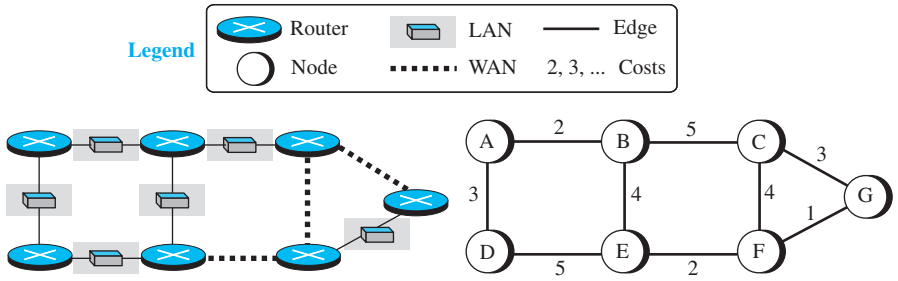
An Internet as a Graph

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

20.1.2 Least-Cost Routing

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

Figure 20.1 *An internet and its graphical representation*

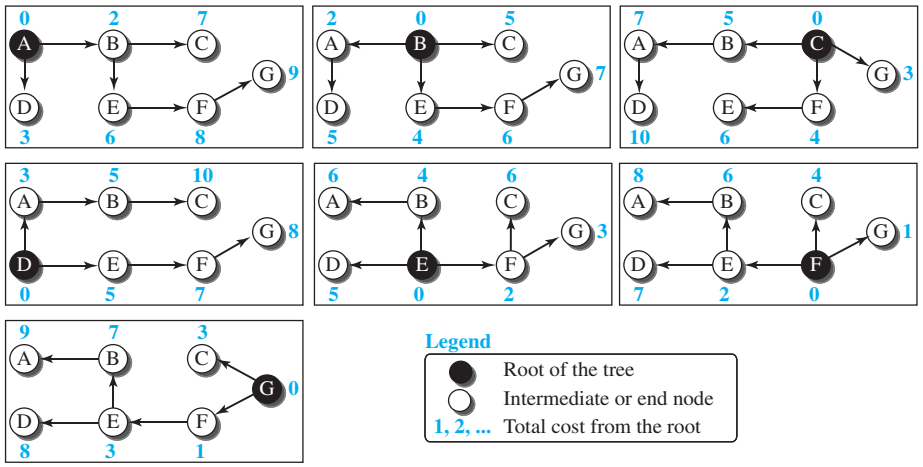


a. An internet b. The weighted graph

Least-Cost Trees

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

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



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

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

20.2 ROUTING ALGORITHMS

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

20.2.1 Distance-Vector Routing

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

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

Bellman-Ford Equation

The heart of distance-vector routing is the famous **Bellman-Ford** equation. This equation is used to find the least cost (shortest distance) between a source node, x, and a destination node, y, through some intermediary nodes (**a, b, c,** . . .) 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.

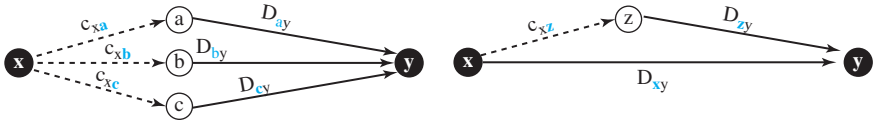
$$D_{xy} = \min\{(c_{xa} + D_{ay}), (c_{xb} + D_{by}), (c_{xc} + D_{cy}), \ldots\}$$

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, as shown below:

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

Figure 20.3 shows the idea graphically for both cases.

Figure 20.3 *Graphical idea behind Bellman-Ford equation*



a. General case with three intermediate nodes

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

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

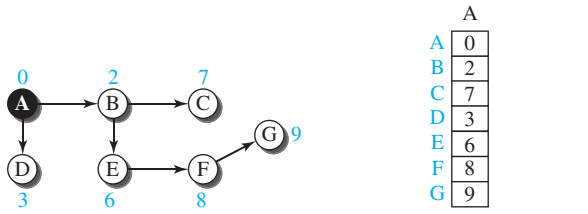
Distance Vectors

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

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

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

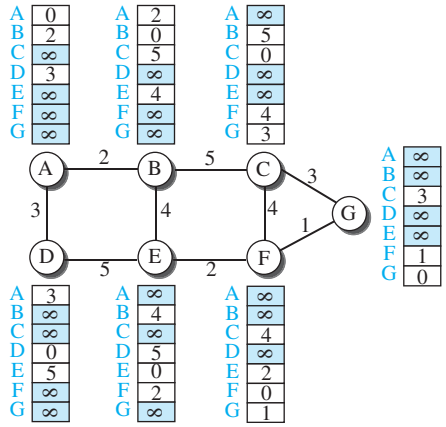
Figure 20.4 *The distance vector corresponding to a tree*



a. Tree for node A b. Distance vector for node A

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

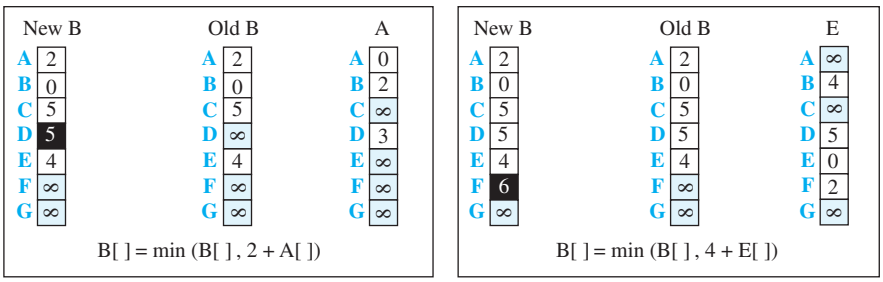
Figure 20.5 *The first distance vector for an internet*



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

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

Figure 20.6 *Updating distance vectors*



a. First event: B receives a copy of A's vector.

b. Second event: B receives a copy of E's vector.

Note:
X[ ]: the whole vector

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

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

Distance-Vector Routing Algorithm

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

Table 20.1 *Distance-Vector Routing Algorithm for a Node*

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

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

```
5     for (y = 1 to N)
6     {
7         if (y is a neighbor)
8             D[y] = c[myself][y]
9         else
10            D[y] = ∞
11    }
12    send vector {D[1], D[2], …, D[N]} to all neighbors
13    // Update (improve the vector with the vector received from a neighbor)
14    repeat (forever)
15    {
16        wait (for a vector 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  } // End of Distance Vector
```

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

Count to Infinity

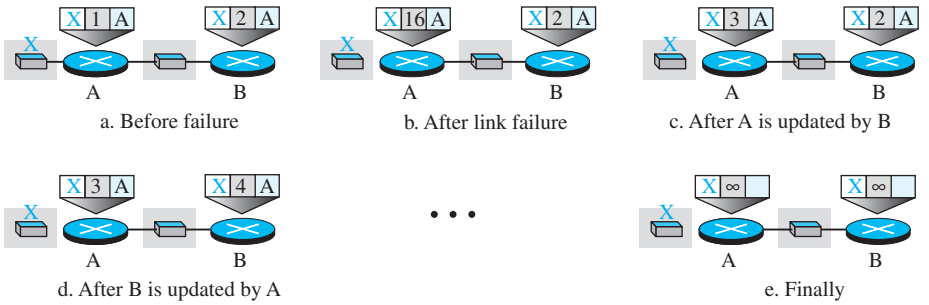
A problem with distance-vector routing is that any decrease in cost (good news) propagates quickly, but any increase in cost (bad news) will propagate slowly. For a routing protocol to work properly, if a link is broken (cost becomes infinity), every other router should be aware of it immediately, but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

Two-Node Loop

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

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

Figure 20.7 *Two-node instability*



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

Split Horizon

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

Poison Reverse

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

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

Three-Node Instability

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

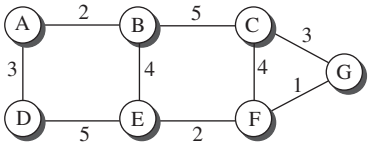
20.2.2 Link-State Routing

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

Link-State Database (LSDB)

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

Figure 20.8 *Example of a link-state database*



a. The weighted graph

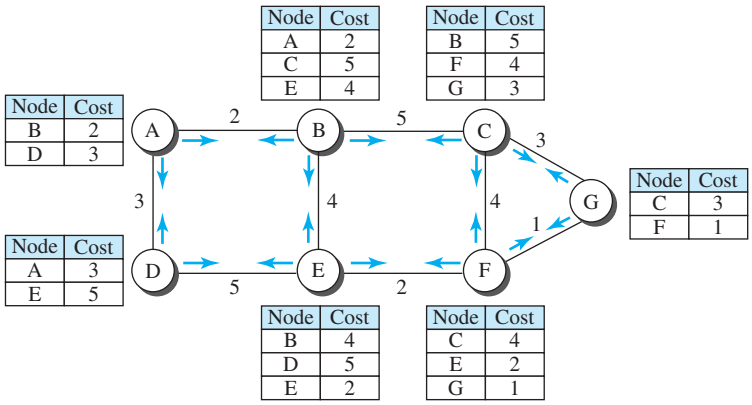
	A	B	C	D	E	F	G
A	0	2	∞	3	∞	∞	∞
B	2	0	5	∞	4	∞	∞
C	∞	5	0	∞	∞	4	3
D	3	∞	∞	0	5	∞	∞
E	∞	4	∞	5	0	2	∞
F	∞	∞	4	∞	2	0	1
G	∞	∞	3	∞	∞	1	0

b. Link state database

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

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

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



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

Formation of Least-Cost Trees

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

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

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

Table 20.2 *Dijkstra's Algorithm*

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

Table 20.2 *Dijkstra's Algorithm (continued)*

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

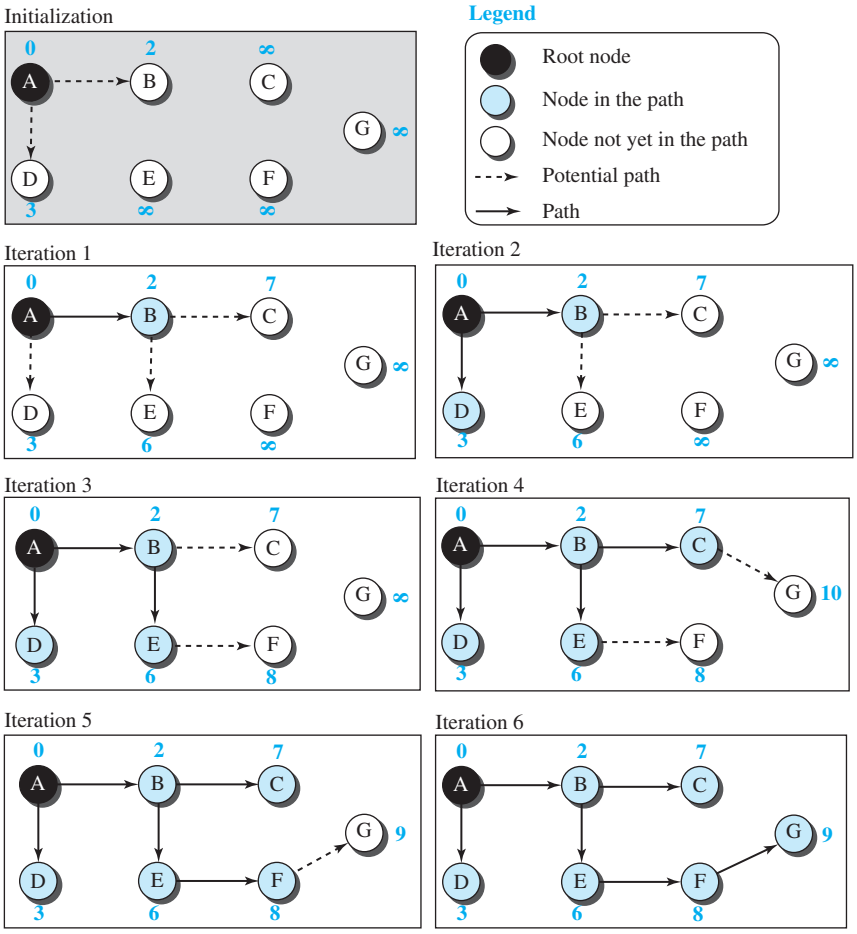
Lines 4 to 13 implement step 1 in the algorithm. Lines 16 to 23 implement step 2 in the algorithm. Step 2 is repeated until all nodes are added to the tree.

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

20.2.3 Path-Vector Routing

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

Figure 20.10 *Least-cost tree*



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

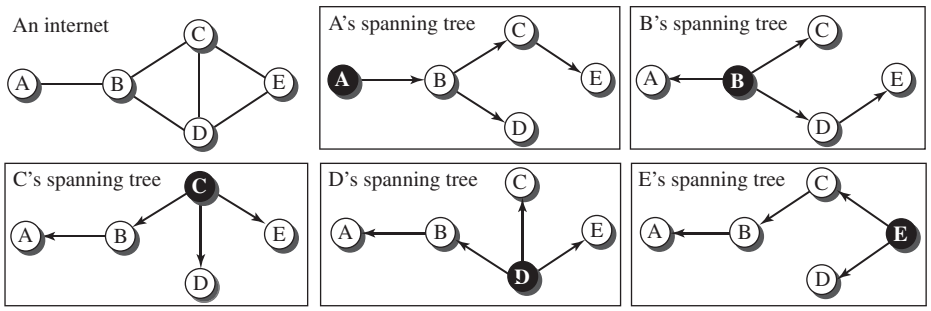
Spanning Trees

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

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

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

Figure 20.11 *Spanning trees in path-vector routing*



Creation of Spanning Trees

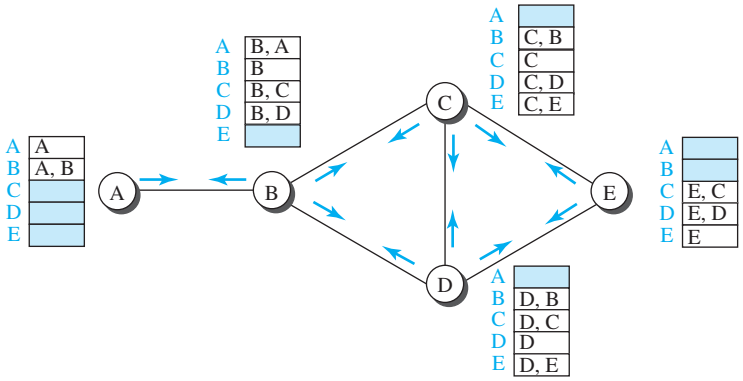
Path-vector routing, like distance-vector routing, is an asynchronous and distributed routing algorithm. The spanning trees are made, gradually and asynchronously, by each node. When a node is booted, it creates a *path vector* based on the information it can obtain about its immediate neighbor. A node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

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

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

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

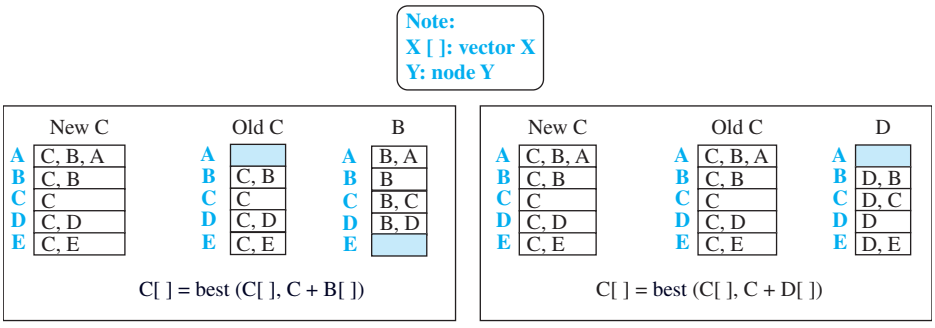
Figure 20.12 *Path vectors made at booting time*



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

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

Figure 20.13 *Updating path vectors*



Event 1: C receives a copy of B's vector

Event 2: C receives a copy of D's vector

Path-Vector Algorithm

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

Table 20.3 *Path-vector algorithm for a node*

```
1  Path_Vector_Routing ( )
2  {
3     // Initialization
4     for (y = 1 to N)
5     {
6        if (y is myself)
7           Path[y] = myself
8        else if (y is a neighbor)
9           Path[y] = myself + neighbor node
10       else
11          Path[y] = empty
12    }
13    Send vector {Path[1], Path[2], …, Path[y]} to all neighbors
14    // Update
15    repeat (forever)
16    {
17       wait (for a vector 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.

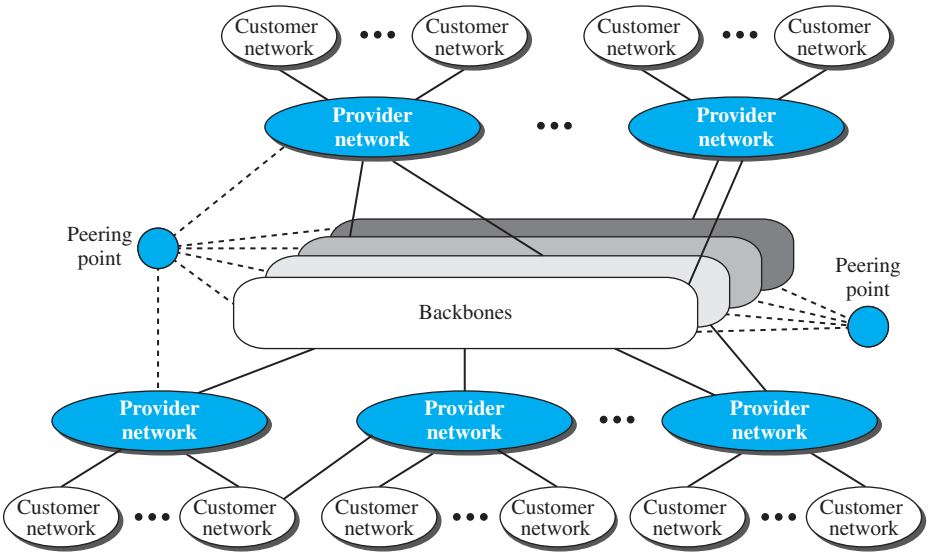
20.3 UNICAST ROUTING PROTOCOLS

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

20.3.1 Internet Structure

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

Figure 20.14 *Internet structure*



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

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

Hierarchical Routing

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

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

Autonomous Systems

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

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

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

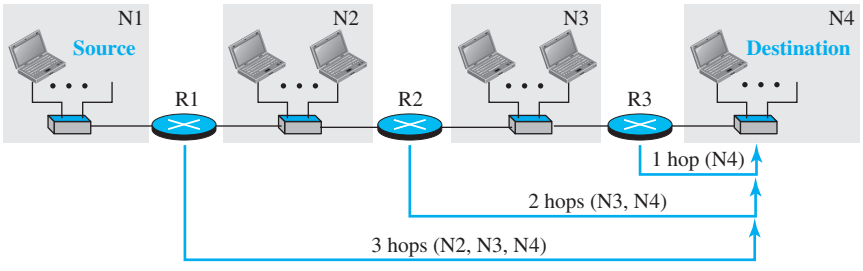
20.3.2 Routing Information Protocol (RIP)

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

Hop Count

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

Figure 20.15 *Hop counts in RIP*



Forwarding Tables

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

Figure 20.16 *Forwarding tables*

Forwarding table for R1

Destination network	Next router	Cost in hops
N1	——	1
N2	——	1
N3	R2	2
N4	R2	3

Forwarding table for R2

Destination network	Next router	Cost in hops
N1	R1	2
N2	——	1
N3	——	1
N4	R3	2

Forwarding table for R3

Destination network	Next router	Cost in hops
N1	R2	3
N2	R2	2
N3	——	1
N4	——	1

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

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

RIP Implementation

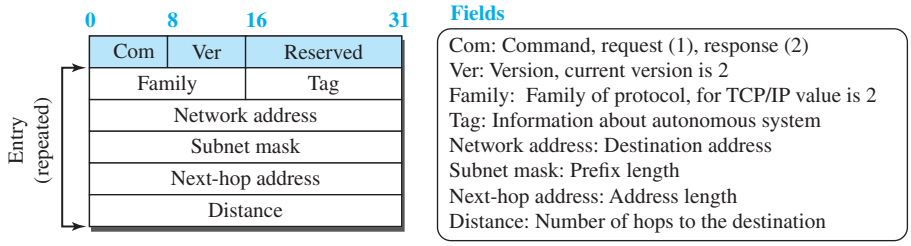
RIP is implemented as a process that uses the service of UDP on the well-known port number 520. In BSD, RIP is a daemon process (a process running in the background), named *routed* (abbreviation for *route daemon* and pronounced *route-dee)*. This means that, although RIP is a routing protocol to help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

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

RIP Messages

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

Figure 20.17 *RIP message format*



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

RIP Algorithm

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

- Instead of sending only distance vectors, a router needs to send the whole contents of its forwarding table in a response message.
- The receiver adds one hop to each cost and changes the next router field to the address of the sending router. We call each route in the modified forwarding table the *received route* and each route in the old forwarding table the *old route*. The received router selects the old routes as the new ones except in the following three cases:

1. If the received route does not exist in the old forwarding table, it should be added to the route.
2. If the cost of the received route is lower than the cost of the old one, the received route should be selected as the new one.
3. If the cost of the received route is higher than the cost of the old one, but the value of the next router is the same in both routes, the received route should be selected as the new one. This is the case where the route was actually advertised

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

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

Example 20.1

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

Timers in RIP

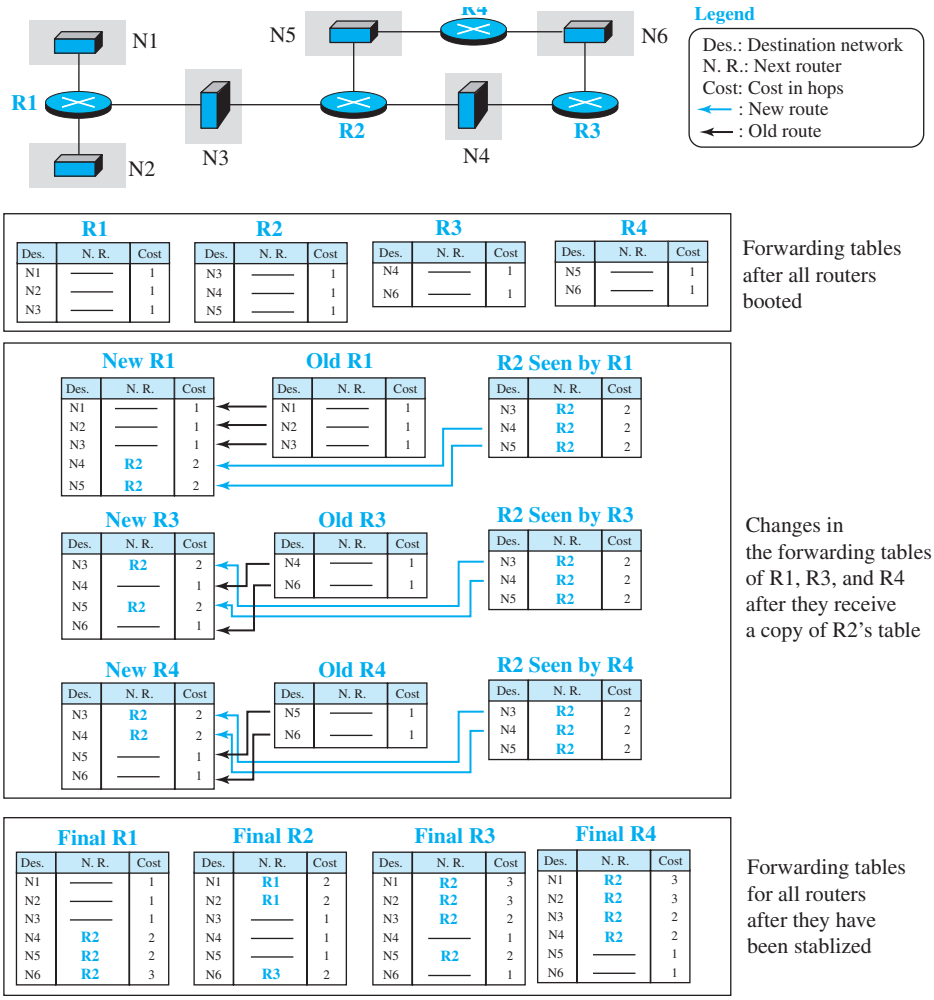
RIP uses three timers to support its operation. The *periodic timer* controls the advertising of regular update messages. Each router has one periodic timer that is randomly set to a number between 25 and 35 seconds (to prevent all routers sending their messages at the same time and creating excess traffic).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

Performance

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

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

Figure 20.18 *Example of an autonomous system using RIP*



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

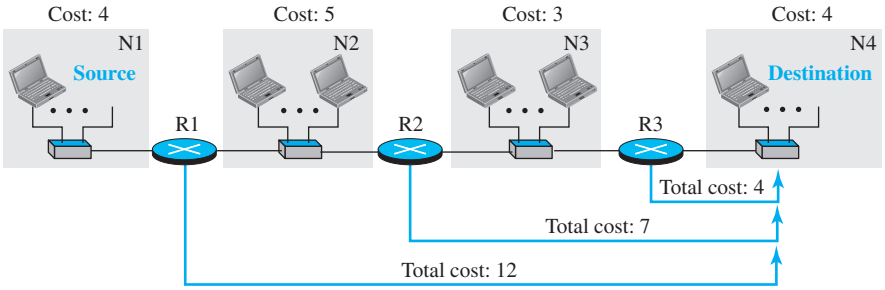
20.3.3 Open Shortest Path First (OSPF)

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

Metric

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

Figure 20.19 *Metric in OSPF*



Forwarding Tables

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

Areas

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

Figure 20.20 *Forwarding tables in OSPF*

Forwarding table for R1

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

Forwarding table for R2

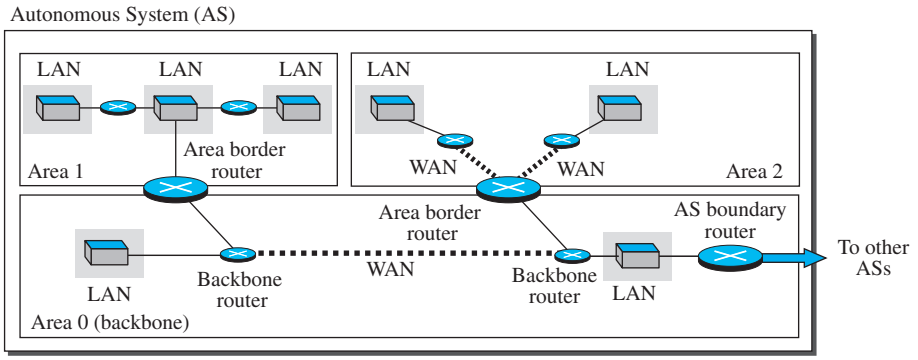
Destination network	Next router	Cost
N1	R1	9
N2	——	5
N3	——	3
N4	R3	7

Forwarding table for R3

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

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

Figure 20.21 *Areas in an autonomous system*

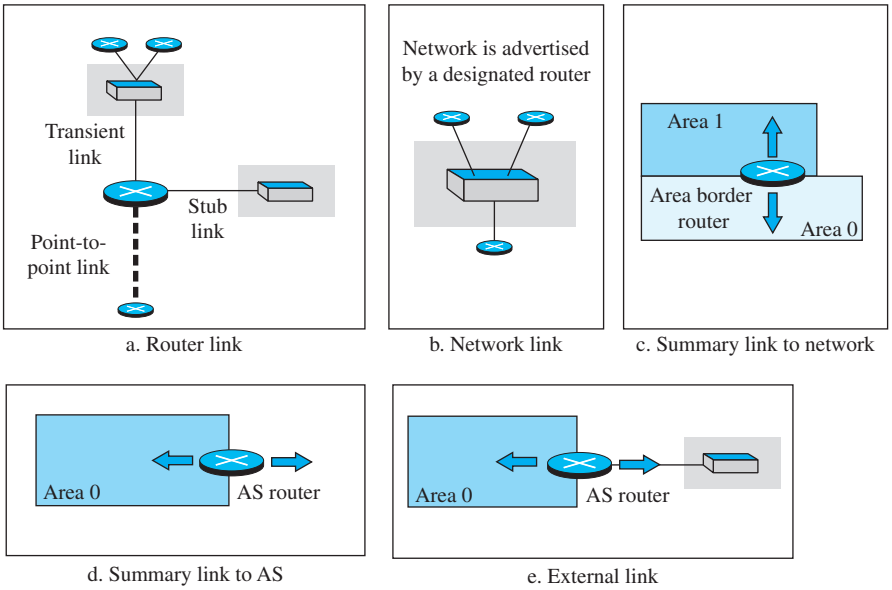


Link-State Advertisement

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

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

Figure 20.22 *Five different LSPs*



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

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

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

OSPF Implementation

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

OSPF Messages

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

Authentication

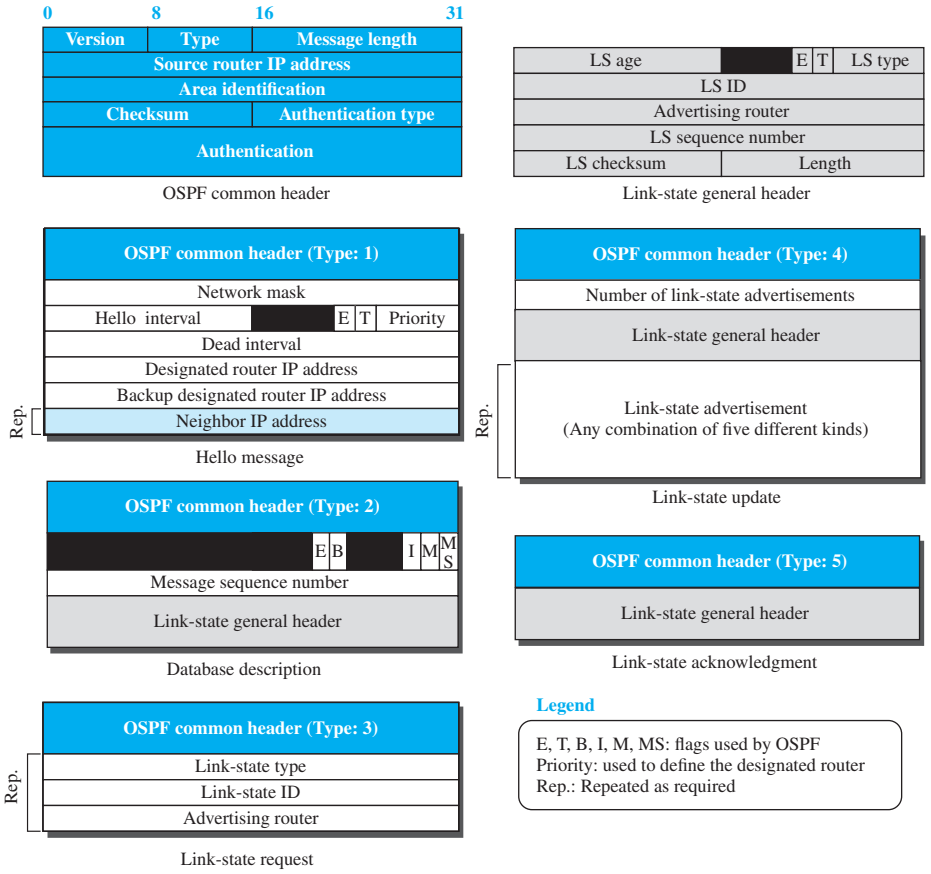
As Figure 20.23 shows, the OSPF common header has the provision for authentication of the message sender. As we will discuss in Chapters 31 and 32, this prevents a malicious entity from sending OSPF messages to a router and causing the router to become part of the routing system to which it actually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

OSPF Algorithm

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

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

Figure 20.23 *OSPF message formats*



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

Performance

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

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

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

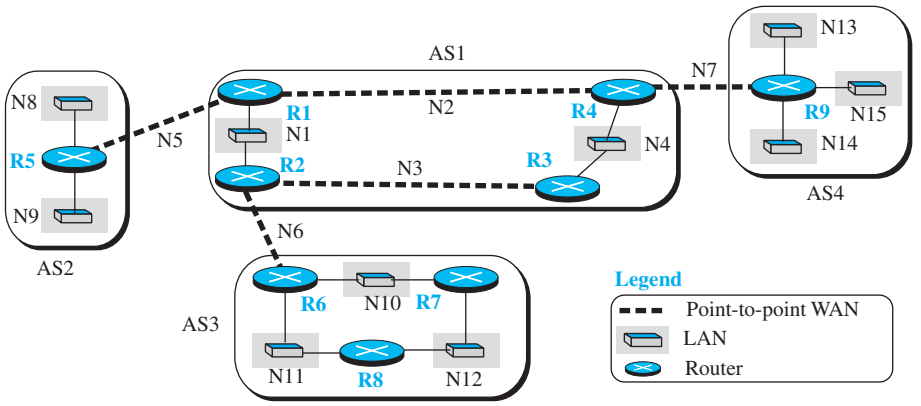
20.3.4 Border Gateway Protocol Version 4 (BGP4)

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

Introduction

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

Figure 20.24 *A sample internet with four ASs*



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

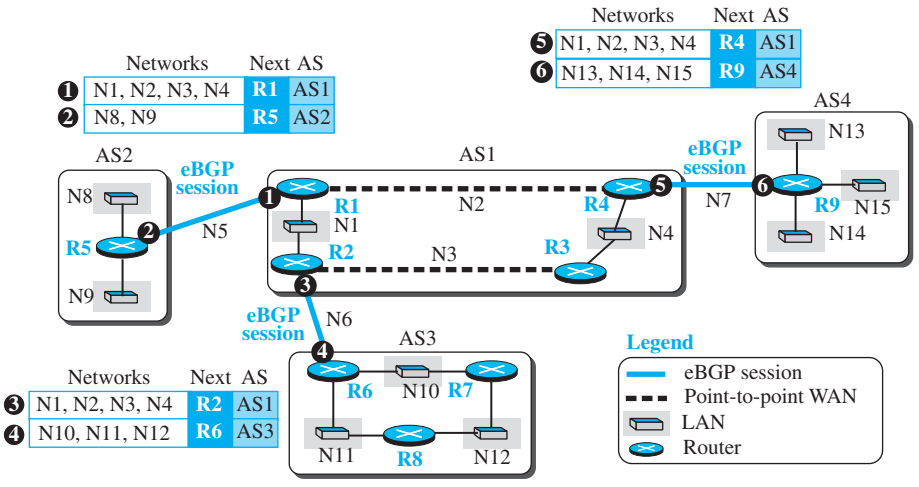
To enable each router to route a packet to any network in the internet, we first install a variation of BGP4, called *external BGP* (*eBGP*), on each *border router* (the one at the edge of each AS which is connected to a router at another AS). We then install the second variation of BGP, called *internal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

Operation of External BGP (eBGP)

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

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

Figure 20.25 *eBGP operation*



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

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

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

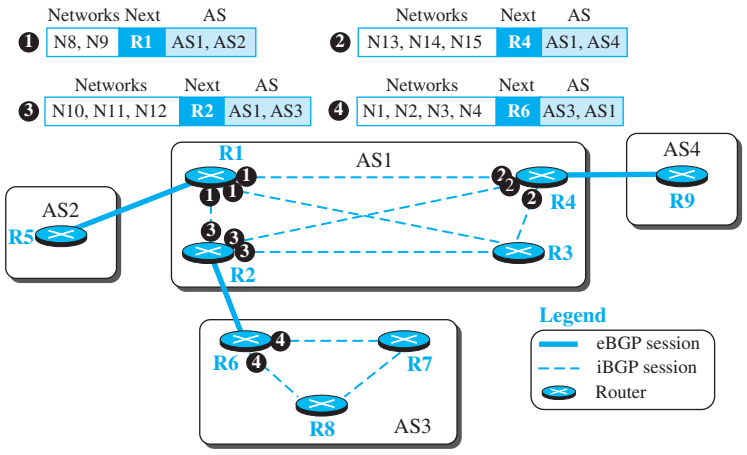
1. Some border routers do not know how to route a packet destined for nonneighbor ASs. For example, R5 does not know how to route packets destined for networks in AS3 and AS4. Routers R6 and R9 are in the same situation as R5: R6 does not know about networks in AS2 and AS4; R9 does not know about networks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

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

Operation of Internal BGP (iBGP)

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

Figure 20.26 *Combination of eBGP and iBGP sessions in our internet*



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

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

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

Figure 20.27 *Finalized BGP path tables*

Path table for R1

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

Path table for R2

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

Path table for R3

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

Path table for R4

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

Path table for R5

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 R6

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 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	R6	AS3, AS1
N8, N9	R6	AS3, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4

Path table for R9

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

Injection of Information into Intradomain Routing

The role of an interdomain routing protocol such as BGP is to help the routers inside the AS to augment their routing information. In other words, the path tables collected and organized by 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 20.24, R5 in AS2 defines R1 as the default router for

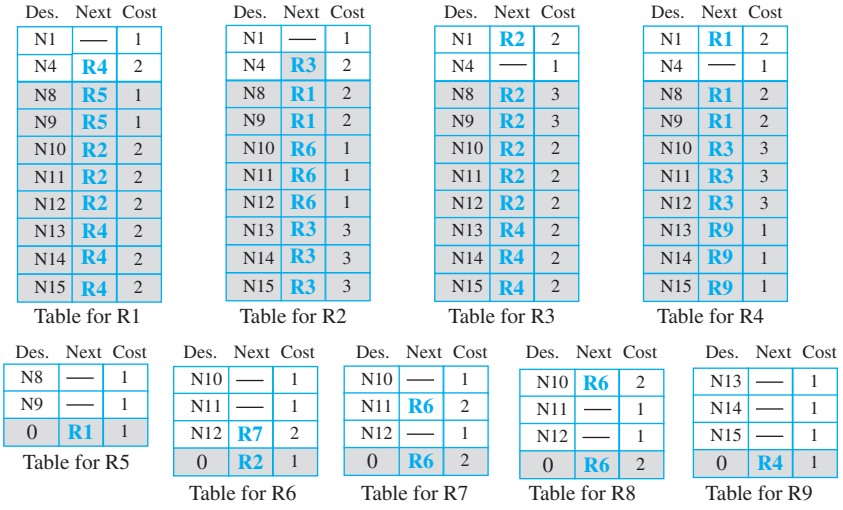
all networks other than N8 and N9. The situation is the same for router R9 in AS4 with the default router to be R4. In AS3, R6 set its default router to be R2, but R7 and R8 set their default router to be R6. These settings are in accordance with the path tables we describe in Figure 20.27 for these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

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

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

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

Figure 20.28 *Forwarding tables after injection from BGP*



Address Aggregation

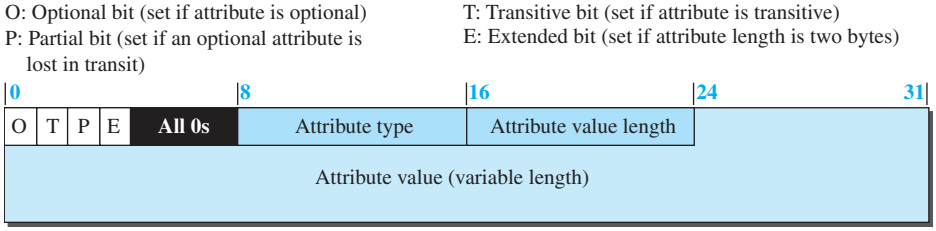
The reader may have realized that intradomain forwarding tables obtained with the help of the BGP4 protocols may become huge in the case of the global Internet because many destination networks may be included in a forwarding table. Fortunately, BGP4 uses the prefixes as destination identifiers and allows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

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

Path Attributes

In both intradomain routing protocols (RIP or OSPF), a destination is normally associated with two pieces of information: next hop and cost. The first one shows the address of the next router to deliver the packet; the second defines the cost to the final destination. 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 need not be. A well-known attribute can be mandatory, which means that it must be present in any BGP update message, or discretionary, which means it does not have to be. An optional attribute can be either transitive, which means it can pass to the next AS, or intransitive, which means it cannot. All attributes are inserted after the corresponding destination prefix in an update message (discussed later). The format for an attribute is shown in Figure 20.29.

Figure 20.29 *Format of path attribute*



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

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

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

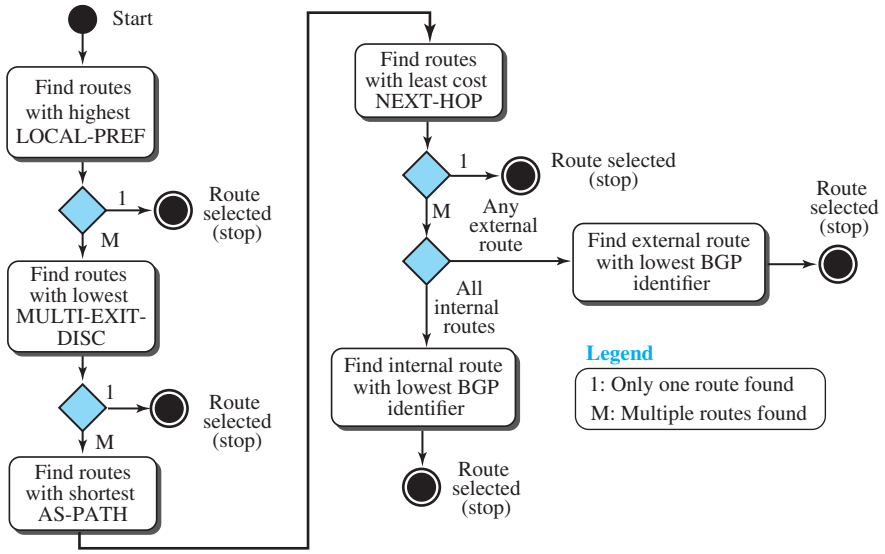
- ***NEXT-HOP (type 3).*** This is a well-known mandatory attribute, which defines the next router to which the data packet should be forwarded. We have also used this attribute in our examples. As we have seen, this attribute helps to inject path information collected through the operations of eBGP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

Route Selection

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

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

Figure 20.30 *Flow diagram for route selection*



Messages

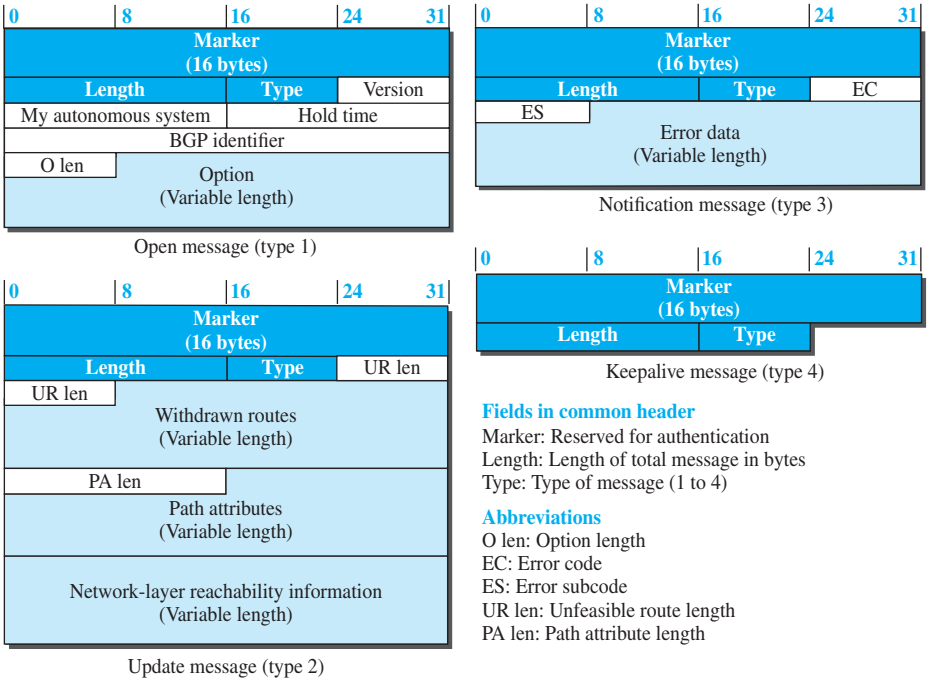
BGP uses four types of messages for communication between the BGP speakers across the ASs and inside an AS: *open, update, keepalive,* and *notification* (see Figure 20.31). All BGP packets share the same common header.

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

Performance

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

Figure 20.31 *BGP messages*



20.4 END-CHAPTER MATERIALS

20.4.1 Recommended Reading

Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], [Koz 05], [Ste 95], [GW 04], [Per 00], [Kes 02], [Moy 98], [WZ 01], and [Los 04].

RFCs

RIP is discussed in RFCs 1058 and 2453. OSPF is discussed in RFCs 1583 and 2328. BGP is discussed in RFCs 1654, 1771, 1773, 1997, 2439, 2918, and 3392.

20.4.2 Key Terms

autonomous system (AS)
Bellman-Ford
Border Gateway Protocol version 4 (BGP4)
Dijkstra's algorithm
distance vector
distance-vector (DV) routing
flooding
least-cost tree