BookName:tcp

transport protocols
Transport layer

more sensitive to delay and jitter than the packets in the first category and hence should be transmitted before them during periods of congestion. Also, within the second category, packets containing compressed video are more sensitive to packet loss than packets containing just audio and hence are given a higher priority.

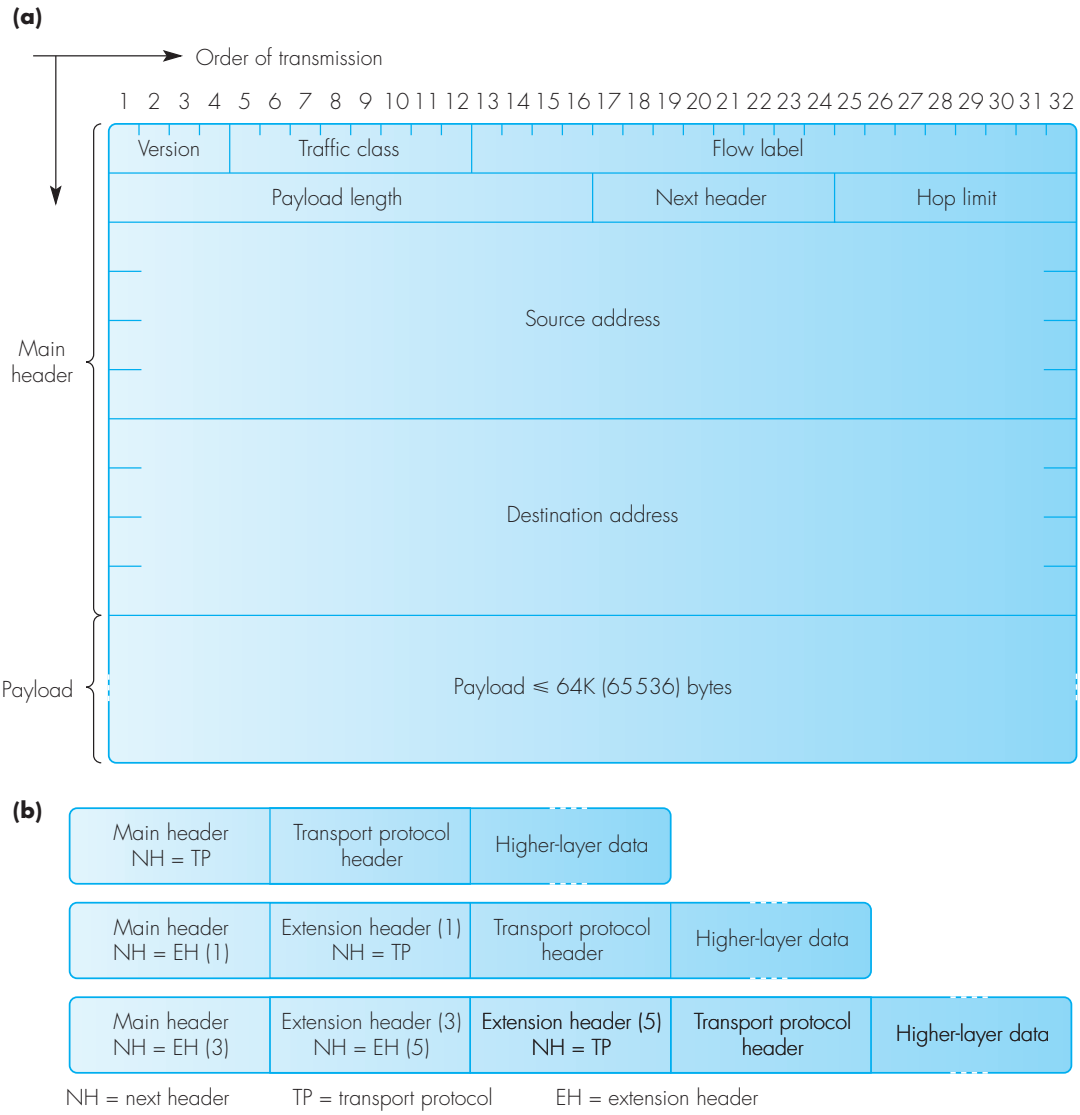


Figure 6.41 IPv6: (a) main header fields and format; (b) position and order of extension headers.

Flow label

This is a new field and is closely linked to the *traffic class* field. It is set to zero in best-effort packets and, in packets in the second category, it is used to enable a router to identify the individual packets relating to the same call/session. As we saw earlier in Section 6.7.1, one approach to handling packets containing real-time streams is to reserve resources – for example transmission bandwidth – for such calls in advance of sending the packets relating to the call. During the reservation procedure, the call is allocated a *flow label* by the source. Also, each router along the reserved path keeps a record of this, together with the source and destination IP addresses, in a table. Routers then use the combined *flow label* and source IP address present in each packet header to relate the packet to a specific call/flow. The related routing information is then retrieved from the routing table and this is used, together with the *traffic class* value, to ensure the QoS requirements of the call/flow are met during the forwarding procedure.

Payload length

This indicates the number of bytes that follow the basic 40-byte header in the datagram. The minimum length of a basic datagram is 536 bytes and the maximum length is 64K bytes. The *payload length* is slightly different from the *total length* field used in the header of an IPv4 datagram since, as we explained in Section 6.2, the *total length* includes the number of bytes in the datagram header.

Next header

As we show in Figure 6.41(b), a basic IPv6 datagram comprises a main header followed by the header of the peer transport layer protocol (TCP/UDP) and, where appropriate, the data relating to the higher layers. With a basic datagram, therefore, the *next header* field indicates the type of transport layer protocol (header) that follows the basic header. If required, however, a number of what are called ***extension headers*** can be inserted between the main header and the transport protocol header. Currently, there are six types of extension header defined and, when present, each extension header starts with a new *next header* field which indicates the type of header that follows. The *next header* field in the last extension header always indicates the type of transport protocol header that follows. Thus, the *next header* field in either the main header or the last extension header plays the same role as the *protocol* field in an IPv4 datagram header.

Hop limit

This is similar to the *time-to-live* parameter in an IPv4 header except the value is a hop count instead of a time. In practice, as we explained in Section 6.2, most IPv4 routers also use this field as a hop count so the change in the field's name simply reflects this. The initial value in the *hop limit* field is set by the source and is decremented by 1 each time the packet/datagram is forwarded. The packet is discarded if the value is decremented to zero.

Source address and destination address

As we indicated earlier, these are 128-bit addresses that are used to identify the source of the datagram and the intended recipient. In most cases this will be the destination host but, as we shall explain later, it might be the next router along a path if source routing is being used. Unlike IPv4 addresses, an IPv6 address is assigned to the (physical) interface, not to the host or router. Hence in the case of routers (which have multiple interfaces) these are identified using any of the assigned interface addresses.

6.8.2 Address structure

As we showed in Figure 6.19, the various networks and internetworks that make up the global Internet are interconnected in a hierarchical way with the access networks at the lowest level in the hierarchy and the global backbone network at the highest level. However, the lack of structure in the netid part of IPv4 addresses means that the number of entries in the routing tables held by each gateway/router increases with increasing height in the hierarchy. At the lowest level, most access gateways associated with a single site LAN have a single netid, while at the highest level, most backbone routers/gateways have a routing table containing many thousands of netids.

In contrast, the addresses associated with telephone networks are hierarchical with, for example, a country, region, and exchange code preceding the local number. This has a significant impact on the size of the routing tables held by the switches since, at a particular level, all calls with the same preceding code are routed in the same way. This is known as **address aggregation**.

As we explained earlier in Section 6.4.3, classless inter-domain routing is a way of introducing a similar structure with IPv4 addresses and reduces considerably the size of the routing tables held by the routers/gateways in the global backbone. From the outset, therefore, IPv6 addresses are hierarchical. Unlike telephone numbers, however, the hierarchy is not constrained just to a geographical breakdown. The large address space available means that a number of alternative formats can be used. For example, to help interworking with existing IPv4 hosts and routers, there is a format that allows IPv4 addresses to be embedded into an IPv6 address. Also, since the majority of access networks are now Internet service provider (ISP) networks, there is a format that allows large blocks of addresses to be allocated to individual providers. The particular format being used is determined by the first set of bits in the address. This is known as the **prefix format** (**PF**) and a list of the prefixes that have been assigned – together with their usage – is given in Figure 6.42(a).

Unicast addresses

As we can see, addresses starting with a prefix of 0000 0000 are used to carry existing IPv4 addresses. There are two types, the formats of which are shown

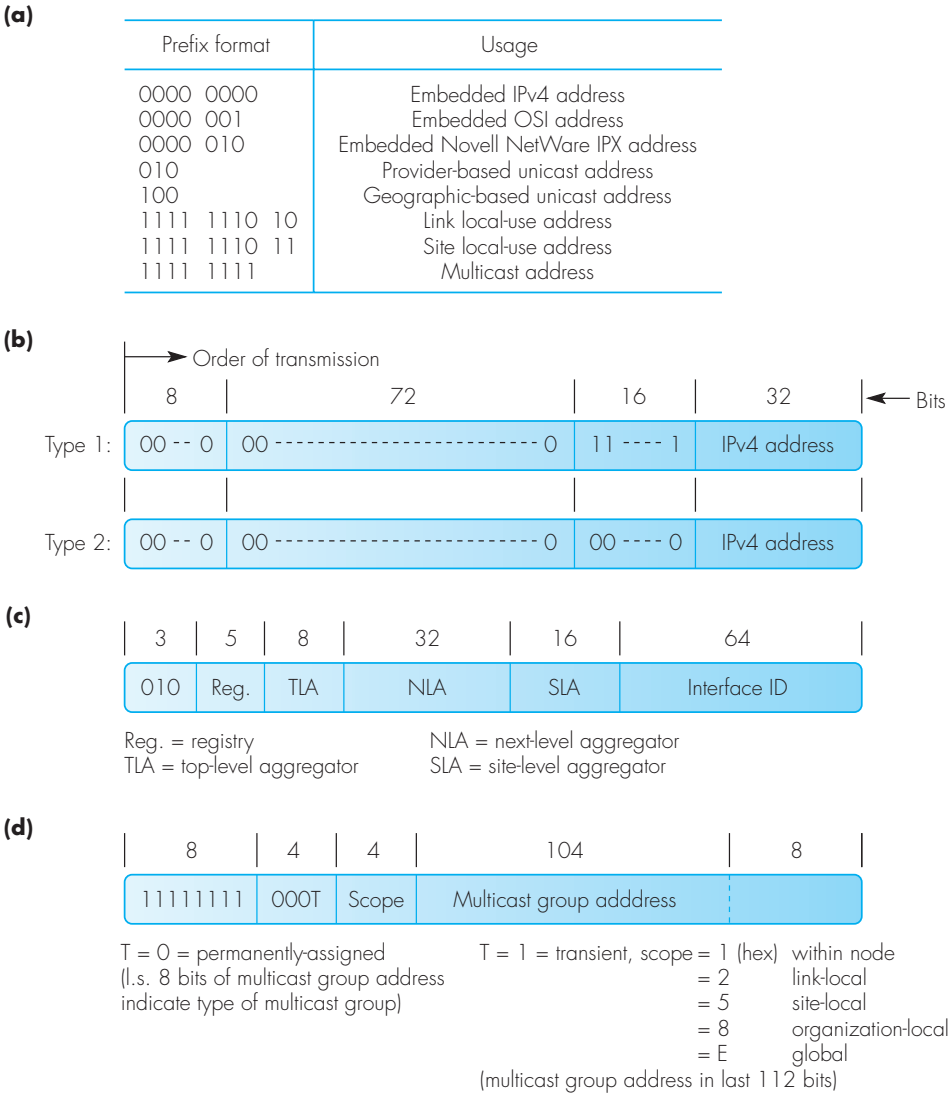


Figure 6.42 IPv6 addresses: (a) prefix formats and their use; (b) IPv4 address types; (c) provider-based unicast address format; (d) multicast address format.

in Figure 6.42(b). As we shall expand upon in Section 6.9, a common requirement during the transition from IPv4 to IPv6 is to tunnel the IPv6 packets being generated by the two communicating IPv6 hosts – often written **V6 hosts** – over an existing IPv4 network/internetwork. Hence to simplify the routing of the IPv4 packet – containing the IPv6 packet within it – the IPv6

address contains the IPv4 address of the destination gateway embedded within it. The second type is to enable a V4 host to communicate with a V6 host. The IPv4 address of the V4 host is then preceded by 96 zeros. In addition, two addresses in this category are reserved for other uses. An address comprising all zeros indicates there is no address present. As we shall expand upon in Section 6.8.4, an example of its use is for the source address in a packet relating to the autoconfiguration procedure. An address with a single binary 1 in the least significant bit position is reserved for the loopback address used during the test procedure of a host protocol stack.

The OSI and NetWare address prefixes have been defined to enable a host connected to one of these networks to communicate directly with a V6 host. The most widely used is the provider-based prefix as this format reflects the current structure of the Internet. A typical format of this type of address is shown in Figure 6.42(c). As we saw in Figure 6.19, the core backbone of the Internet consists of a number of very high bandwidth lines that interconnect the various continental backbones together. The routers that perform this function are owned by companies known as **top-level aggregators** (**TLAs**). Each TLA is allocated a large block of addresses by what is called a **registry**, the identity of which is in the field immediately following the 010 prefix. Examples include the North American registry, the European registry, the Asia and Pacific registry, and so on.

From their allocation, the TLAs allocate blocks of addresses to large Internet service providers and global enterprises. These are known as **next-level aggregators** (**NLAs**) and, in the context of Figure 6.19, operate at the continental backbone and national and regional levels. The various NLAs allocate both single addresses to individual subscribers and blocks of addresses to large business customers. The latter are known as **site-level aggregators** (**SLAs**) and include ISPs that operate at the regional and national levels. The 64-bit **interface ID** is divided locally into a fixed subnetid part and a hostid part. Typically, the latter is the 48-bit MAC address of the host and hence 16 bits are available for subnetting.

As we can deduce from Figure 6.42(c), the use of hierarchical addresses means that each router in the hierarchy can quickly determine whether a packet should be routed to a higher-level router or to another router at the same level simply by examining the appropriate prefix. Also, the routers at each level can route packets directly using the related prefix. The same overall description applies to the processing of geographic-based addresses.

As their names imply, the two types of **local-use addresses** are for local use only and have no meaning in the context of the global Internet. As we shall expand upon in Section 6.8.4, *link local-use addresses* are used in the autoconfiguration procedure followed by hosts to obtain an IPv6 address from a local router. The router only replies to the host on the same link the request was received and hence this type of packet is not forwarded beyond the router.

The *site local-use addresses* are used, for example, by organizations that are not currently connected to the Internet but wish to utilize the technology

associated with it. Normally, the 64-bit interface ID part is subdivided and used for routing purposes within the organization. In this way, should the organization wish to be connected to the Internet at a later date, it is only necessary to change the site local-use prefix with the allocated subscriber prefix.

Multicast addresses

The format of an IPv6 multicast address is shown in Figure 6.42(d). As we can see, following the multicast prefix are two additional fields that have been introduced to limit the geographic scope of the related multicast operation. The first is known as the *flags* field and is used to indicate whether the multicast is a permanently-assigned (reserved) address (0000) or a temporary (transient) address (0001). In the case of a permanently-assigned address, the least significant 8 bits of the *multicast group address* field identify the type of the multicast operation, while for a transient address, the full 112 bits identify the multicast group address.

In both cases, the 4-bit *scope* field defines the geographic scope of the multicast packet. The various alternatives are identified in the figure and mrouters use this field to determine whether the (multicast) packet should be forwarded further or discarded.

Anycast addresses

In addition to unicast and multicast addresses, a new address type known as an ***anycast group address*** has been defined. These are allocated from the unicast address space and are indistinguishable from a unicast address. With an anycast address, however, a group of hosts or routers can all have the same (anycast) address. A common requirement in a number of applications is for a host or router to send a packet to any one of a group of hosts or routers, all of which provide the same service. For example, a group of servers distributed around a network may all contain the same database. Hence in order to avoid all clients needing to know the unique address of its nearest server, all the servers can be members of the same anycast group and hence have the same address. In this way, when a client makes a request, it uses the assigned anycast address of the group and the request will automatically be received by its nearest server. Similarly, if a single network/internetwork has a number of gateway routers associated with it, they can all be allocated the same anycast address. As a result, the shortest-path routes from all other networks/internetworks will automatically use the gateway nearest to them.

In order to perform the routing function, although an anycast address has the same format as a unicast address, when an anycast address is assigned to a group of hosts or routers, it is necessary for each host/router to be explicitly informed – by network management for example – that it is a member of an anycast group. In addition, each is informed of the common part of the address prefixes which collectively identify the topological region in which all the hosts/routers reside. Within this region, all the routers then maintain a separate entry for each member of the group in its routing table.

Address representation

A different form of representation of IPv6 addresses has been defined. Instead of each 8-bit group being represented as a decimal number (with a dot/period between them), groups of 16 bits are used. Each 16-bit group is then represented in its hexadecimal form with a colon between each group. An example of an IPv6 address is:

FEDC:BA98:7654:3210:0000:0000:0000:0089

In addition, a number of abbreviations have been defined:

- One or more consecutive groups of all zeros can be replaced by a pair of colons.
- Leading zeros in a group can be omitted.

Hence the preceding address can also be written as:

FEDC:BA98:7654:3210::89

Also, for the two IPv4 embedded address types, the actual IPv4 address can remain in its dotted decimal form. Hence assuming a dotted decimal address of 15.10.0.6, the two embedded forms are:

:: 150.10.0.6 (IPv4 host address)
:: FFFF:150.10.0.6 (IPv4 tunnel address)

Example 6.6

Derive the hexadecimal form of representation of the following link-local multicast addresses:

(i) a permanently-assigned multicast group address of 67,

(ii) a transient multicast group address of 317.

Answer:

The formats of the two types of multicast addresses were shown in Figure 6.42(d). Hence:

The most significant 16 bits are FF02 = permanently-assigned, link-local
and FF12 = transient, link-local

(i) A permanently-assigned multicast group address of 67 = 0043 (hex)
Hence IPv6 address = FF02 :: 67

(ii) A transient multicast group address of 317 = 013D (hex)
Hence IPv6 address = FF12 :: 13D

6.8.3 Extension headers

As we have just indicated, if required, a number of extension headers can be added to the main header to convey additional information, either to the routers visited along the path followed or to the destination host. The six types of extension header currently defined are:

- **hop-by-hop options**: information for the routers visited along a path;
- **routing**: list of routers relating to source routing information;
- **fragment**: information to enable the destination to reassemble a fragmented message;
- **authentication**: information to enable the destination to verify the identity of the source;
- **encapsulating security payload**: information to enable the destination to decrypt the payload contents;
- **destination options**: optional information for use by the destination.

The two options headers can contain a variable number of option fields, each possibly of a variable length. For these, each option field is encoded using a **type-length-value** (**TLV**) format. The *type* and *length* are both single bytes and indicate the option type and its length (in bytes) respectively. The *value* is then found in the following number of bytes indicated by the *length*. The option type identifiers have been chosen so that the most significant two bits specify the action that must be taken if the type is not recognized. These are:

00 ignore this option and continue processing the other option fields in the header;

01 discard the complete packet;

10 discard the packet and return an ICMP error report to the source indicating a parameter problem and the option type not recognized;

11 same as for 10 except the ICMP report is only returned if the destination address is not a multicast address.

Note also that since the type of extension header is indicated in the preceding *next header* field, the related decoder that has been written to decode the contents of the header is invoked automatically as each header is processed. Some examples of each header type now follow.

Hop-by-hop options

This type of header contains information that must be examined by all the gateways and routers the packet visits along its route. The *next header* value for this is 0 and, if this header is present, it must follow the main header. Hence the *next header* field in the main header is set to 0. An example of its use is for a host to send a datagram that contains a payload of more than 64K bytes. This is

particularly useful, for example, when a host is transferring many very large files over a path that supports a maximum transmission unit (MTU) significantly greater than 64 kbytes. Datagrams that contain this header are known as **jumbograms** and the format of the header is shown in Figure 6.43(a).

As we can see, this type of header is of fixed length and comprises two 32-bit words (8 bytes). The *header extension length* field indicates the length of the header in multiples of 8 bytes, excluding the first 8 bytes. Hence in this case the field is 0. This option contains only one option field, the *jumbo payload length*. As we indicated earlier, this is encoded in the TLV format. The *type* for this option is 194 (11000010) and the *length* is 4 (bytes). The *value* in the *jumbo payload length* is the length of the packet in bytes, excluding the main header but including the 8 bytes in the extension header. This makes the *payload length* in the main header redundant and hence this is set to zero.

Routing

This plays a similar role to the (strict) *source routing* and *loose source routing* optional headers used in IPv4 datagrams. The *next header* value for this type of

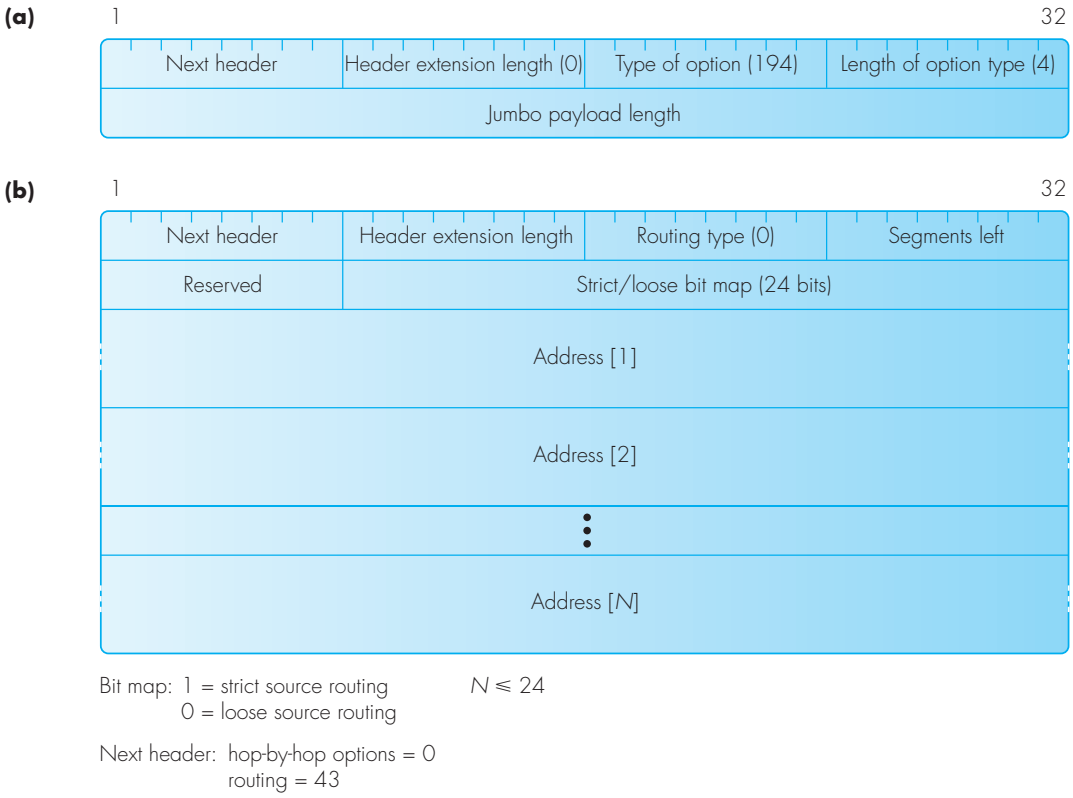


Figure 6.43 Extension header formats: (a) hop-by-hop options; (b) routing.

header is 43. Currently only one type of routing header has been defined, the format of which is shown in Figure 6.43(b).

The *header extension length* is the length of the header in multiples of 8 bytes, excluding the first 8 bytes. Hence, as we can deduce from the figure, this is equal to two times the number of (16-byte) addresses present in the header. This must be an even number less than or equal to 46. The *segments left* field is an index to the list of addresses that are present. It is initialized to 0 and is then incremented by the next router in the list as it is visited. The maximum therefore is 23.

The second 4-byte word contains a *reserved* byte followed by a 24-bit field called the *strict/loose bit map*. This contains one bit for each potential address present starting at the leftmost bit. If the related bit is a 1, then the address must be that of a directly attached (neighbor) router – that is, strict source routing. If the bit is a 0, then the address is that of a router with possibly several other routers in between – loose source routing. The latter is used, for example, when tunneling is required to forward the datagram. In the case of strict source routing, at each router the destination address in the main header is changed to that obtained from the list of addresses before the index is incremented. In the case of loose source routing, the destination address will be that of an attached neighbor which is on the shortest-path route to the specified address.

Example 6.7

A datagram is to be sent from a source host with an IPv6 address of A to a destination host with an IPv6 address of B via a path comprising three IPv6 routers. Assuming the addresses of the three routers are R1, R2, and R3 and strict source routing is to be used, (i) state what the contents of the initial values in the various fields in the extension header will be and (ii) list the contents of the source and destination address fields in the main header and the *segments left* field in the extension header as the datagram travels along the defined path.

Answer:

(i) Extension header initial contents:

Next header = Transport layer protocol
Header extension length = $2 \times 3 = 6$
Routing type = 0
Segments left = 0
Strict/loose bit map = 11100000 00000000 00000000
List of addresses = R1, R2, R3 (each of 16 bytes)

(ii) Contents of main header fields:

At source	SA = A	DA = R1	Segments left = 0
At R1	SA = A	DA = R2	Segments left = 1
At R2	SA = A	DA = R3	Segments left = 2

Fragment

This header is present if the original message submitted by the transport layer protocol exceeds the MTU of the path/route to be used. The *next header* value for a *fragment* extension header is 44. The fragmentation and reassembly procedures are similar to those used with IPv4 but, in the case of IPv6, the fragmentation procedure is carried out only in the source host and not by the routers/gateways along the path followed by the packet(s). As we saw in Figure 6.41(a), there is no don't fragment (D) bit in the IPv6 main header since, in order to speed up the processing/routing of packets, IPv6 routers do not support fragmentation. Hence, as we explained in Section 6.6.9, either the minimum MTU size of 576 bytes must be used or the *path MTU discovery* procedure is used to determine if the actual MTU size is greater than this. In either case, if the submitted message (including the transport protocol header) exceeds the chosen MTU (minus the 40 bytes for the IPv6 main header), then the message must be sent in multiple packets, each with a main header and a *fragment* extension header. The various fields and the format of each *fragment* extension header are shown in Figure 6.44(a). An example is shown in Figure 6.44(b).

Each packet contains a main header – plus, if required, a *hop-by-hop options* header and a *routing* header – followed by a *fragment* extension header and the fragment of the message being transmitted. Thus the maximum size of the payload (and hence message fragment) in each packet will be the MTU size being used minus the number of 8-byte fields required for the main header and any extension headers that are present. The *payload length* field in the main header of the first packet indicates the total number of bytes in the message being transmitted – including the IP header – plus the number of bytes that are required for the other extension headers that are being used. The *payload length* in the main header of the remaining packets indicates the number of bytes in the packet following the main header.

The various fields in each *fragment* header have similar functions to those used with IPv4. The *fragment offset* indicates the position of the first byte of the fragment contained within the packet relative to the start of the complete message being transmitted. Its value is in units of 8-bytes. The *M-bit* is the *more fragments bit*; it is a 1 if more fragments follow and a 0 if the packet contains the last fragment. Similarly, the value in the *identification* field is used by the destination host, together with the source address, to relate the data fragments contained within each packet to the same original message. Normally, the source uses a 32-bit counter (that is incremented by 1 for each new message transmitted) to keep track of the next value to use.

Authentication and encapsulating security payload

As we shall expand upon in Section 10.7.1, authentication and the related subject of encryption are both mechanisms that are used to enhance the security of a message during its transfer across a network. In the case of authentication, this enables the recipient of a message to validate that the

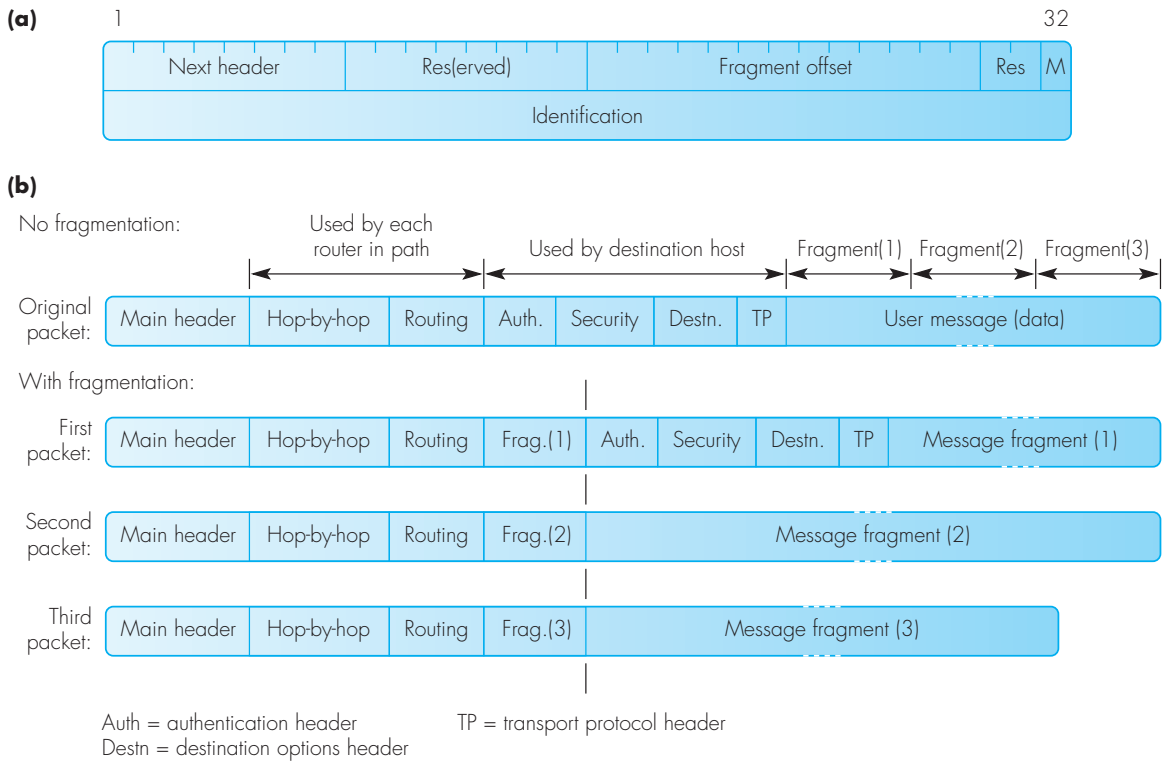


Figure 6.44 IPv6 fragmentation: (a) fragment header fields and format; (b) example.

message was indeed sent by the source address present in the packet/datagram header and not by an impostor. Encryption is concerned with ensuring that the contents of a message can only be read by – and hence have meaning to – the intended recipient. The *authentication* and *encapsulating* security payload (ESP) extension headers are present when both these features are being used at the network layer.

When using IPv6 authentication, prior to any information (packets) being exchanged, the two communicating hosts first use a secure algorithm to exchange secret keys. An example is the MD5 algorithm we describe later in Section 10.3. Then, for each direction of flow, the appropriate key is used to compute a checksum on the contents of the entire datagram/packet. The computed checksum is then carried in the authentication header of the packet. The same computation is repeated at the destination host, and only if the computed checksum is the same as that carried in the authentication header is it acknowledged that the packet originated from the source host address indicated in the main header and also that the packet contents have

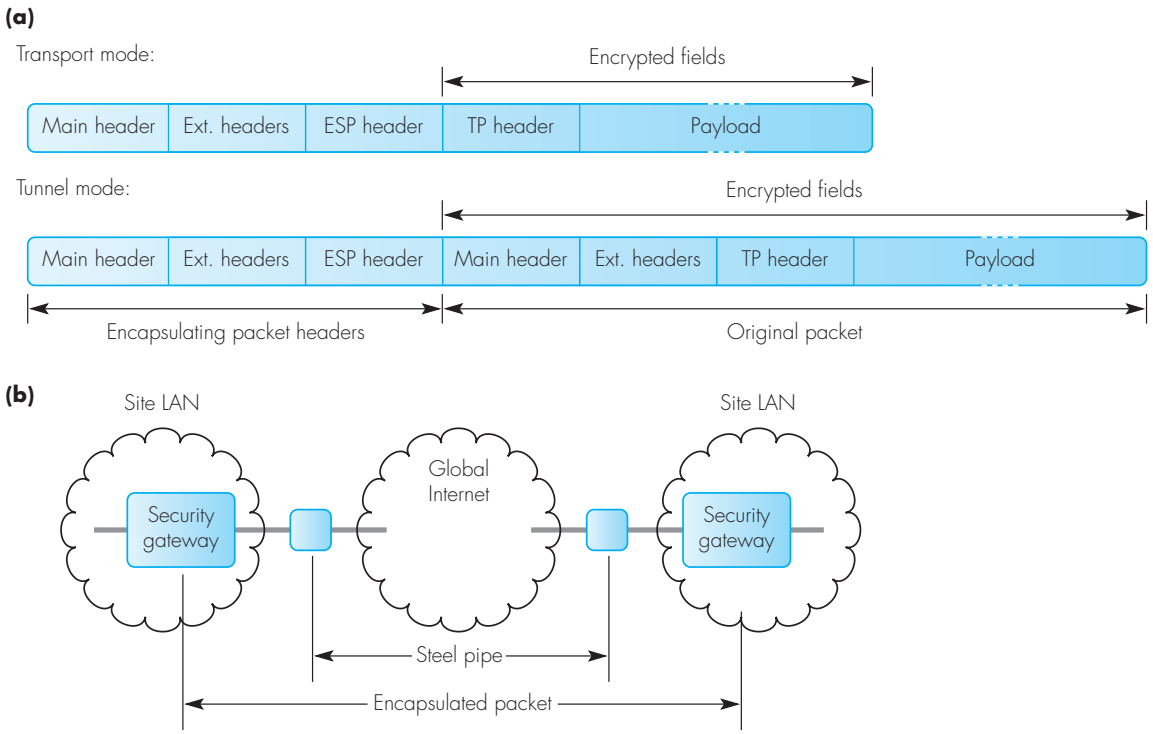


Figure 6.45 Encapsulating security payload: (a) transport and tunnel modes; (b) example application of tunnel mode.

not been modified during its transfer over the network. We discuss the subject of authentication further in Section 10.4.

The encryption algorithm used with the ESP is also based on the use of an agreed secret key. An example is the DES algorithm we describe later in Section 10.2.3. The agreed key is used to encrypt either the transport protocol header and payload parts of each packet or, in some instances, the entire packet including the main header and any extension headers present. In the case of the latter, the encrypted packet is then carried in a second packet containing a completely different main and, if necessary, extension headers. The first is known as **transport mode encryption** and the second **tunnel mode encryption**. Figure 6.45(a) illustrates the principle of operation of both modes.

As we can deduce from the figure, the overheads associated with the tunnel mode are significantly higher than those of the transport mode. The extra security obtained with the tunnel mode is that the information in the main and extension headers of the original packet cannot be interpreted by a person passively monitoring the transmissions on a line. An example use of this mode is in

multisite enterprise networks that use the (public) Internet to transfer packets from one site to another. The general scheme is shown in Figure 6.45(b).

As we will show in Figure 9.4(a) and explain in the accompanying text, associated with each site is a security gateway through which all packet transfers to and from the site take place. Hence to ensure the header information (especially the routing header) of packets is not visible during the transfer of the packet across the Internet, the total packet is encrypted and inserted into a second packet by the IP in the security gateway with the IPv6 address of the two communicating gateways in the source and destination address fields of the main header. The path through the Internet connecting the two gateways is referred to as a **steel pipe**.

Destination options

These are used to convey information that is examined only by the destination host. As we indicated earlier, one of the ways of encoding options is to use the type-length-value format. Hence in order to ensure a header that uses this format comprises a multiple of 8 bytes, two *destination options* have been defined. These are known as Pad1 and Pad*N*, the first to insert one byte of padding and the second two or more bytes of padding. Currently these are the only two options defined.

6.8.4 Autoconfiguration

As we described earlier in Section 6.1, the allocation of the IP addresses for a new network involves a central authority to allocate a new netid – the ICANN – and a local network administrator to manage the allocation of hostids to each attached host/end system. Thus, the allocation, installation and administration of IPv4 addresses can entail considerable effort and expenditure. To alleviate this, IPv6 supports an autoconfiguration facility that enables a host to obtain an IP address dynamically via the network and, in the case of mobile hosts, use it just for the duration of the call/session.

Two types of autoconfiguration are supported. The first involves the host communicating with a local (site) router using a simple (stateless) request-response protocol. The second involves the host communicating with a site (or enterprise) address server using an application protocol known as the **dynamic host configuration protocol** (**DHCP**). The first is suitable for small networks that operate in the broadcast mode (such as an Ethernet LAN) and the second for larger networks in which the allocation of IP addresses needs to be managed.

With the first method, a simple protocol known as **neighbor discovery** (**ND**) is used. As we show in Figure 6.46, this involves the host broadcasting a *router solicitation* packet/message on the subnet/network and the router responding with a *router advertisement* message. Both messages are ICMPv6 messages and hence are carried in an IPv6 packet. The latter is then broadcast over the LAN in a standard frame.

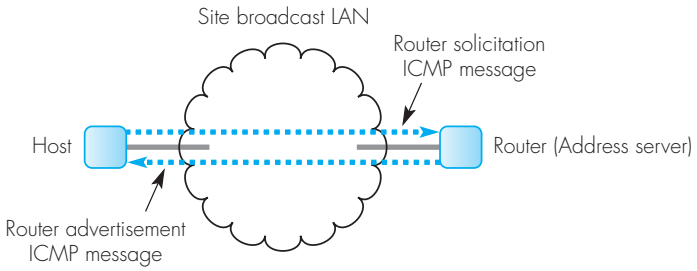


Figure 6.46 Neighbor discovery protocol messages.

The main header of the IPv6 packet containing the router solicitation message has an IPv6 source address created by the host. This is made up of the (standard) link-local address prefix and the 48-bit MAC address of the host's LAN interface. As we indicated earlier, with IPv6 a number of permanent multicast group addresses have been defined including an all-routers group address. Hence the destination address in the packet main header is set to this and, since the packet is broadcast over the LAN, it is received by the ICMP in all the routers that are attached to the LAN.

A single router is selected to process this type of ICMP message and this responds to a router solicitation message with a route advertisement message containing the Internet-wide address prefix for the subnet/network. On receipt of this, the ICMP in the host proceeds to create its own IPv6 address by adding its 48-bit MAC address to the prefix. As we can deduce from this, in relation to IPv4, this is equivalent to the router providing the netid of the site and the host using its own MAC address as the hostid. Also, the same procedure can be used by mobile hosts.

With the second method, the host requests an IPv6 address from the site address server using the DHCP. The **DHCP address server** first validates the request and then allocates an address from the managed list of addresses the server contains. Alternatively if a site does not have its own address server – for example if the site LAN is part of a larger enterprise network — then a designated router acts as a **DHCP relay agent** to forward the request to the DHCP address server.

6.9 IPv6/IPv4 interoperability

The widespread deployment of IPv4 equipment means that the introduction of IPv6 is being carried out in an incremental way. Hence a substantial amount of the ongoing standardization effort associated with IPv6 is concerned with the interoperability of the newer IPv6 equipment with existing IPv4 equipment. Normally, when a new network/internetwork is created, it is based on the IPv6 protocol and, in the context of the existing

(IPv4) Internet, it is referred to as an **IPv6 island**. It is necessary to provide a means of interworking between the two types of network at both the address level and the protocol level. In this section, we identify a number of situations where interoperability is required and describe a selection of the techniques that are used to achieve this.

6.9.1 Dual protocols

Dual stacks are already widely used in networks that use dissimilar protocol stacks, for example a site server that supports IPX on one port and IP on a second port. In a similar way, dual protocols can be used to support both IPv4 and IPv6 concurrently. An example is shown in Figure 6.47.

In this example, the site has a mix of hosts, some that use IPv4 and others IPv6. In order to be able to respond to requests originating from both types of host, the server machine has both an IPv4 and an IPv6 protocol at the network layer. The value in the version field of the datagram header is then used by the link layer protocol to pass a datagram to the appropriate IP. In this way the upper layer protocols are unaware of the type of IP being used at the network layer.

6.9.2 Dual stacks and tunneling

A common requirement is to interconnect two IPv6 islands (networks/internetworks) through an intermediate IPv4 network/internetwork. To achieve this, the gateway/router that connects each IPv6 island to the IPv4 network

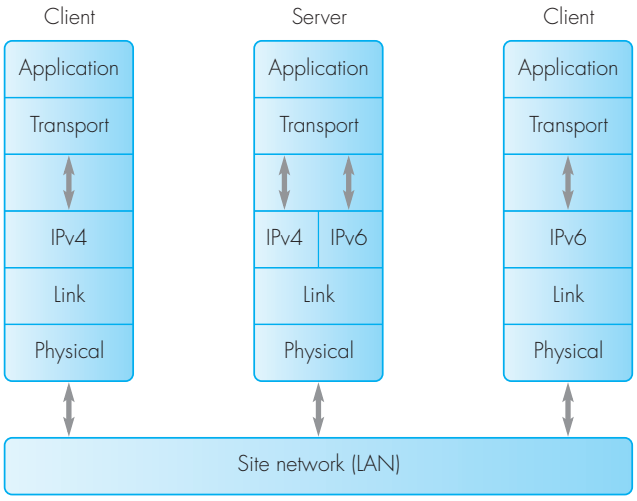


Figure 6.47 IPv6/IPv4 interoperability using dual (IPv6/IPv4) protocols.

must have dual stacks, one that supports IPv6 and the other IPv4. The IPv6 packets are then transferred over the IPv4 network using tunneling. The general approach is illustrated in Figure 6.48(a) and the protocols involved in Figure 6.48(b).

As we showed earlier in Figure 6.16 and explained in the accompanying text, tunneling is used to transfer a packet relating to one type of network layer protocol across a network that uses a different type of network layer protocol. Hence in the example shown in Figure 6.48, each IPv6 packet is transferred from one (IPv6/IPv4) edge gateway to the other edge gateway within an IPv4 packet. As we show in the figure, in order to do this, the two

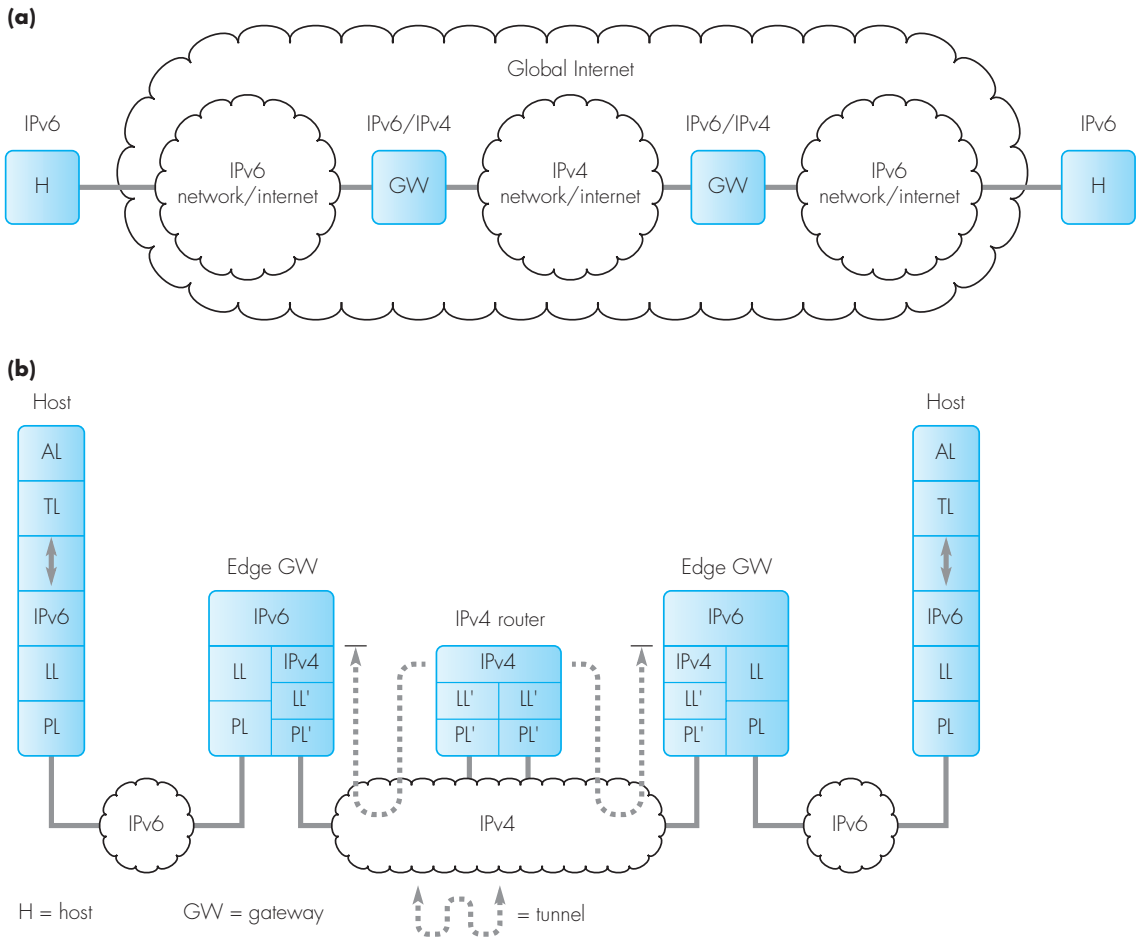


Figure 6.48 IPv6/IPv4 interoperability using dual stacks and tunneling: (a) schematic; (b) protocols.

edge gateways contain dual stacks each of which has a related IPv6/IPv4 address associated with it. Normally, entered by network management, the routing table entry for the remote destination IPv6 host is the IPv4 address of the remote edge gateway.

The IPv6 in each gateway, on determining from its routing table that an (IPv6) packet should be forwarded to a remote IPv6 network via an IPv4 tunnel, passes the packet to the IPv4 protocol together with the IPv4 address of the remote gateway. The IPv4 protocol first encapsulates the IPv6 packet in an IPv4 datagram/packet with the IPv4 address of the remote gateway in the destination address field. It then uses this address to obtain the (IPv4) address of the next-hop router from its own (IPv4) routing table and proceeds to forward the packet over the IPv4 network/internetwork. On receipt of the packet, the IPv4 in the remote gateway, on detecting from its own routing table that it is a tunneled packet, strips off the IPv4 header and passes the payload – containing the original IPv6 packet – to the IPv6 layer. The latter then forwards the packet to the destination host identified in the packet header in the normal way.

6.9.3 Translators

A third type of interoperability requirement is for a host attached to an IPv6 network – and hence having an IPv6 address and using the IPv6 protocol – to communicate with a host that is attached to an IPv4 network – hence having an IPv4 address and using the IPv4 protocol. In this case, both the addresses and the packet formats are different and so a translation operation must be carried out by any intermediate routers/gateways. As we show in Figure 6.49, the translations can be performed at either the network layer – part (a) – or the application layer – part (b).

Using the first approach, on receipt of an IPv6/IPv4 packet, this is converted into a semantically equivalent IPv4 /IPv6 packet. This involves a **network address translator** (**NAT**) and a **protocol translator** (**PT**). The intermediate gateway is then known as a **NAT-PT gateway**. As we explained earlier in Section 6.8.2, an IPv4 address can be embedded in an IPv6 address. Hence to send a datagram/packet from a host with an IPv6 address to a host with an IPv4 address, the NAT in the gateway can readily obtain the destination IPv4 address from the destination address in the main header of the IPv6 packet. The issue is what the source address in the IPv4 packet header should be.

A proposed solution is for the NAT to be allocated a block of hostids for the destination IPv4 network. These, together with the netid of the destination network, then form a block of unique IPv4 addresses. For each new call/session, the NAT allocates an unused IPv4 address from this block for the duration of the call/session. It then makes an entry in a table containing the IPv6 address of the V6 host and its equivalent (temporary) IPv4 address. The NAT translates between one address and the other as the

packet is relayed. A timeout is applied to the use of such addresses and, if no packets are received within the timeout interval, the address is transferred back to the free address pool.

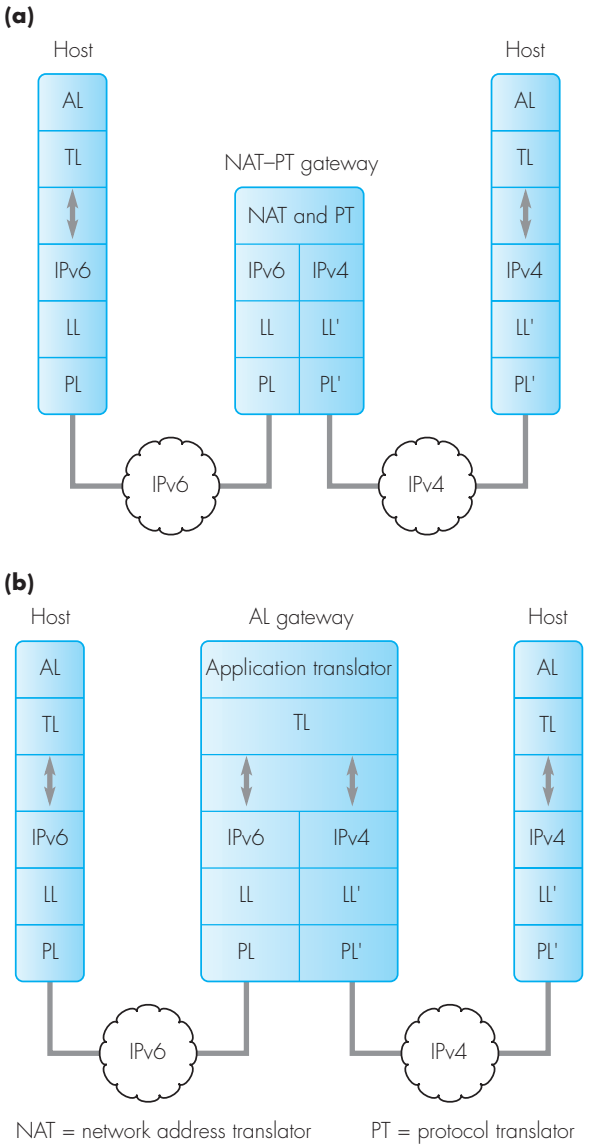


Figure 6.49 IPv6/IPv4 interoperability using translators: (a) network level; (b) application level.

The protocol translation operation is concerned with translating the remaining fields in the packet header and, in the case of ICMP messages, converting ICMPv4 messages into and from ICMPv6 messages. As we indicated in Section 6.8.1, most of the fields in the IPv6 main header have the same meaning as those in the IPv4 header and hence their translation is relatively straightforward. In general, however, there is no attempt to translate the fields in the options part. Similarly, since ICMPv6 messages have different *type* fields, the main translation performed is limited to changing this field. For example, the two ICMPv4 query messages have *type* values of 8 and 0 and the corresponding ICMPv6 messages are 128 and 129.

The use of a NAT-PT gateway works providing the packet payload does not contain any network addresses. Although this is the case for most application protocols, a small number do. The FTP application protocol, for example, often has IP addresses embedded within its protocol messages. In such cases, therefore, the translation operation must be carried out at the application layer. The associated gateway is then known as an **application level gateway** (**ALG**). This requires a separate translation program for each application protocol. Normally, therefore, most translations are performed at the network layer – using a NAT and a PT – and only the translations relating to application protocols such as FTP are carried out in the application layer.

Summary

A summary of the topics discussed in this chapter is given in Figure 6.50.

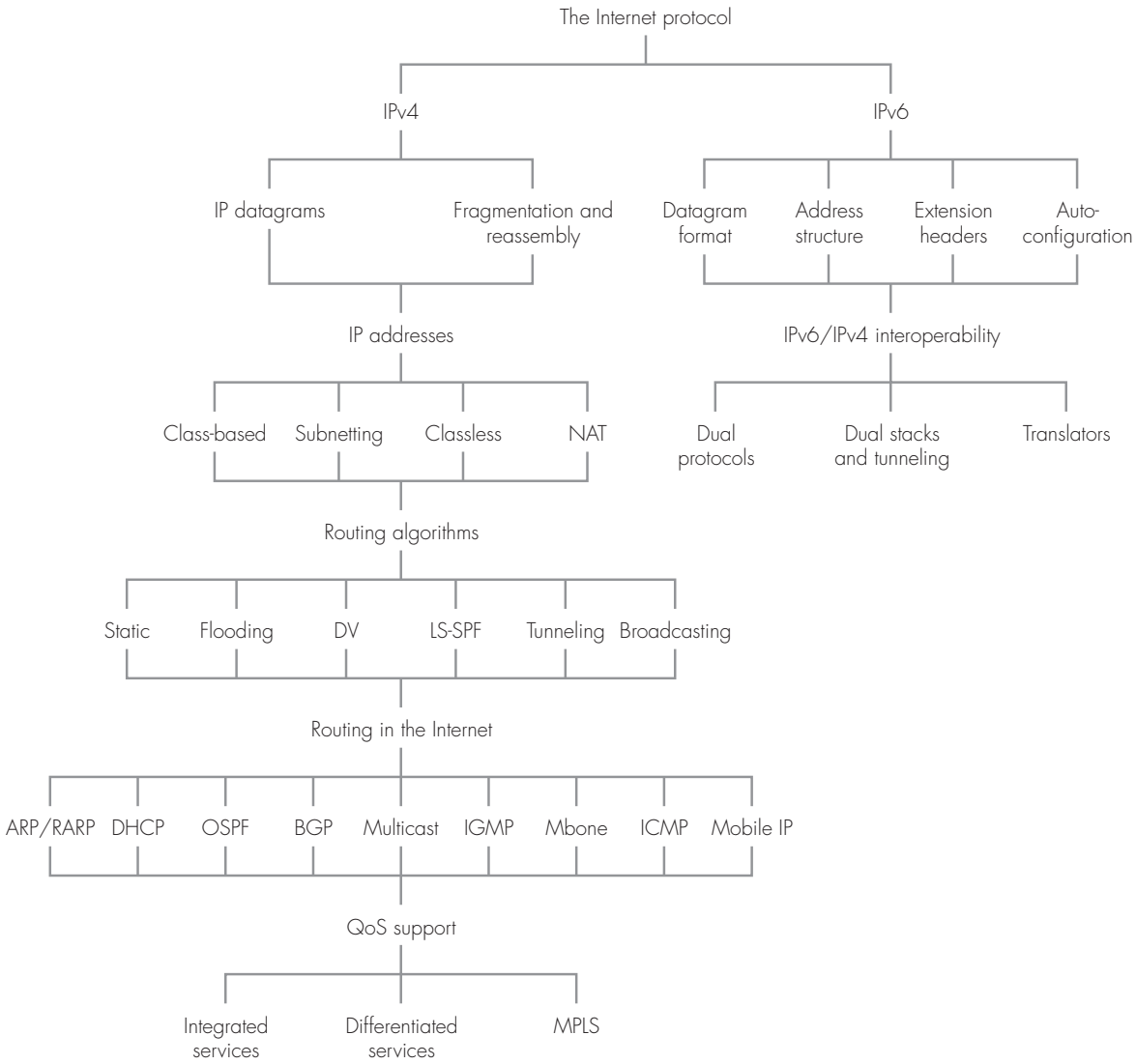


Figure 6.50 Internet protocol summary.

Exercises

Section 6.1

6.1 With the aid of the network schematic shown in Figure 6.1, explain briefly the role of the following network components and protocols:
(i) access network gateway,
(ii) routing gateway,
(iii) internet protocol (IP),
(iv) datagram/packet.

6.2 With the aid of the protocol stack shown in Figure 6.2, explain briefly the meaning of the term "adjunct protocol" and how the IP in the destination host determines to which protocol the contents/payload of a received IP datagram should be passed.

Section 6.2

6.3 In relation to the IP datagram/packet format shown in Figure 6.3, explain the role of the following header fields:
(i) IHL,
(ii) TOS,
(iii) Total length and Identification,
(iv) flag bits,
(v) Fragment offset,
(vi) Time-to-live,
(vii) Protocol,
(viii) Header checksum,
(ix) Options.

Section 6.3

6.4 Assume a message block of 7000 bytes is to be transferred from one host to another as shown in the example in Figure 6.4. In this instance, however, assume the token ring LAN has an MTU of 3000 bytes. Compute the header fields in each IP packet shown in the figure as it flows
(i) over the token ring LAN,
(ii) over the Ethernet LAN.

6.5 State why fragmentation is avoided whenever possible and the steps followed within each host IP to achieve this.

Section 6.4

6.6 List the four types of IP address schemes that are in use and give a brief summary of the use of each scheme.

6.7 Explain the meaning of the term "IP address class" and why these classes were created. Hence, with the aid of the three (unicast) address classes identified in Figure 6.5, identify a particular application for each class.

6.8 State the meaning of the following addresses:
(i) an address with a netid of all 0s,
(ii) an address with a netid of all 1s,
(iii) an address with a hostid of all 0s,
(iv) an address of all 1s.

6.9 Explain the meaning of the term "dotted decimal". Hence derive the netid and hostid for the following IP addresses expressed in dotted decimal notation:
(i) 13.0.0.15,
(ii) 132.128.0.148,
(iii) 220.0.0.0,
(iv) 128.0.0.0,
(v) 224.0.0.1.

6.10 With the aid of the example in Figure 6.6, explain why subnetting was introduced. Hence state the meaning of a subnet router and an address mask.

6.11 A site with a netid of 127.0 uses 20 subnet routers. Suggest a suitable address mask for the site which allows for a degree of expansion in the future. Give an example of a host IP address at this site.

6.12 With the aid of Example 6.3, explain the classless inter-domain routing (CIDR) scheme.

6.13 With the aid of Example 6.4, explain how, with CIDR, multiple address matches are possible with a network that has been allocated a large block of addresses. State which of these matches is chosen and why.

6.14 Explain why the network address translation (NAT) scheme is now used with most new access networks.

6.15 In relation to the NAT scheme shown in Figure 6.7(a), determine the number of host addresses/interfaces that have been declared for private use.

6.16 In relation to the outline operation of NAT shown in Figure 6.7(b), explain:
(i) how the four IP addresses are used and the role of the NAT table,
(ii) how the source and destination addresses in the related IP and TCP fields are derived.

Section 6.5

6.17 State the meaning of the following terms relating to the routing of packets over an internet:
(i) line cost,
(ii) path cost,
(iii) hopcount,
(iv) routing metric,
(v) shortest path.

6.18 With the aid of the routing table entries shown in Figure 6.9, explain the meaning of the terms:
(i) static routing tables,
(ii) next-hop routing,
(iii) optionality principle,
(iv) alternative paths.

6.19 With the aid of the broadcast diagram shown in Figure 6.10, explain:
(i) why the broadcast following the route via R2 is assumed to arrive first
(ii) how duplicate copies of a packet are determined by R3
(iii) how the number of copies of the packet produced is limited
(iv) why flooding is an example of an adaptive/dynamic routing algorithm.

6.20 In relation to the distance vector algorithm, with the aid of the example shown in Figure 6.11, explain:
(i) the meaning of the term "connectivity/adjacency table" and how the table's contents are obtained,
(ii) how the final routing table entries for R3 are built up,
(iii) how a packet from a host attached to netid3 is routed to a host attached to netid1,
(iv) the limitations of the algorithm including how looping may arise.

6.21 Assuming the connectivity/adjacency tables given in Figure 6.12(a), show how the overall network topology is built up by router R3 using the link state algorithm. Hence derive the contents of the netid location table for R3.

6.22 Assuming the initial network topology shown in Figure 6.13(a), use the Dijkstra algorithm to derive the shortest paths from R3 to each other router. State the meaning of the terms "tentative" and "permanent" relating to the algorithm and the implications of alternative paths/routers.

6.23 Using the set of link state, routing, and connectivity tables for R1, R2 and R3 in Figure 6.15, explain how a packet received by R3 from G3 with a destination netid of 1 is routed using hop-by-hop routing.

6.24 Explain how a packet received by R3 from G3 with a destination netid of 1 is routed using source routing. Include how the routing tables you use are derived.

6.25 In relation to the link-state algorithm, explain why each link-state message contains a sequence number and a timeout value. How are these used?

6.26 Explain the term "tunneling" and when it is used. Hence with the aid of the schematic diagram shown in Figure 6.16, explain how the host on the left of the diagram sends an IP datagram/packet to a host attached to the Internet. Include in your explanation the role of the two multiprotocol routers.

State an application of tunneling IP packets over an IP network.

6.27 What are the aims of both the reverse path forwarding algorithm and the spanning tree broadcast algorithm?

6.28 Use the final routing tables and broadcast sequence relating to the reverse path forwarding algorithm shown in Figure 6.17 to explain why only the (broadcast) packet received by SR6 from SR3 is broadcast at the fourth stage. What is the number of duplicate broadcasts that occur?

6.29 Assuming the network topology shown in Figure 6.18(a) and that SR3 is an access gateway, determine the spanning tree derived by each subnet router. Use this to derive the broadcast sequence.

Section 6.6

6.30 With the aid of the generalized Internet architecture shown in Figure 6.19, give an example of the type of network that is used at each tier in the hierarchy and the routing method that is used with it.

6.31 Define the terms "IP address", "MAC address", and "hardware/physical address". Also explain the terms "address-pair" and "ARP cache".

6.32 In relation to the simple network topology shown in Figure 6.20, explain why:
(i) on receipt of an ARP request message, each host retains a copy of the IP/MAC address-pair of the source host in its ARP cache,
(ii) on receipt of an ARP reply message, the ARP in the source host makes an entry of the IP/MAC address-pair in its own cache,
(iii) the Lan port of the gateway keeps a copy of the IP/MAC address-pair from each ARP request and reply message that it receives.

6.33 Explain the role of a proxy ARP. Hence explain how an IP packet sent by a host at one site is routed to a host at a different site. Also explain how the reply packet is returned to the host that sent the first packet.

6.34 Explain how the reverse ARP is used to enable a diskless host to determine its own IP address from its local server.

6.35 With the aid of the two frame formats shown in Figure 6.21, explain:
(i) how the MAC sublayer in the receiver determines whether a received frame is in the Ethernet format or IEEE802.3,
(ii) the number of pad bytes required with each frame type.

6.36 State the role of the dynamic host configuration protocol (DHCP).

6.37 Use the example network topology shown in Figure 6.22(a) to describe the DHCP message exchange sequence to obtain an IP address.

6.38 In relation to the simplified Internet structure shown in Figure 6.23, explain the function of the four types of router.

6.39 With the aid of the simplified autonomous system (AS) topology shown in Figure 6.24(a), describe the operation of the OSPF algorithm. First describe how the directed graph is derived and then the derivation of the SPF tree for R7.

6.40 List the message types used with OSPF and explain their function when routing within an AS.

6.41 List the four message types that are used in the border gateway protocol (BGP) and explain their function.

6.42 Given the example backbone topology shown in Figure 6.25, give an example of an update message that changes the route followed between a pair of boundary routers.

6.43 In relation to multicasting over a LAN, describe how ICANN controls the allocation of

multicast addresses. Also explain how the 48-bit MAC address and 28-bit IP address of a host are derived from the allocated address. Hence with the aid of the schematic diagram shown in Figure 6.27(b), describe how a host joins a multicast session that is taking place over the LAN. Include the role of the multicast address table and group address table held by each member of the group.

6.44 What is the meaning of the term "multicast router"? Outline the sequence of steps that are followed to route an IP packet with a multicast address over the Internet.

6.45 Assume the same topology, multicast address table contents, routing table contents, and routing table entries as shown in Figure 6.28. Assuming the DVMRP, explain how a packet arriving from one of its local networks with a multicast address of C is routed by MR3 to all the other MRs that have an interest in this packet.

6.46 Repeat Exercise 6.45 but this time using the MOSPF routing protocol and the spanning tree shown in Figure 6.29(b).

6.47 What is the role of the IGMP protocol?

With the aid of the example shown in Figure 6.30, explain how a host that is attached to a local network/subnet of an MR joins a multicast session. Include in your explanation the table entries retained by both the host and the MR and how multicast packets relating to the session are then routed to the host.

6.48 With the aid of the example shown in Figure 6.30, explain the procedure followed when a host that is attached to a local network/subnet of an MR leaves a multicast session.

6.49 The multicast backbone (M-bone) network shown in Figures 6.28 and 6.29 comprised a set of multicast routers interconnected by single links. In practice, these are logical links since each may comprise multiple interconnected routers that do not take part in multicast routing. Explain how a multicast packet is sent from one mrouter to another using IP tunneling.

6.50 Explain briefly the role of the ICMP protocol and the different procedures associated with it. Hence explain how the path MTU discovery procedure is used to determine the MTU of a path/route prior to sending any datagrams.

6.51 In the simplified network architecture shown in Figure 6.32, explain the use of the following terms:

(i) home agent,
(ii) foreign agent.

Discuss the issues to be resolved when Host A wants to communicate with Host B.

6.52 In relation to the example shown in Figure 6.33, use a sequence diagram to illustrate the exchange of mobile IP messages in order to register a mobile host (Host B) with its HA and FA. List the main addresses held by both the HA and the FA after the registration procedure is complete.

6.53 Using the example shown in Figure 6.34, describe the indirect routing method used to route packets between Host A and Host B once Host B has been registered with the HA and FA.

Section 6.7

6.54 Discuss the reasons why improved levels of QoS support are now being used within the Internet.

6.55 Describe the role and principle of operation of the following control mechanisms used within Internet routers:

(i) token bucket filter,
(ii) weighted fair queuing,
(iii) random early detection.

6.56 Define the three different classes of service used with the IntServ scheme. With the aid of the network topology shown in Figure 6.35(a) describe the operation of the resource reservation protocol (RSVP). Include in your

description the meaning/role of the following:
(i) path, reserve, and path-tear messages,
(ii) path-state table,
(iii) cleanup timer,
(iv) soft-state.

6.57 Define the usage of the type of service (ToS) field in each packet header with the DiffServ scheme including the meaning of the term "DS packet codepoint".

6.58 With the aid of the general architecture shown in Figure 6.35(b), describe the operation of the DiffServ scheme. Include in your description the meaning/role of the following components of an ingress router:
(i) behavior aggregate,
(ii) traffic meter module,
(iii) MF classifier,
(iv) marker module,
(v) shaper/dropper.

6.59 Use the schematic diagram of a router in Figure 6.36 to explain the packet forwarding procedure in a conventional router.

6.60 Explain the terms traffic engineering, class-based queuing, shaping and grooming in an MPLS network.

6.61 In relation to the MPLS network architecture shown in Figure 6.37, explain where and why the various QoS mechanisms are located.

6.62 Using the example topology shown in Figure 6.38, insert two further sets of table entries to illustrate the label switching procedure.

6.63 Use the schematic diagram of the area border router/label edge router shown in Figure 6.39 to explain the MPLS forwarding procedure. Include in your explanation the role of the packet classifier, the LSP table, the output queue interfaces and the associated scheduling rules.

6.64 Explain how alternative routes to those computed using OSPF are used with the constraint-routed label distribution protocol (CR-LDP). Describe how the alternative routing tables are downloaded after the CR analysis.

Also explain the meaning/role of the following components of a core router:
(i) BA classifier,
(ii) per-hop behavior,
(iii) expedited forwarding,
(iv) assured forwarding.

Section 6.8

6.65 Discuss the reasons behind the definition of IP version 6, IPv6/IPng, including the main new features associated with it.

6.66 With the aid of the frame format shown in Figure 6.41(a), explain the role of the following fields in the IPv6 packet header:
(i) traffic class,
(ii) flow label,
(iii) payload length (and how this differs from the total length in an IPv4 packet header),
(iv) next header,
(v) hop limit,
(vi) source and destination addresses.

6.67 In relation to IPv6 addresses, with the aid of the prefix formats shown in Figures 6.42(a) and (b), explain the meaning/use of:
(i) address aggregation,
(ii) prefix formats,
(iii) embedded IPv4 addresses.

6.68 With the aid of the frame format shown in Figure 6.42(c), explain the meaning/use of the following IPv6 fields:
(i) registry,
(ii) top-level aggregators,
(iii) next-level agrregators,
(iv) site-level aggregators,
(v) interface ID.
Comment on the implications of adopting a hierarchical address structure.

6.69 With the aid of examples, explain the use of a link local-use address and a site local-use address.

6.70 Explain the format and use of
(i) a multicast address,
(ii) an anycast address.

6.71 With the aid of examples, show how an IPv6 address can be represented:
(i) in hexadecimal form,
(ii) with leading zeros removed,
(iii) when it contains an IPv4 embedded address.

6.72 Explain the role of the extension headers that may be present in an IPv6 packet. List the six types of extension header and state their use. Also, with the aid of examples, state the position and order of the extension headers in relation to the main header.

6.73 The fields in an options extension header are encoded using a type-length-value format. Use the hop-by-hop options header as an example to explain this format.

6.74 In relation to the routing extension header, explain:
(i) the difference between strict and loose source routing, and the associated bit map,
(ii) the use of the segments left field.

6.75 In relation to the packet formats shown in Figure 6.44, explain:
(i) the meaning and use of the identification field and the M-bits in each extension header,
(ii) why the hop-by-hop and routing headers are present in each fragment packet.

6.76 In relation to the encapsulating security payload header, with the aid of diagrams, explain:
(i) the difference between transport mode and tunnel mode encryption,
(ii) the meaning and use of the term "steel pipe".

6.77 State the aim of the autoconfiguration procedure used with IPv6 and the application domain of:
(i) the neighbor discovery (ND) protocol,
(ii) the dynamic host configuration protocol (DHCP).

6.78 With the aid of Figure 6.46, explain the operation of the ND protocol. Include the role of the router solicitation and router advertisement messages and how a host creates its own IP address.

6.79 Explain how an IPv6 address is obtained:
(i) using a DHCP address server,
(ii) using a DHCP relay agent.

Section 6.9

6.80 With the aid of Figure 6.47, explain how a LAN server can respond to requests from both an IPv4 and an IPv6 client using dual protocols.

6.81 With the aid of Figure 6.48, explain how two hosts, each of which is attached to a different IPv6 network, communicate with each other if the two IPv6 networks are interconnected using an IPv4 network. Include the addresses that are used in each message transfer.

6.82 State the meaning of the terms "network address translation" (NAT) and "protocol translation" (PT). Hence, with the aid of the schematic diagram shown in Figure 6.49(a), explain the role and operation of a NAT-PT gateway. Include what the source address in each IPv6 packet should be.

6.83 Identify when the use of a NAT-PT gateway is not practical. Hence, with the aid of the schematic diagram shown in Figure 6.49(b), explain the role of an application level gateway.

transport protocols

7.1 Introduction

As we saw earlier in Section 1.2, although the range of Internet applications (and the different types of access network used to support them) are many and varied, the protocol suites associated with the different application/network combinations have a common structure. As we saw in Section 1.2.2, the different types of network operate in a variety of modes – circuit-switched or packet-switched, connection-oriented or connectionless – and hence each type of network has a different set of protocols for interfacing to it. Above the network-layer protocol, however, all protocol suites comprise one or more application protocols and a number of what are called application-support protocols.

For example, in order to mask the application protocols from the services provided by the different types of network protocols, all protocol suites have one or more transport protocols. These provide the application protocols with a network-independent information interchange service and, in the case of the TCP/IP suite, they are the **transmission control protocol** (**TCP**) and the **user datagram protocol** (**UDP**). TCP provides a connection-oriented (reliable) service and UDP a connectionless (best-effort) service. Normally, both protocols are present in the suite and the choice of protocol used is determined by the requirements of the application. In addition, when the

application involves the transfer of streams of audio and/or video in real time, the timing information required by the receiver to synchronize the incoming streams is provided by the **real-time transport protocol** (**RTP**) and its associated **real-time transport control protocol** (**RTCP**). There is also a version of TCP for use with wireless networks. We describe the operation of these five protocols and the services they provide to application protocols.

7.2 TCP/IP protocol suite

Before describing the various protocols, it will be helpful to illustrate the position of each protocol relative to the others in the TCP/IP suite. This is shown in Figure 7.1. Normally, the IP protocols and network-dependent protocols below them are all part of the operating system kernel with the various application protocols implemented as separate programs/processes. The two transport protocols, TCP and UDP, are then implemented to run either within the operating system kernel, as separate programs/processes, or in a library package linked to the application program.

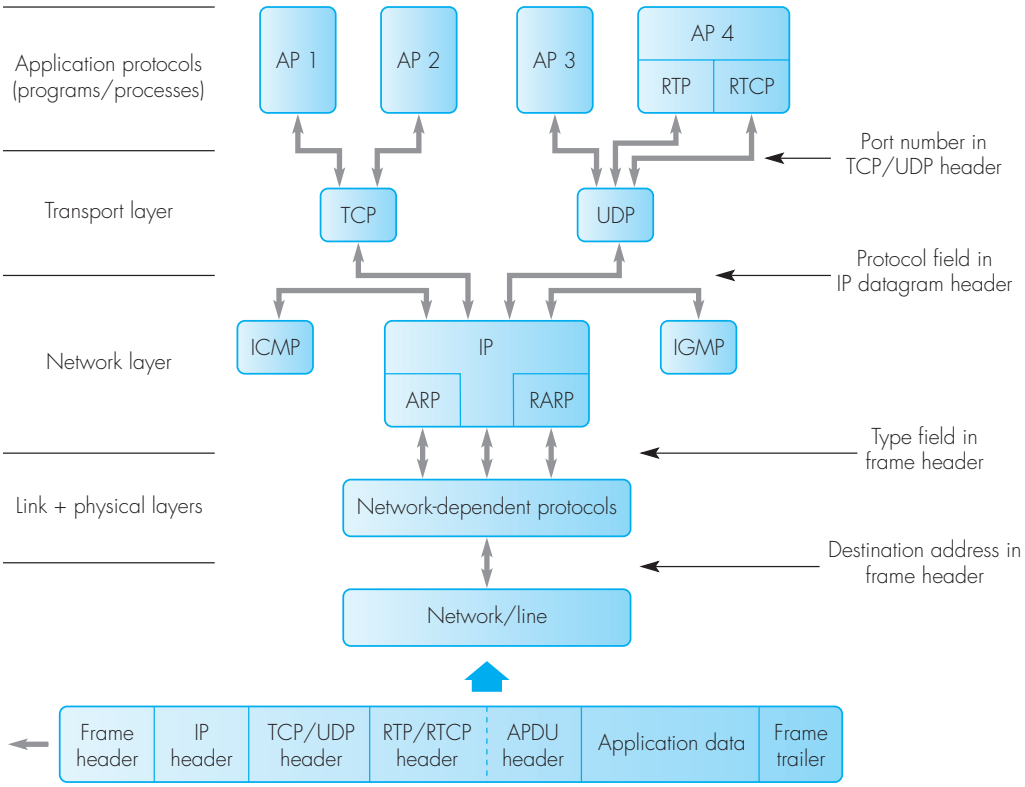


Figure 7.1 TCP/IP protocol suite and interlayer address selectors.

With most networked applications, the client–server paradigm is used. The client application protocol/program runs in one computer – typically a PC or workstation – and this communicates with a similar (peer) application program that runs (normally continuously) in a remote server computer. Examples of applications of this type are file transfers and the messages associated with electronic mail, both of which require a reliable service; that is, the transferred information should be free of transmission errors and the messages delivered in the same sequence that they were submitted. Hence applications of this type use the reliable service provided by TCP.

Thus the role of TCP is to convert the best-effort service provided by IP into a reliable service. For other applications, a simple best-effort service is acceptable and hence they use UDP as the transport protocol. Examples of applications of this type are interpersonal applications that involve the transfer of streams of (compressed) audio and/or video in real time. Clearly, since new information is being received and output continuously, it is inappropriate to request blocks that are received with errors to be retransmitted. Also, it is for applications of this type that RTP and RTCP are used. Other applications that use UDP are application protocols such as HTTP and SNMP, both of which involve a single request-response message exchange.

As we saw in Figure 6.1 and its accompanying text, all message blocks – protocol data units (PDUs) – relating to the protocols that use the services of the IP layer are transferred in an IP datagram. Hence, as we can deduce from Figure 7.1, since there are a number of protocols that use the services of IP – TCP, UDP, ICMP and IGMP – it is necessary for IP to have some means of identifying the protocol to which the contents of the datagram relate. As we saw in Section 6.2, this is the role of the *protocol* field in each IP datagram header. Similarly, since a number of different application protocols may use the services of both TCP and UDP, it is also necessary for both these protocols to have a field in their respective PDU header that identifies the application protocol to which the PDU contents relate. As we shall see, this is the role of the source and destination *port numbers* that are present in the header of the PDUs of both protocols. In addition, since a server application receives requests from multiple clients, in order for the server to send the responses to the correct clients, both the source port number and the source IP address from the IP datagram header are sent to the application protocol with the TCP/UDP contents.

In general, within the client host, the port number of the source application protocol has only local significance and a new port number is allocated for each new transfer request. Normally, therefore, client port numbers are called **ephemeral ports** as they are short-lived. The port numbers of the peer application protocol in the server application protocols are fixed and are known as **well-known port numbers**. Their allocation is managed by ICANN and they are in the range 1 through to 1023. For example, the well-known port number of the server-side of the file transfer (application) protocol (FTP) is 21. Normally, ephemeral port numbers are allocated in the range 1024 through to 5000.

As we can see in Figure 7.2, all the protocols in both the application and transport layers communicate directly with a similar peer protocol in the remote host computer (end system). The protocols in both these layers are said, therefore, to communicate on an end-to-end basis. In contrast, the IP protocols present in each of the two communicating hosts are network-interface protocols. These, together with the IP in each intermediate gateway/router involved, carry out the transfer of the datagram across the internetwork. The IP protocol in each host is said to have local significance and the routing of each datagram is carried out on a hop-by-hop basis.

7.3 TCP

The transmission control protocol (TCP) provides two communicating peer application protocols – normally one in a client computer and the other in a server computer – with a two-way, reliable data interchange service. Although the APDUs associated with an application protocol have a defined structure, this is transparent to the two communicating peer TCP protocol entities which treat all the data submitted by each local application entity as a stream of bytes. The stream of bytes flowing in each direction is then transferred (over the underlying network/internet) from one TCP entity to the other in a reliable way; that is, to a high probability, each byte in the stream flowing in

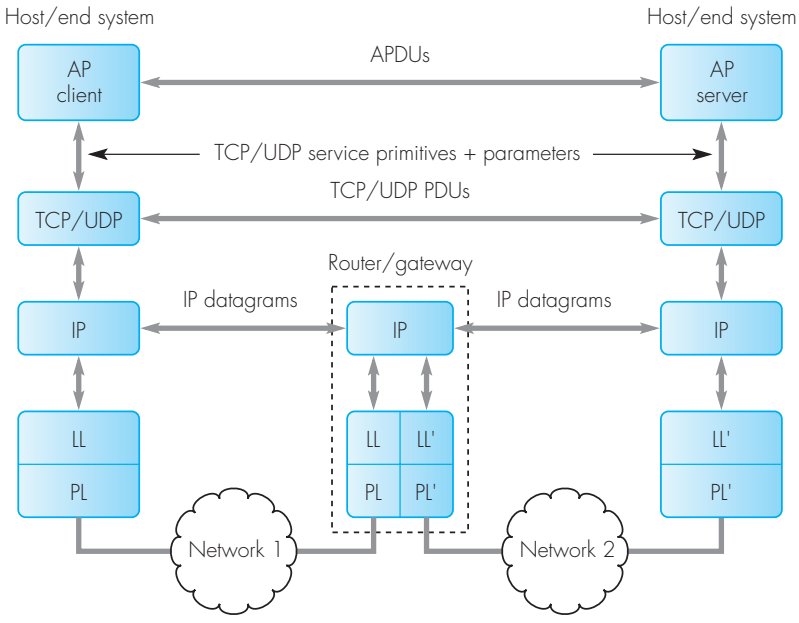


Figure 7.2 TCP/IP protocol suite interlayer communications.

each direction is free of transmission errors, with no lost or duplicate bytes, and the bytes are delivered in the same sequence as they were submitted. The service provided by TCP is known, therefore, as a **reliable stream service**.

As we explained in Section 6.2, the service provided by IP is an unreliable best-effort service. Hence in order to provide a reliable service, before any data is transferred between the two TCP entities, a logical connection is first established between them in order for the sequence numbers in each TCP entity – which are required for error correction and flow control purposes – to be initialized. Also, once all data has been transferred in both directions, the logical connection is closed.

During the actual data transfer phase, in order for the receiving TCP to detect the presence of transmission errors, each TCP entity divides the submitted stream of bytes into blocks known as **segments**. For interactive applications involving a user at a terminal, a segment may contain just a single byte, while for large file transfers a segment may contain many bytes. There is an agreed **maximum segment size** (**MSS**) used with a connection that is established by the two peer TCP entities during the setting up of the connection. This is such that an acceptable proportion of segments are received by the destination free of transmission errors. The default MSS is 536 bytes although larger sizes can be agreed. Normally, the size chosen is such that no fragmentation is necessary during the transfer of a segment over the network/internet and hence is determined by the path MTU. The TCP protocol then includes a retransmission procedure in order to obtain error-free copies of those segments that are received with transmission errors.

In addition, the TCP protocol includes a flow control procedure to ensure no data is lost when the TCP entity in a fast host – a large server for example – is sending data to the TCP in a slower host such as a PC. It also includes a congestion control procedure which endeavors to control the rate of entry of segments into the network/internet to the rate at which segments are leaving.

In the following subsections we discuss firstly the user services provided by TCP, then selected aspects of the operation of the TCP protocol, and finally the formal specification of the protocol. Collectively these are defined in RFCs 793, 1122 and 1323.

7.3.1 User services

The most widely used set of user service primitives associated with TCP are the **socket primitives** used with Berkeley Unix. Hence, although there are a number of alternative primitives, in order to describe the principles involved, we shall restrict our discussion to these. They are operating system calls and collectively form what is called an **application program interface** (**API**) to the underlying TCP/IP protocol stack. A typical list of primitives is given in Table 7.1 and their use is shown in diagrammatic form in Figure 7.3.

Table 7.1 List of socket primitives associated with TCP and their parameters.

Primitive	*Parameters*
socket ( )	service type, protocol, address format, return value = socket descriptor or error code
bind ( )	socket descriptor, socket address (= host IP address + port number), return value = success or error code
listen ( )	socket descriptor, maximum queue length, return value = success or error code
accept ( )	socket descriptor, socket address, return value = success or error code
connect ( )	socket descriptor, local port number, destination port number, destination IP address, precedence, optional data (for example a user name and a password), return value = success or error code
send ( )	socket descriptor, pointer to message buffer containing the data to send, data length (in bytes), push flag, urgent flag, return value = success or error code
receive ( )	socket descriptor, pointer to a message buffer into which the data should be put, length of the buffer, return value = success or end of file (EOF) or error code
close ( )	socket descriptor, return value = success or error code
shutdown ( )	socket descriptor, return value = success or error code

Each of the two peer user application protocols/processes (APs) first creates a communications channel between itself and its local TCP entity. This is called a **socket** or **endpoint** and, in the case of the server AP, involves the AP issuing a sequence of primitive (also known as system of function) calls each with a defined set of parameters associated with it: *socket( )*, *bind( )*, *listen( )*, *accept( )*. Each call has a return value(s) or an error code associated with it.

The parameters associated with the *socket( )* primitive include the service required (reliable stream service), the protocol (TCP), and the address format (Internet). Once a socket has been created – and send/receive memory buffers allocated – a **socket descriptor** is returned to the AP which it then uses with each of the subsequent primitive calls. The AP then issues a

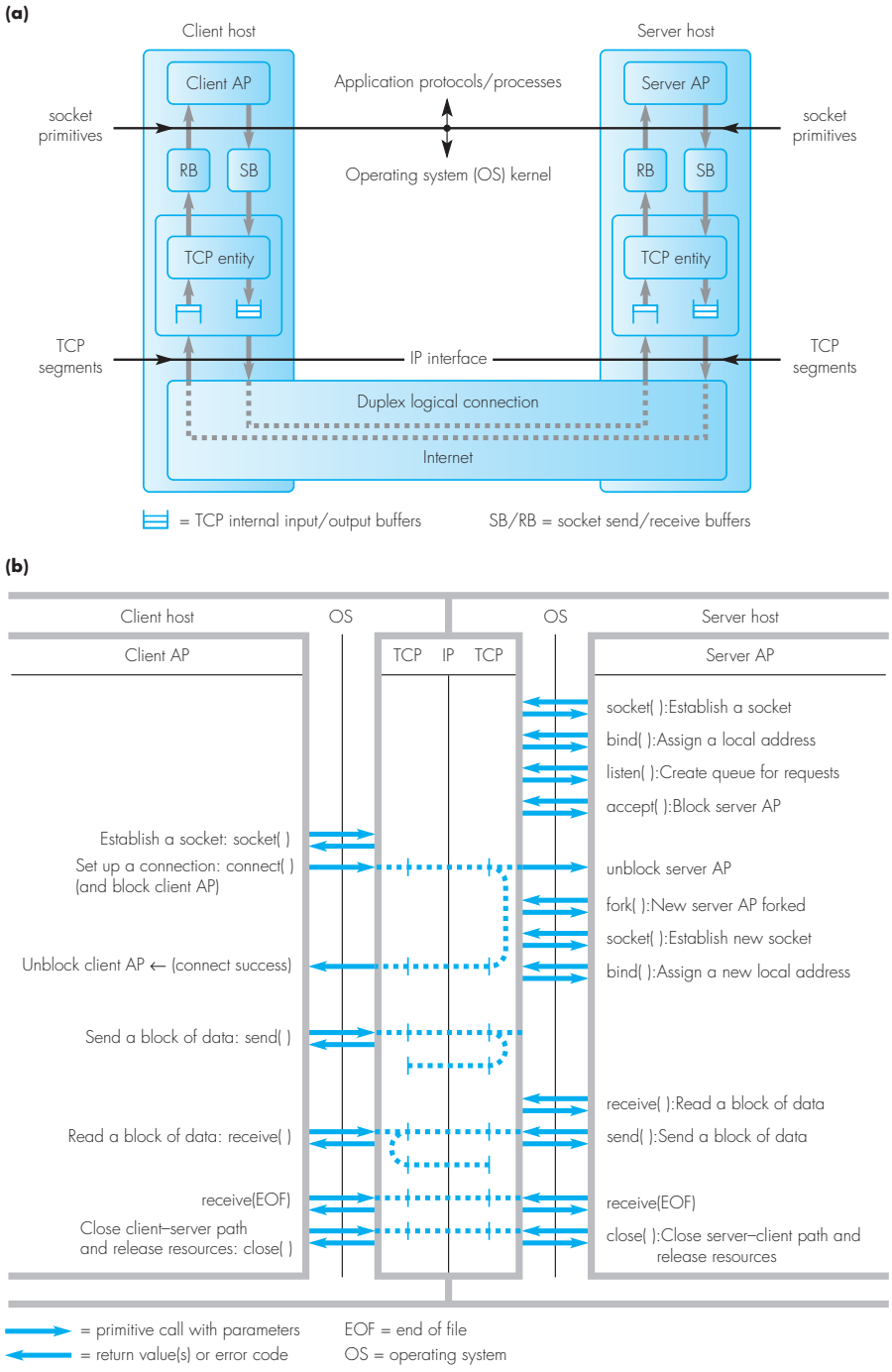


Figure 7.3 TCP socket primitives: (a) socket interface; (b) primitives and their use.

bind( ) primitive, which, in addition to the socket descriptor, has an address parameter associated with it. This is the address the AP wishes to be assigned to the newly created socket and is called the **socket address**. This comprises the Internet-wide IP address of the host and, in the case of the server AP, the 16-bit well-known port number associated with this type of application protocol (FTP and so on).

The *listen( )* primitive call results in the local TCP entity creating a queue (whose maximum length is given as a parameter) to hold incoming connection requests for the server AP. The *accept( )* primitive is then used to put the AP in the blocked state waiting for an incoming connection request to be received from a client TCP entity. Collectively, this sequence of four primitives forms what is called a **passive-open**.

In the case of a client AP, since it can only set up a single TCP connection at a time and the socket address has only local significance, it simply issues a *socket( )* primitive to create a new socket with the same parameters as those used by the server (AP). This is then followed by a *connect( )* primitive, which, in addition to the locally allocated socket descriptor, has parameters that contain the IP address of the remote (server) host, the well-known port number of the required server AP, the local port number that has been assigned to this socket by the client AP, a precedence value, and an optional item of data such as a user name and a password.

The local port number, together with the host IP address, forms the address to be assigned to this socket. The precedence parameter is a collection of parameters that enable the IP protocol to specify the contents of the *type of service* field in the header of the IP datagram that is used to transfer the segments associated with the connection over the Internet. We identified the contents of this field in Figure 6.2 when we discussed the operation of the IP protocol. Note that the IP address of the remote (server) host and the precedence parameters are used by the IP protocol and not TCP. They are examples of what are called **pass-through parameters**, that is, a parameter that is passed down from one protocol layer to another without modification.

Once the *connect( )* call has been made, this results in the calling AP being put into the blocked state while the local TCP entity initiates the setting up of a logical connection between itself and the TCP entity in the server. Collectively, these two primitives form what is called an **active-open**.

The TCP entity in a client host may support multiple concurrent connections involving different user APs. Similarly, the TCP entity in a server may support multiple connections involving different clients. Hence in order for the two TCP entities to relate each received segment to the correct connection, when each new connection is established, both TCP entities create a **connection record** for it. This is a data structure that contains a *connection identifier* (comprising the pair of socket addresses associated with the connection), the agreed *MSS* for the connection, the *initial sequence number* (associated with the acknowledgment procedure) to be used in each direction, the *precedence value*, the *size of the window* associated with the TCP flow control procedure, and a number of fields associated with the operation of the protocol entity including *state variables* and the current state of the protocol entity.

At the server side, when a new connection request (PDU) is received, the server AP is unblocked and proceeds to create a new instance of the server AP to service this connection. Typically, this is carried out using the Unix *fork primitive*. A new socket between the new AP and the local TCP entity is then created and this is used to process the remaining primitives associated with this connection. The parent server AP then either returns to the blocked state waiting for a new connection request to arrive or, if one is already waiting in the server queue, proceeds to process the new request. Once a new instance of the server AP has been created and linked to its local TCP entity by a socket, both the client and server APs can then initiate the transfer of blocks of data in each direction using the *send( )* and *receive( )* primitives.

Associated with each socket is a *send buffer* and a *receive buffer*. The send buffer is used by the AP to transfer a block of data to its local TCP entity for sending over the connection. Similarly, the receive buffer is used by the TCP entity to assemble data received from the connection ready for reading by its local AP. The *send( )* primitive is used by an AP to transfer a block of data of a defined size to the send buffer associated with the socket ready for reading by its local TCP entity. The parameters include the local socket descriptor, a pointer to the memory buffer containing the block of data, and also the length (in bytes) of the block of data. With TCP there is no correlation between the size of the data block(s) submitted (by an AP to its local TCP entity for sending) and the size of the TCP segments that are used to transfer the data over the logical connection. As we saw in Section 7.2, normally the latter is determined by the path MTU and, in many instances, this is much smaller than the size of the data blocks submitted by an AP.

With some applications, however, each submitted data block may be less than the path MTU. For example, in an interactive application involving a user at a keyboard interacting with a remote AP, the input data may comprise just a few bytes/characters. So in order to avoid the local TCP entity waiting for more data to fill an MTU, the user AP can request that a submitted block of data is sent immediately. This is done by setting a parameter associated with the *send( )* primitive called the *push flag*. A second parameter called the *urgent flag* can also be set by a user AP. This again is used with interactive applications to enable, for example, a user AP to abort a remote computation that it has previously started. The (urgent) data – string of characters – associated with the abort command are submitted by the source AP with the urgent flag set. The local TCP entity then ceases waiting for any further data to be submitted and sends what is outstanding, together with the urgent data, immediately. On receipt of this, the remote TCP entity proceeds to interrupt the peer user AP, which then reads the urgent data and acts upon it.

Finally, when a client AP has completed the transfer of all data blocks associated with the connection, it initiates the release of its side of the connection by issuing a *close( )* – or sometimes a *shutdown( )* – primitive. When the server AP is informed of this (by the local TCP entity), assuming it also has finished sending data, it responds by issuing a *close( )* primitive to release the other side of the connection. Both TCP entities then delete their connection records

and also the server AP that was forked to service the connection. As we shall expand upon later, the *shutdown( )* primitive is used when only half of the connection is to be closed.

7.3.2 Protocol operation

As we can see from the above, the TCP protocol involves three distinct operations: setting up a logical connection between two previously created sockets, transferring blocks of data reliably over this connection, and closing down the logical connection. In practice, each phase involves the exchange of one or more TCP segments (PDUs) and, since all segments have a common structure, before describing the three phases we first describe the usage of the fields present in each segment header.

Segment format

All segments start with a common 20-byte header. In the case of acknowledgment and flow control segments, this is all that is present. In the case of connection-related segments, an options field may be present and a data field is present when data is being transferred over a connection. The fields making up the header are shown in Figure 7.4(a).

The 16-bit *source port* and *destination port* fields are used to identify the source and destination APs at each end of a connection. Also, together with the 32-bit source and destination IP addresses of the related hosts, they form the 48-bit socket address and the 96-bit connection identifier. Normally, the port number in a client host is assigned by the client AP while the port number in a server is a well-known port.

The *sequence number* performs the same function as the send sequence number in the HDLC protocol and the *acknowledgment number* the same function as the receive sequence number. Also, as with HDLC, a logical connection involves two separate flows, one in each direction. Hence the *sequence number* in a segment relates to the flow of bytes being transmitted by a TCP entity and the *acknowledgment number* relates to the flow of bytes in the reverse direction. However, with the TCP, although data is submitted for transfer in blocks, the flow of data in each direction is treated simply as a stream of bytes for error and flow control purposes. Hence the *sequence* and *acknowledgment numbers* are both 32-bits in length and relate to the position of a byte in the total session stream rather than to the position of a message block in the sequence. The *sequence number* indicates the position of the first byte in the *data* field of the segment relative to the start of the byte stream, while the *acknowledgment number* indicates the byte in the stream flowing in the reverse direction that the TCP entity expects to receive next.

The presence of an *options* field in the segment header means that the header can be of variable length. The *header length* field indicates the number of 32-bit words in the header. The 6-bit *reserved* field, as its name implies, is reserved for possible future use.

All segments have the same header format and the validity of selected fields in the segment header is indicated by the setting of individual bits in

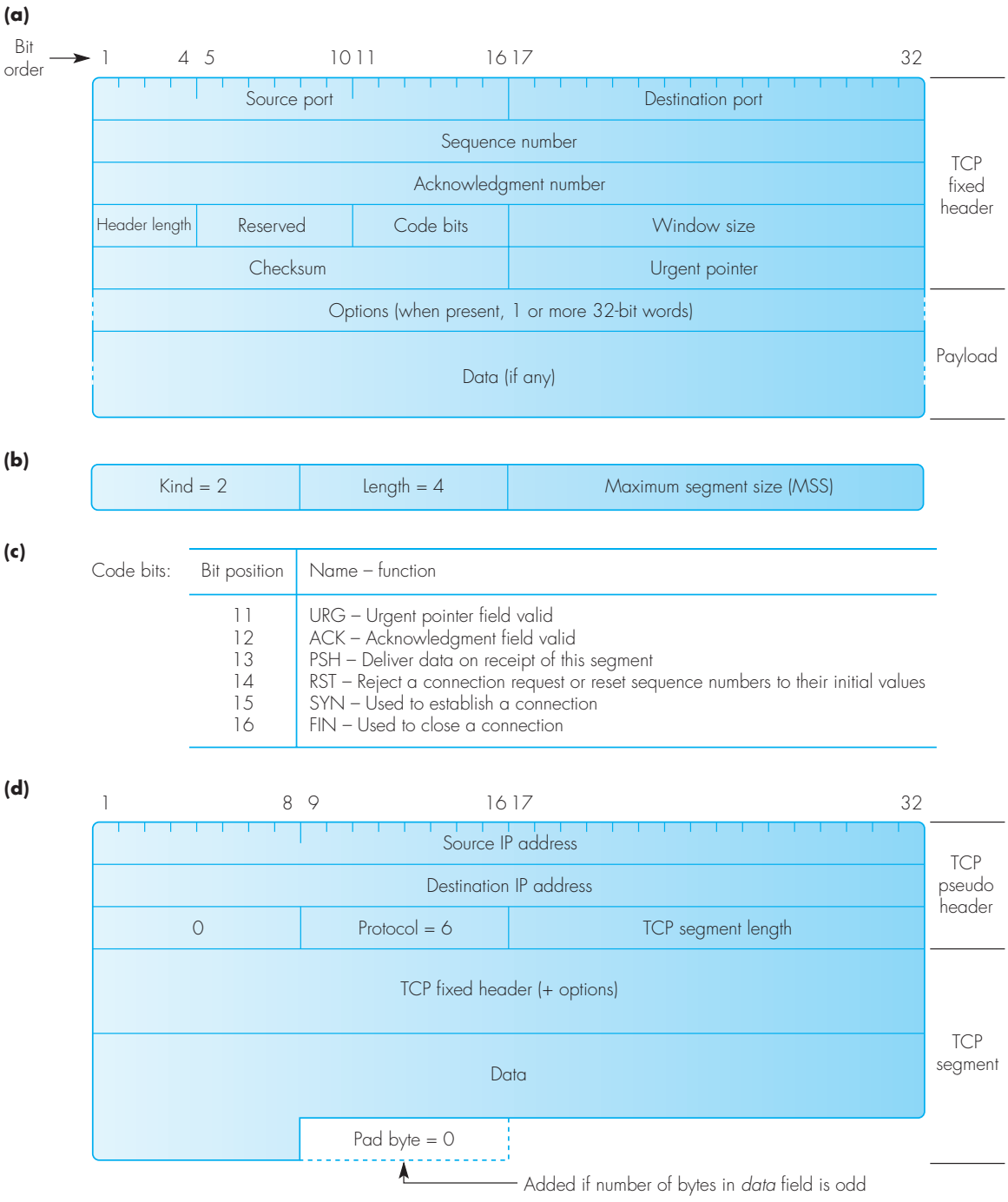


Figure 7.4 TCP segment format: (a) header fields; (b) MSS option format; (c) code bit definitions; (d) pseudo header fields.

the 6-bit *code* field; if a bit is set (a binary 1), the corresponding field is valid. Note that multiple bits can be set in a single segment. The bits have the meaning shown in Figure 7.4(c).

The *window size* field relates to a sliding window flow control scheme the principles of which we considered in Section 1.4.4. The number in the window size indicates the number of data bytes (relative to the byte being acknowledged in the *acknowledgment* field) that the receiver is prepared to accept. It is determined by the amount of unused space in the receive buffer the remote TCP entity is using for this connection. The maximum size of the receive buffer – and hence the maximum size of the window – can be different in each direction and has a default value which, typically, is 4096, 8192 or 16384 bytes.

As we saw in Section 6.2, the checksum field in the header of each IP datagram applies only to the fields in the IP header and not the datagram contents. Hence the *checksum* field in the TCP segment header covers the complete segment; that is, header plus contents. In addition, since only a simple checksum is used to derive the checksum value in the IP header, in order to add an additional level of checking, some selected fields from the IP header are also included in the computation of the TCP checksum. The fields used form what is called the (TCP) **pseudo header** and these are identified in Figure 7.4(d).

As we can see, these are the source and destination IP addresses and the protocol value (=6 for TCP) from the IP header, plus the total byte count of the TCP segment (header plus contents). The computation of the checksum uses the same algorithm as that used by IP. As we saw in Section 6.2, this is computed by treating the complete datagram as being made up of a string of 16-bit words that are then added together using 1s complement arithmetic. Since the number of bytes in the original TCP segment *data* field may be odd, in order to ensure the same checksum is computed by both TCP entities, a **pad byte** of zero is added to the data field whenever the number of bytes in the original *data* field is odd. As we can deduce from this, the byte count of the TCP segment must always be an even integer.

When the URG (urgent) flag is set in the *code* field, the *urgent pointer* field is valid. This indicates the number of bytes in the *data* field that follow the current *sequence number*. This is known as **urgent data** – or sometimes **expedited data** – and, as we mentioned earlier, it should be delivered by the receiving TCP entity immediately it is received.

The *options* field provides the means of adding extra functionality to that covered by the various fields in the segment header. For example, it is used during the connection establishment phase to agree the maximum amount of data in a segment each TCP entity is prepared to accept. During this phase, each indicates its own preferred maximum size and hence can be different for each direction of flow. As we indicated earlier, this is called the maximum segment size (MSS) and excludes the fixed 20-byte segment header. If one of the TCP entities does not specify a preferred maximum size then a default value of 536 bytes is chosen. The TCP entity in all hosts connected to the Internet must accept a segment of up to 556 bytes – 536 plus a 20-byte header – and all IPs must accept a datagram of 576 bytes – 556 bytes plus a further

20-byte (IP) header. Hence the default MSS of 536 bytes ensures the datagram (with the TCP segment in its payload) will be transferred over the Internet without fragmentation. The format of the MSS option is shown in Figure 7.4(b). Also, although not shown, if this is the last or only option present in the header, then a single byte of zero is added to indicate this is the end of the option list.

Connection establishment

On receipt of a *connect( )* primitive (system call) from a client AP, the local TCP entity attempts to set up a logical connection with the TCP entity in the server whose IP address (and port number) are specified in the parameters associated with the primitive. This is achieved using a three-way exchange of segments. Collectively, this is known as a **three-way handshake** procedure and the segments exchanged for a successful *connect( )* call are shown in Figure 7.5(a).

As we indicated in the previous section, the flow of data bytes in each direction of a connection is controlled independently. The TCP entity at each end of a connection starts in the CLOSED state and chooses its own *initial sequence number* (*ISN*). These are both non-zero and change from one connection to another. This ensures that any segments relating to a connection that get delayed during their transfer over the internet – and hence arrive at the client/server after the connection has been closed – do not interfere with the segments relating to the next connection. Normally, each TCP entity maintains a separate 32-bit counter that is incremented at intervals of either 4 or 8 μs. Then, when a new ISN is required, the current contents of the counter are used.

- To establish a connection, the TCP at the client first reads the ISN to be used (from the counter) and makes an entry of this in the *ISN* and *send sequence variable* fields of the connection record used for this connection. It then sends a segment to the TCP in the server with the SYN code bit on, the ACK bit off, and the chosen ISN (X) in the sequence (number) field. Note that since no window or MSS option fields are present, then the receiving TCP assumes the default values. The TCP entity then transfers to the SYN_SENT state.
- On receipt of the SYN, if the required server AP – as determined by the destination port and IP address – is not already in the LISTEN state, the server TCP declines the connection request by returning a segment with the RST code bit on. The client TCP entity then aborts the connection establishment procedure and returns an error message with a reason code to the client AP. Alternatively, if the server AP is in the LISTEN state, the server TCP makes an entry of the ISN (to be used in the client–server direction and contained within the received segment) in its own connection record – in both the *ISN* and *receive sequence variable* fields – together with the ISN it proposes to use in the return direction. It then proceeds to create a new segment with the SYN bit on and the chosen ISN(Y) in the sequence field. In addition, it sets the ACK bit on

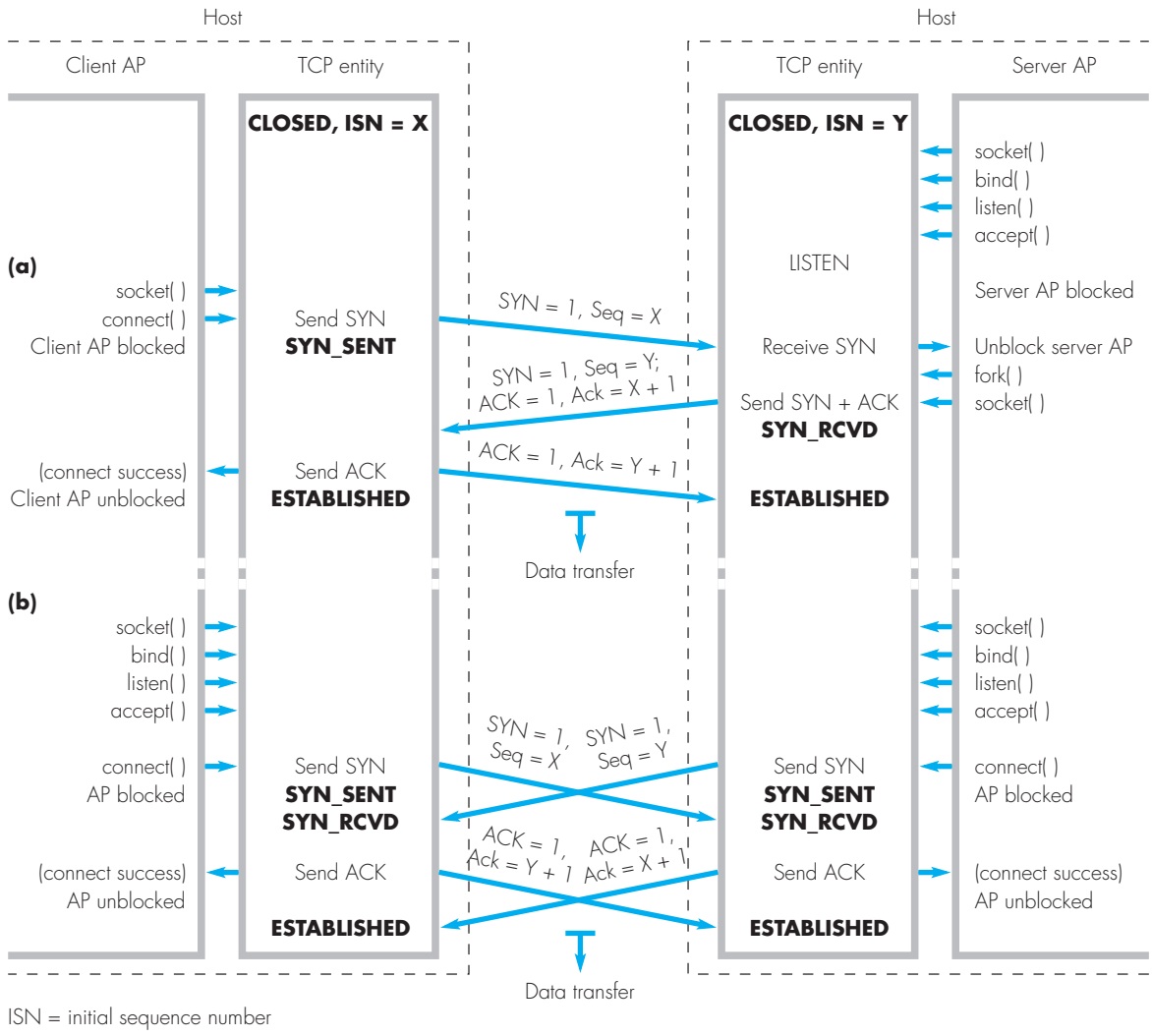


Figure 7.5 TCP connection establishment examples: (a) client–server; (b) connection collision.

and returns a value of X+1 in the acknowledgment field to acknowledge receipt of the client's SYN. It then sends the segment to the client and enters the SYN_RCVD state.

- On receipt of the segment, the client TCP makes an entry of the ISN to be used in the server–client direction in its connection record – in both the *ISN* and *receive sequence variable* fields – and increments the send sequence variable in the record by 1 to indicate the SYN has been acknowledged. It then acknowledges receipt of the SYN by returning a

segment with the ACK bit on and a value of Y+1 in the acknowledgment field. The TCP entity then enters the (connection) ESTABLISHED state.

- On receipt of the ACK, the server TCP increments the send sequence variable in its own connection record by 1 and enters the ESTABLISHED state. As we can deduce from this, the acknowledgment of each SYN segment is equivalent to a single data byte being transferred in each direction. At this point, both sides are in the ESTABLISHED state and ready to exchange data segments.

Although in a client–server application the client always initiates the setting up of a connection, in applications not based on the client–server model the two APs may try to establish a connection at the same time. This is called a **simultaneous open** and the sequence of segments exchanged in this case is as shown in Figure 7.5(b).

As we can see, the segments exchanged are similar to those in the client–server case and, since both ISNs are different, each side simply returns a segment acknowledging the appropriate sequence number. However, since the connection identifier is the same in both cases, only a single connection is established.

Data transfer

The error control procedure associated with the TCP protocol is similar to that used with the HDLC protocol. The main difference is that the sequence and acknowledgment numbers used with TCP relate to individual bytes in the total byte stream whereas with HDLC the corresponding send and receive sequence numbers relate to individual blocks of data. Also, because of the much larger round-trip time of an internet (compared with a single link), with TCP the window size associated with the flow control procedure is not derived from the sequence numbers. Instead, a new window size value is included in each segment a TCP entity sends to inform the other TCP entity of the maximum number of bytes it is willing to receive from it. This is known also as a **window size advertisement**.

In addition, because the TCP protocol may be operating (on an end-to-end basis) over a number of interconnected networks rather than a single line, it includes a congestion control procedure. This endeavors to regulate the rate at which the sending TCP entity sends data segments into the internet to the rate segments are leaving the internet. We shall discuss the main features of the different procedures that are used by means of examples.

Small segments

In order to explain the features of the protocol that relate to the exchange of small segments – that is, all the segments contain less than the MSS – we shall consider a typical data exchange relating to a networked interactive application. An example application protocol of this type is Telnet. Typically, this involves a user at the client side typing a command and the server AP in a remote host responding to it. An example set of segments is shown in Figure 7.6(a).

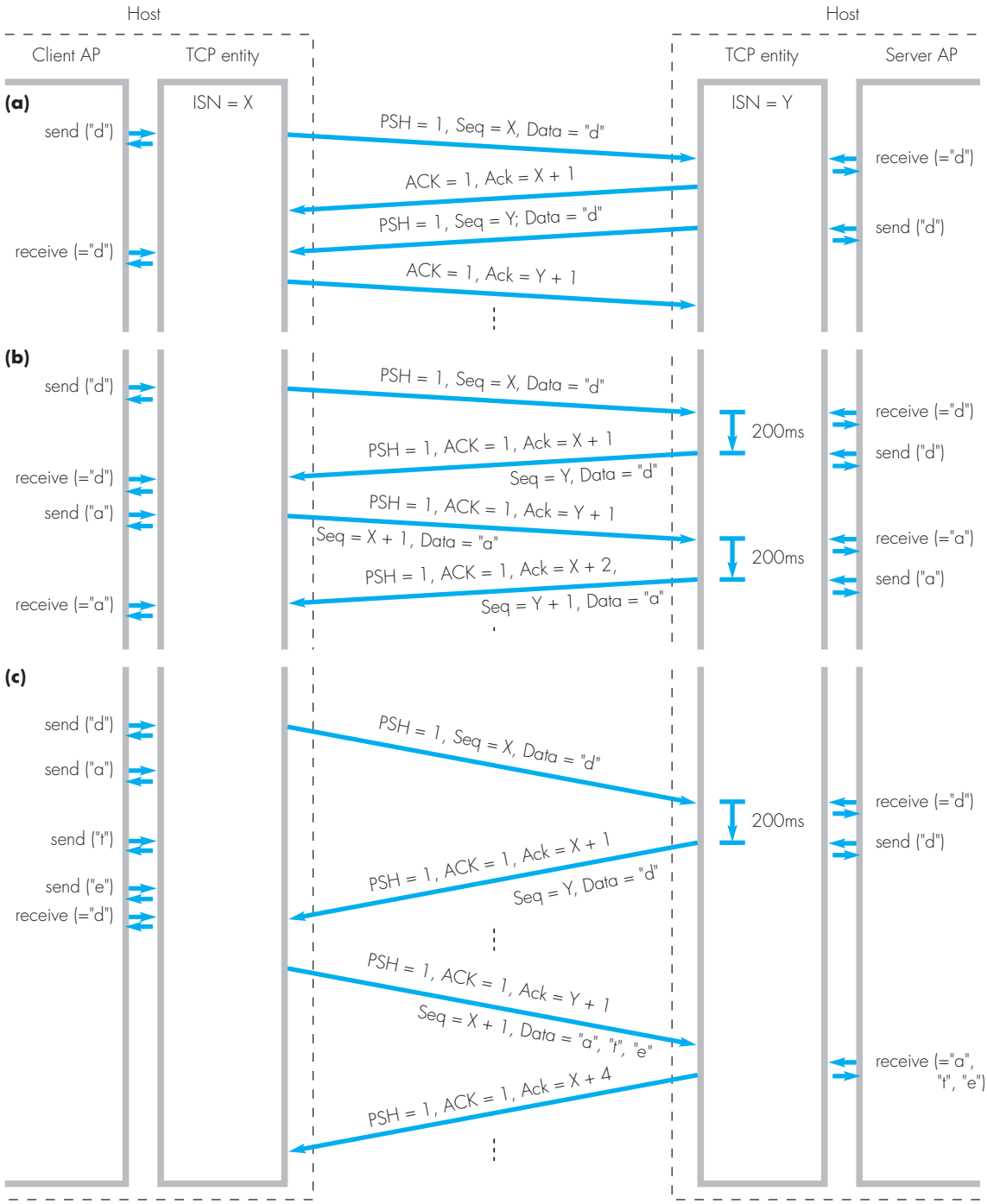


Figure 7.6 Small segment data transfers: (a) immediate acknowledgments; (b) delayed acknowledgments; (c) Nagle algorithm.

With interactive applications involving a user at a keyboard, each character typed is sent directly by the client AP to the server side. The server AP then reads the character from the receive buffer and immediately echoes the character back to the client side by writing it into the send buffer. On receipt of the character, the client AP displays it on the host screen. Hence each typed character is sent directly in a separate segment with the PSH flag on. Similarly, the echoed character is also sent in a separate segment with the PSH flag on. In addition to these two segments, however, each TCP entity returns a segment with the ACK bit on to acknowledge receipt of the segment containing the typed/echoed character. This means that for each character that is typed, four segments are sent, each with a 20-byte header and a further 20-byte IP header.

In practice, in order to reduce the number of segments that are sent, a receiving TCP entity does not return an ACK segment immediately it receives an (error-free) data segment. Instead, it waits for up to 200 ms to see if any data is placed in the send buffer by the local AP. If it is, then the acknowledgment is piggybacked in the segment that is used to send the data. This procedure is called **delayed acknowledgments** and, as we can see in Figure 7.6(b), with interactive applications it can reduce significantly the number of segments that are required.

Although this mode of working is acceptable when communicating over a single network such as a LAN, when communicating over an internet which has a long round-trip time (RTT), the delays involved in waiting for each echoed character can be annoying to the user. Hence a variation of the basic delayed acknowledgment procedure is often used. This is called the **Nagle algorithm** and is defined in RFC 896. When the algorithm is enabled, each TCP entity can have only a single small segment waiting to be acknowledged at a time. As a result, in interactive applications, when the client TCP entity is waiting for the ACK for this segment to be received, a number of characters may have been typed by the user. Hence when the ACK arrives, all the waiting characters in the send buffer are transmitted in a single segment. A sequence diagram showing this is given in Figure 7.6(c).

In these examples, the window size has not been shown since, in general, when small segments are being exchanged it has no effect on the flow of the segments. Also, the Nagle algorithm is not always enabled. For example, when the interactions involve a mouse, each segment may contain a collection of mouse movement data and, when echoed, the movement of the cursor can appear erratic. An example application of this type is X-Windows.

Error control

As we saw in Figure 7.4(a), each TCP segment contains only a single acknowledgment number. Hence should a segment not arrive at the destination host, the receiving TCP can only return an acknowledgment indicating the next in-sequence byte that it expects. Also, since the packets relating to a message are being transmitted over an internet, when the path followed has alternative routes, packets may arrive at the destination host out of sequence. Hence the

receiving TCP simply holds each out-of-sequence segment that it receives in temporary storage and returns an ACK indicating the next in-sequence byte – and hence segment – that it expects. Normally, the out-of-sequence segment arrives within a short time interval at which point the receiving TCP returns an ACK that acknowledges all the segments now received including those held in temporary storage.

At the sending side, the TCP receives an ACK indicating a segment has been lost but, within a short time interval, it receives an ACK for a segment that it transmitted later (so acknowledging receipt of all bytes up to and including the last byte in the later segment). Hence, since this is a relatively frequent occurrence, the sending TCP does not initiate a retransmission immediately it receives an out-of-sequence ACK. Instead, it only retransmits a segment if it receives three duplicate ACKs for the same segment – that is, three consecutive segments with the same acknowledgment number in their header – since it is then confident that the segment has been lost rather than simply received out of sequence.

In addition, to allow for the possibility that the sending TCP has no further segments ready for transmission, when a loss is detected it starts a *retransmission timer* for each new segment it transmits. A segment is then retransmitted if the TCP does not receive an acknowledgment for it before the timeout interval expires. An example illustrating the two possibilities is shown in Figure 7.7. In the example we assume:

- there is only a unidirectional flow of data segments;
- the sending AP writes a block of data – a message – comprising 3072 bytes into the send buffer using a *send( )* primitive;
- the MSS being used for the connection is 512 bytes and hence six segments are required to send the block of data;
- the size of the receive buffer is 8192 bytes and hence the sending TCP can send the complete set of segments without waiting for an acknowledgment;
- an ACK segment is returned on receipt of each error-free data segment;
- segments 2 and 6 are both lost – owing to transmission errors for example – as is the final ACK segment.

To follow the transmission sequence shown in the figure we should note the following:

- The sending TCP has a send sequence variable, V(S), in its connection record, which is the value it places in the sequence number field of the next new segment it sends. Also it has a *retransmission list* to hold segments waiting to be acknowledged. Similarly, the receiving TCP has a receive sequence variable, V(R), in its connection record (which is the sequence number it expects to receive next) and a *receive list* to hold segments that are received out of sequence.

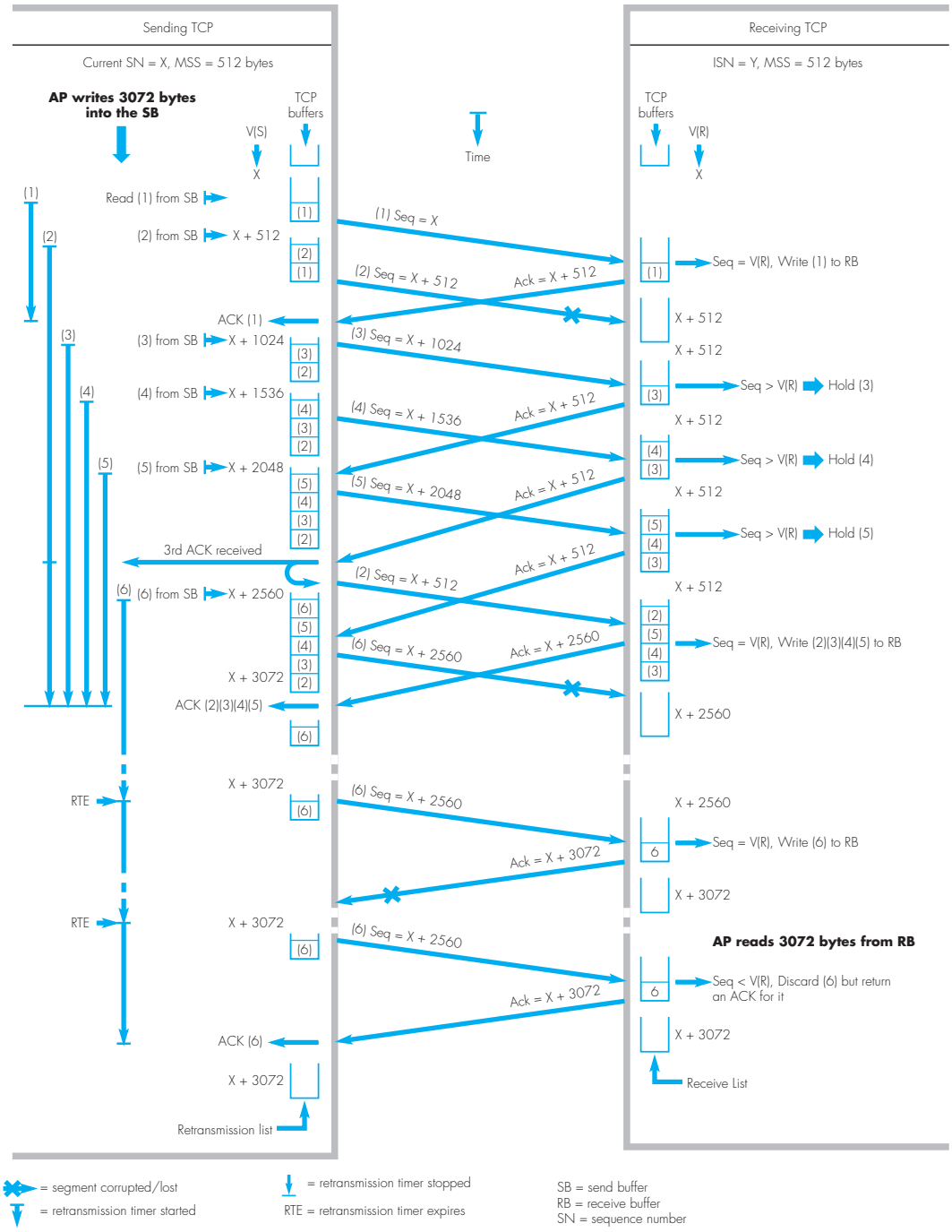


Figure 7.7 Example segment sequence showing TCP error control procedure.

- Segment (1) is received error-free and, since its sequence number (Seq = X) is equal to V(R), the 512 bytes of data it contains are transferred directly into the receive buffer (ready for reading by the destination AP), V(R) is incremented to X + 512, and an ACK (with Ack = X + 512) is returned to the sending side.
- On receipt of the ACK, the sending TCP stops the retransmission timer for (1) and, in the meantime, segment (2) has been sent.
- Since segment (2) is corrupted, no ACK for it is returned and hence its timer continues running.
- The sending TCP continues to send segments (3), (4) and (5), all of which are received error-free. However, since they are out of sequence – segment (2) is missing – the receiving TCP retains them in its receive list and returns an ACK with Ack = X + 512 in each segment for each of them to indicate to the sending TCP that segment (2) is missing.
- On receipt of the third ACK with an ACK = X + 512, the sending TCP retransmits segment (2) without waiting for the retransmission timer to expire. As we indicated earlier, this is done since if three or more ACKs with the same acknowledgment number are received one after the other, it is assumed that the segment indicated has been lost rather than received out of sequence. Because the retransmission occurs before the timer expires, this procedure is called **fast retransmit**.
- This time segment (2) is received error-free and, as a result, the receiving TCP is able to transfer the contents of segments (2), (3), (4) and (5) to the receive buffer – ready for reading by the AP – and returns an ACK with Ack = X + 2560 to indicate to the sending TCP that all bytes up to and including byte 2560 have been received and their timers can be stopped.
- In the meantime, segment (6) has been transmitted but is corrupted. Hence, since no other segments are waiting to be sent, its timer continues running until it expires when the segment is retransmitted.
- This time the segment is received error-free and so its contents are passed to the receive buffer directly and an ACK for it is returned. Also, it is assumed that at this point the receiving AP reads the accumulated 3072 bytes from the receive buffer using a *receive( )* primitive.
- The ACK for segment (6) is corrupted and hence the timer for the segment expires again and the segment is retransmitted. However, since the sequence number is less than the current V(R), the receiving TCP assumes it is a duplicate. Hence it discards it but returns an ACK to stop the sending TCP from sending another copy.

As we can see from this example, a key parameter in the efficiency of the error control procedure is the choice of the **retransmission timeout** (**RTO**) **interval**. With a single data link the choice of an RTO is straightforward since the worst-case round-trip time – the time interval between sending a packet/frame and receiving an ACK for it – can be readily determined. The

RTO is then set at a value slightly greater than this. With an internet, however, the RTT of a TCP connection can vary considerably over relatively short intervals as the traffic levels within routers build up and subside. Hence choosing an RTO when the internet is lightly loaded can lead to a significant number of unnecessary retransmissions, while choosing it during heavy load conditions can lead to unnecessary long delays each time a segment is corrupted/lost. The choice of RTO, therefore, must be dynamic and vary not only from one connection to another but also during a connection.

The initial approach used to derive the RTO for a connection was based on an **exponential backoff** algorithm. With this an initial RTO of 1.5 seconds is used. Should this prove to be too short – that is, each segment requires multiple retransmission attempts – the RTO is doubled to 3 seconds. This doubling process then continues until no retransmissions occur. To allow for network problems, a maximum RTO of 2 minutes is used at which point a segment with the RST flag bit on is sent to abort the connection/session.

Although very simple to implement, a problem with this method is that when an ACK is received, it is not clear whether this is for the last retransmission attempt or an earlier attempt. This is known as the **retransmission ambiguity problem** and was identified by Karn. Because of this, a second approach was proposed by Jacobson and is defined in RFC 793. With this method, the RTO is computed from actual RTT measurements. The RTO is then updated as each new RTT measurement is made. In this way, the RTO for each connection is continuously being updated as each new estimate of the RTT is determined.

As we indicated earlier, when each data segment is sent, a separate retransmission timer is started. A connection starts with a relatively large RTO. Then, each time an ACK segment is received before the timer expires, the actual RTT is determined by subtracting the initial start time of the timer from the time when the ACK was received. The current estimate of the RTT is then updated using the formula

$$\text{RTT} = \alpha\,\text{RTT} + (1-\alpha)\,M$$

where M is the measured RTT and α is a smoothing factor that determines by how much each new M influences the computation of the new RTT relative to the current value. The recommended value for α is 0.9. The new RTO is then set at twice the updated RTT to allow for a degree of variance in the RTT.

Although this method performed better than the original method, a refinement of it was later introduced. This was done because by using a fixed multiple of each updated RTT ($\times 2$) to compute the new RTO, it was found that it did not handle well the wide variations that occurred in the RTT. Hence in order to obtain a better estimate, Jacobson proposed that each new RTO should be based not just on the mean of the measured RTT but also on the variance. In the proposed algorithm, the mean deviation of the RTT measurements, D, is used as an estimate of the variance. It is computed using the formula:

$$D = \alpha\,D + (1-\alpha)|\text{RTT} - M|$$

where α is a smoothing factor and $|RTT - M|$ is the modulus of the difference between the current estimate of the RTT and the new measured RTT (M). This is also computed for each new RTT measurement and the new estimate of the RTO is then derived using the formula:

$$RTO = RTT + 4D$$

As we indicated earlier, to overcome the retransmission ambiguity problem, the RTT is not measured/updated for the ACKs relating to retransmitted segments.

Note also that although in the above example an ACK is returned on receipt of each data segment, this is not always the case. Indeed, in most implementations, providing there is a steady flow of data segments, the receiving TCP only returns an ACK for every other segment it receives correctly. On sending each ACK, a timer – called the **delayed ACK timer** – is started and, should a second segment not be received before it expires, then an ACK for the first segment is sent. Note, however, that when a single ACK is sent for every other segment, since the V(R) is incremented on receipt of each segment, then the acknowledgment number within the (ACK) segment acknowledges the receipt of all the bytes in both segments.

Flow control

As we indicated earlier, the value in the *window size* field of each segment relates to a sliding window flow control scheme and indicates the number of bytes (relative to the byte being acknowledged in the *acknowledgment* field) that the receiving TCP is able to accept. This is determined by the amount of free space that is present in the receive buffer being used by the receiving TCP for the connection. Recall also that the maximum size of the window is determined by the size of the receive buffer. Hence when the sending TCP is running in a fast host – a large server for example – and the receiving TCP in a slow host, the sending TCP can send segments faster than, firstly, the receiving TCP can process them and, secondly, the receiving AP can read them from the receive buffer after they have been processed. The window flow control scheme, therefore, is present to ensure that there is always the required amount of free space in the receive buffer at the destination before the source sends the data. An example showing the sequence of segments that are exchanged to implement the scheme is given in Figure 7.8. In the example, we assume:

- there is only a unidirectional flow of data segments;
- the sizes of both the send and the receive buffers at the sending side are 4096 bytes and those at the receiving side 2048 bytes. Hence the maximum size of the window for the direction of flow shown is 2048 bytes;
- associated with each direction of flow the sending side maintains a *send window variable*, W_S, and the receiving side a *receive window variable*, W_R;

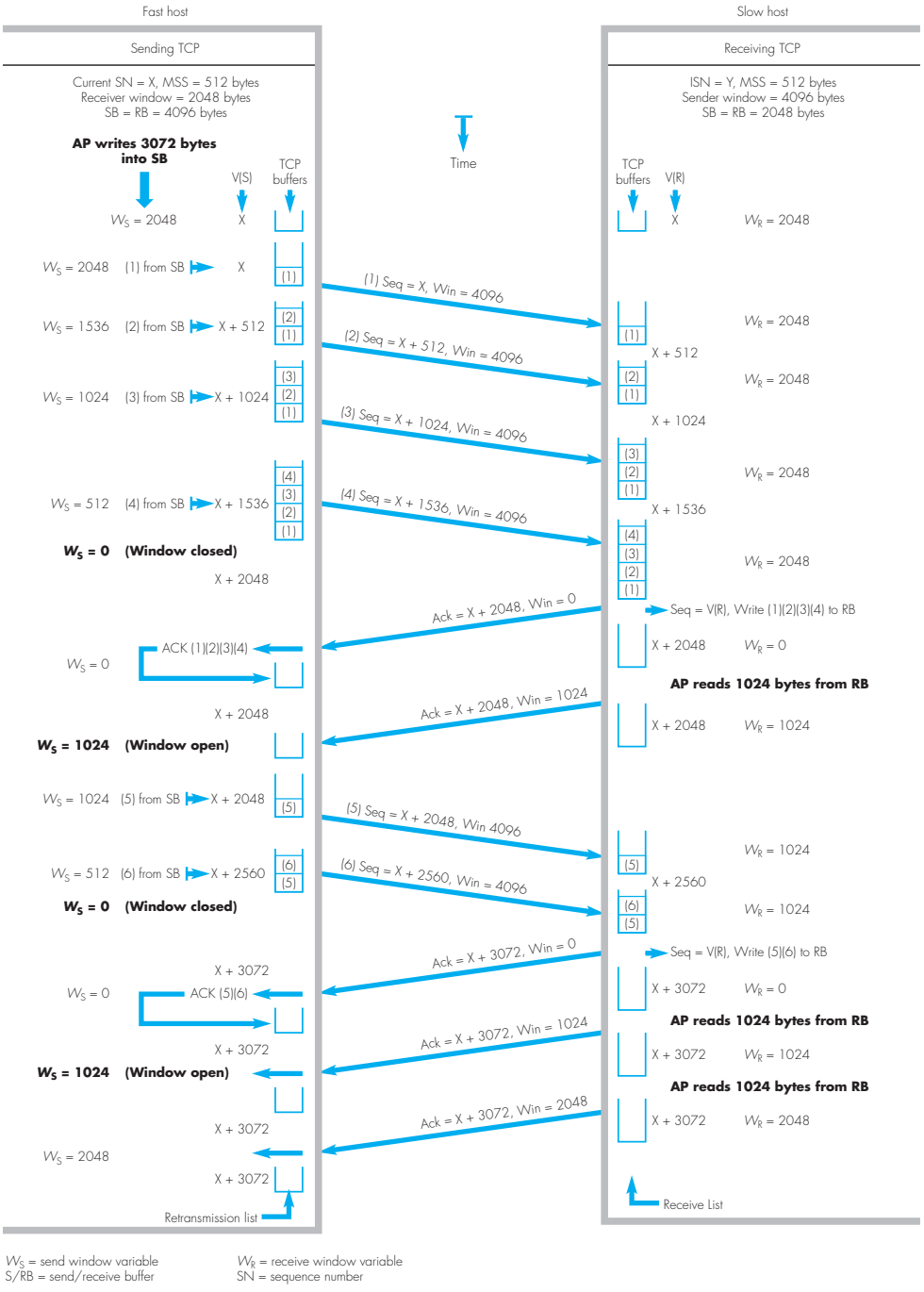


Figure 7.8 Example segment sequence showing TCP flow control procedure.

- the source AP can write bytes into the send buffer up to the current value of W_S and, providing W_S is greater than zero, the sending TCP can read data bytes from its send buffer up to the current value of W_S and initiate their transmission. Flow is stopped when $W_S = 0$ and the window is then said to be closed;
- at the destination, the receiving TCP, on receipt of error-free data segments, transfers the data they contain to the receive buffer and increments W_R by the number of bytes transferred. W_R is then decremented when the destination AP reads bytes from the receive buffer and, after each read operation, the receiving TCP returns a segment with the number of bytes of free space now available in the window size field of the segment.

The following should be noted when interpreting the sequence:

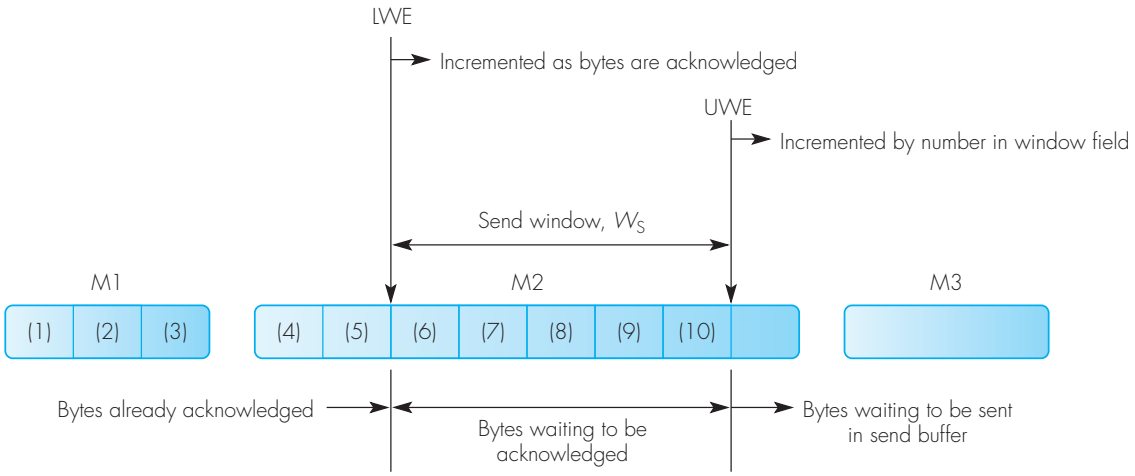
- The flow of segments starts with the AP at the sending side writing 3072 bytes into the send buffer using a *send( )* primitive.
- Since the sending host is much faster than the receiving host, the sending TCP is able to send a full window of 2048 bytes (in four 512-byte segments) before the receiving TCP is able to start processing them. The sending TCP must then stop as W_S is now zero.
- When the receiving TCP is scheduled to run, it finds four segments in its receive list and, since the first segment – segment (1) – has a sequence number equal to V(R), it transfers its contents to the receive buffer. It then proceeds to process and transfer the contents of segments (2), (3) and (4) in the same way and, after it has done this, it returns a single ACK segment to acknowledge receipt of these four segments but with a window size field of zero.
- On receipt of the ACK, the sending TCP deletes segments (1), (2), (3) and (4) from its retransmission list but leaves $W_S = 0$.
- When the receiving AP is next scheduled to run, we assume it reads just 1024 bytes from the receive buffer. On detecting this, the TCP returns a second ACK with the same acknowledgment number but with a window size of 1024. The second ACK is known, therefore, as a **window update**.
- At this point, since its sending window is now open, the sending TCP proceeds to send segments (5) and (6) at which point W_S again becomes zero.
- At the receiving side, when the TCP is next scheduled to run it finds segments (5) and (6) in the receive list and, since their sequence numbers are in sequence, it transfers their contents to the receive buffer. It then returns a single ACK for them but with a window size of zero.
- On receipt of the ACK, the sending TCP deletes segments (5) and (6) from its retransmission list but leaves $W_S = 0$.

- At some point later, the receiving AP is scheduled to run and we assume it again reads 1024 bytes from the receive buffer. Hence when the TCP next runs it returns a window update of 1024.
- Finally, after the receiving AP reads the last 1024 bytes from the RB, the TCP – some time later – returns a window update of 2048. After this has been received, both sides are back to their initial state.

We should note that there are a number of different implementations of TCP and hence the sequence shown in Figure 7.8 is only an example. For example, the receiving TCP may return two ACKs – one for segments (1) and (2) and the other for segments (3) and (4) – rather than a single ACK. In this case there would be a different distribution of segments between the TCP buffers and the receive buffer. Nevertheless, providing the size of the TCP buffers in the receiver is the same as the receive buffer, then the window procedure ensures there is sufficient buffer storage to hold all received segments. A schematic diagram summarizing the operation of the sliding window procedure is given in Figure 7.9.

Congestion control

A segment may be discarded during its transfer across an internet either because transmission errors are detected in the packet containing the



LWE = lower window edge, initialized to (ISN + 1)
UWE = upper window edge, initialized to (ISN + 1) + advertised window
W_S = (UWE – LWE), flow stopped when W_S = 0
M1, 2, 3 = sending AP messages
(1), (2) etc. = segments sent by TCP entity
Note: TCP chooses the size of segments it sends

Figure 7.9 TCP sliding window.

segment or because a router or gateway along the path being followed becomes congested; that is, during periods of heavy traffic it temporarily runs out of buffer storage for packets in the output queue associated with a line. However, the extensive use of optical fiber in the transmission network means that the number of lost packets due to transmission errors is relatively small. Hence the main reason for lost packets is congestion within the internet.

To understand the reason for congestion, it should be remembered that the path followed through an internet may involve a variety of different transmission lines some of which are faster – have a higher bit rate – than others. In general, therefore, the overall speed of transmission of segments over the path being followed is determined by the bit rate of the slowest line. Also, congestion can arise at the sending side of this line as the segments relating to multiple concurrent connections arrive at a faster rate than the line can transmit them. Clearly, if this situation continues for even a relatively short time interval, the number of packets in the affected router output queue builds up until the queue becomes full and packets have to be dropped. This also affects the ACKs within the lost segments and, as we have just seen, this can have a significant effect on the overall time that is taken to transmit a message.

In order to reduce the likelihood of lost packets occurring, the TCP in each host has a congestion control/avoidance procedure that, for each connection, uses the rate of arrival of the ACKs relating to a connection to regulate the rate of entry of data segments – and hence IP packets – into the internet. This is in addition to the window flow control procedure, which, as we have just seen, is concerned with controlling the rate of transmission of segments to the current capacity of the receive buffer in the destination host. Hence in addition to a send window variable, W_S, associated with the flow control procedure, each TCP also has a **congestion window** variable, W_C, associated with the congestion control/avoidance procedure. Both are maintained for each connection and the transmission of a segment relating to a connection can only take place if both windows are in the open state.

As we can see from the above, under lightly loaded network conditions the flow of segments is controlled primarily by W_S and, under heavily loaded conditions, it is controlled primarily by W_C. However, when the flow of segments relating to a connection first starts, because no ACKs have been received, the sending TCP does not know the current loading of the internet. So to stop it from sending a large block of segments – up to the agreed window size – the initial size of the congestion window, W_C, is set to a single segment which, because W_C has a dimension of bytes, is equal to the agreed MSS for the connection.

As we show in Figure 7.10, the sending TCP starts the data transfer phase of a connection by sending a single segment of up to the MSS. It then starts the retransmission timer for the segment and waits for the ACK to be received. If the timer expires, the segment is simply retransmitted. If the ACK is received before the timer expires, W_C is increased to two segments, each equal to the MSS. The sending TCP is then able to send two segments and,

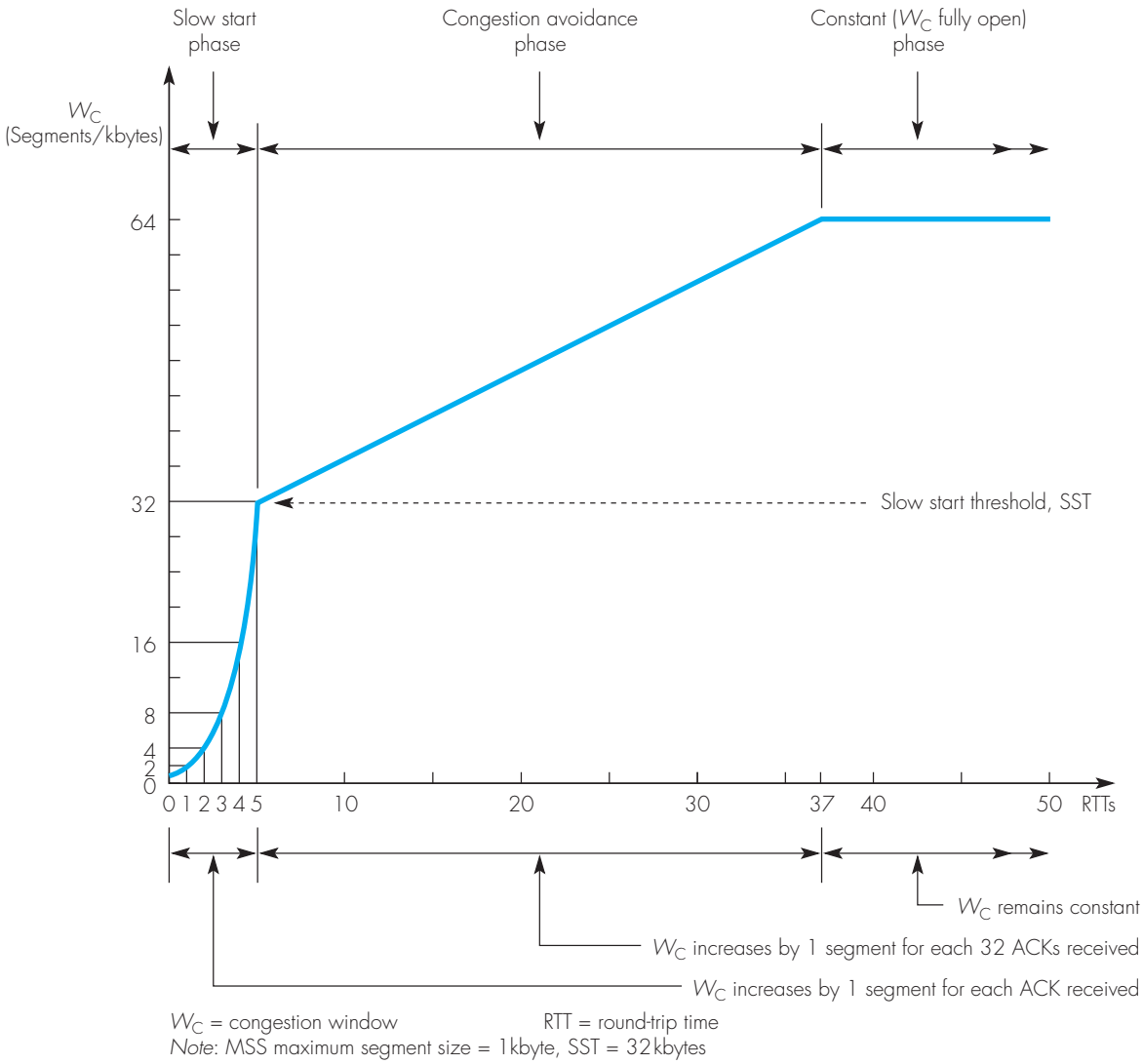


Figure 7.10 TCP congestion control window procedure.

for each of the ACKs it receives for these segments, W_C is increased by one segment (MSS). Hence, the sending TCP can now send four segments and, as we can see, W_C increases exponentially. Even though W_C increases rapidly, this phase is called **slow start** since it builds up from a single segment. It continues until a timeout for a lost segment occurs, or a duplicate ACK is received, or an upper threshold is reached. This is called the **slow start threshold** (**SST**) and, for each new connection, it is set to 64 kbytes. In the example, however, it is assumed to be initialized to 32 kbytes, which, because

the MSS is 1 kbyte, is equal to 32 segments. Assuming the SST is reached, this is taken as an indication that the path is not congested. Hence the connection enters a second phase during which, instead of W_C increasing by 1 segment (MSS) for each ACK it receives, it increases by $1/W_C$ segments for each ACK received. Hence, as we can see, W_C now increases by 1 segment for each set of W_C ACKs that are received. This is called the **congestion avoidance** phase and, during this phase, the increase in W_C is additive. It continues until a second threshold is reached and, in the example, this is set at 64 kbytes. On reaching this, W_C remains constant at this value.

The profile shown in Figure 7.10 is typical of a lightly loaded internet in which none of the lines making up the path through the internet is congested. Providing W_C remains greater than the maximum flow control window, the flow of segments relating to the connection is controlled primarily by W_S. During these conditions all segments are transferred with a relatively constant transfer delay and delay variation. As the number of connections using the internet increases, however, so the traffic level increases up to the point at which packet (and hence segment) losses start to occur and, when this happens, the TCP controlling each connection starts to adjust its congestion window in a way that reflects the level of congestion.

The steps taken depend on whether a lost packet is followed by duplicate ACKs being received or the retransmission timer for the segment expiring. In the case of the former, as we saw earlier in Figure 7.7, the receipt of duplicate ACKs is indicative that segments are still being received by the destination host. Hence the level of congestion is assumed to be light and, on receipt of the third duplicate ACK relating to the missing segment – fast retransmit – the current W_C value is halved and the congestion avoidance procedure is invoked starting at this value. This is called **fast recovery** and an example is shown in Figure 7.11(a).

In this example it is assumed that the first packet loss occurs when W_C is at its maximum value of 64 segments, which, with an MSS of 1 kbyte, is equal to 64 kbytes. Hence on receipt of the third duplicate ACK, the lost segment is retransmitted and W_C is immediately reset to 32 segments/kbytes. The W_C is then incremented back up using the congestion avoidance procedure. However, when it reaches 34 segments, a second segment is lost. It is assumed that this also is detected by the receipt of duplicate ACKs and hence W_C is reset to 17 segments before the congestion avoidance procedure is restarted.

In the case of a lost segment being detected by the retransmission timer expiring, it is assumed that the congestion has reached a level at which no packets/segments relating to the connection are now getting through. As we show in the example in Figure 7.11(b), when a retransmission timeout (RTO) occurs, irrespective of the current W_C, it is immediately reset to 1 segment and the slow start procedure is restarted. Thus, when the level of congestion reaches the point at which RTOs start to occur, the flow of segments is controlled primarily by W_C.

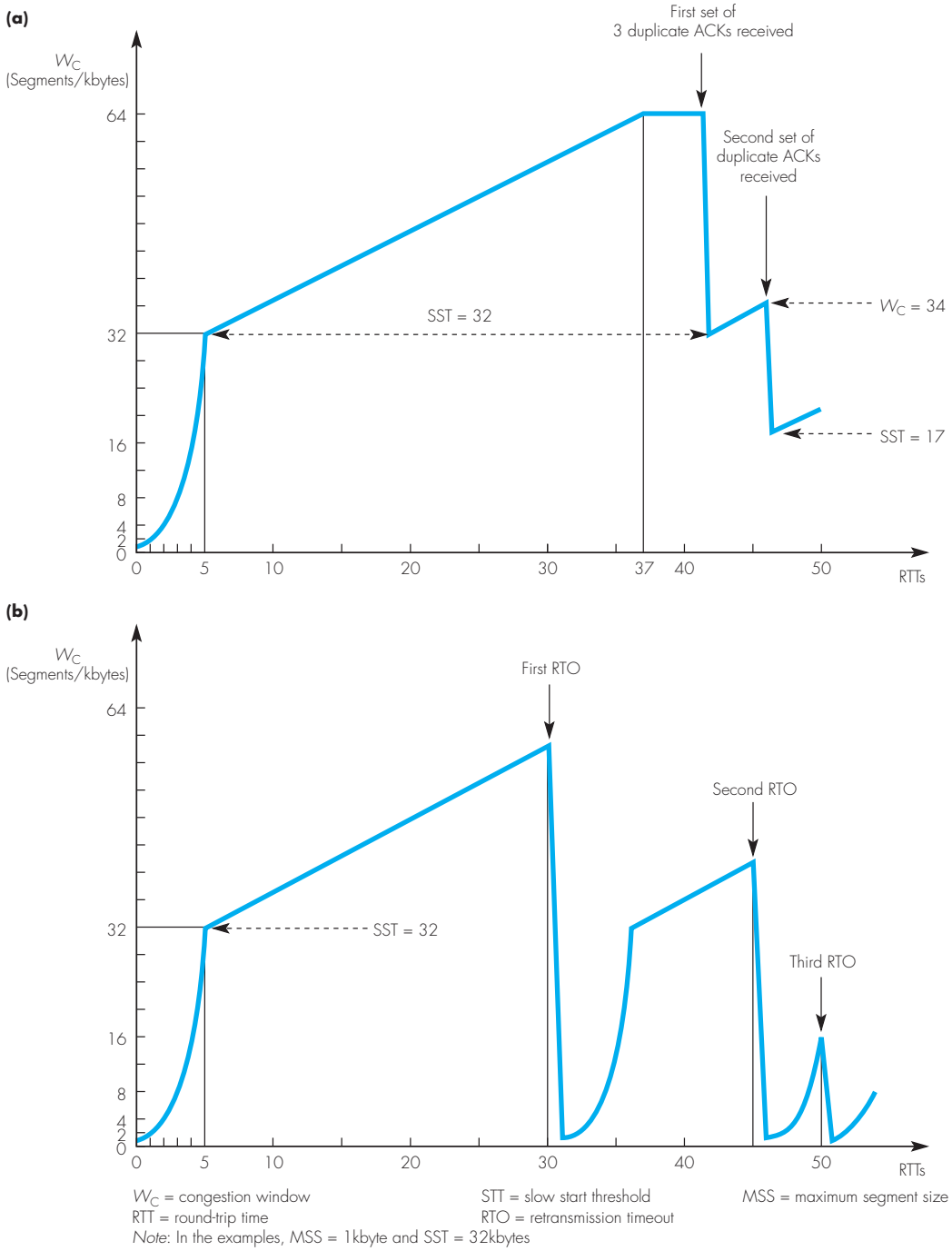
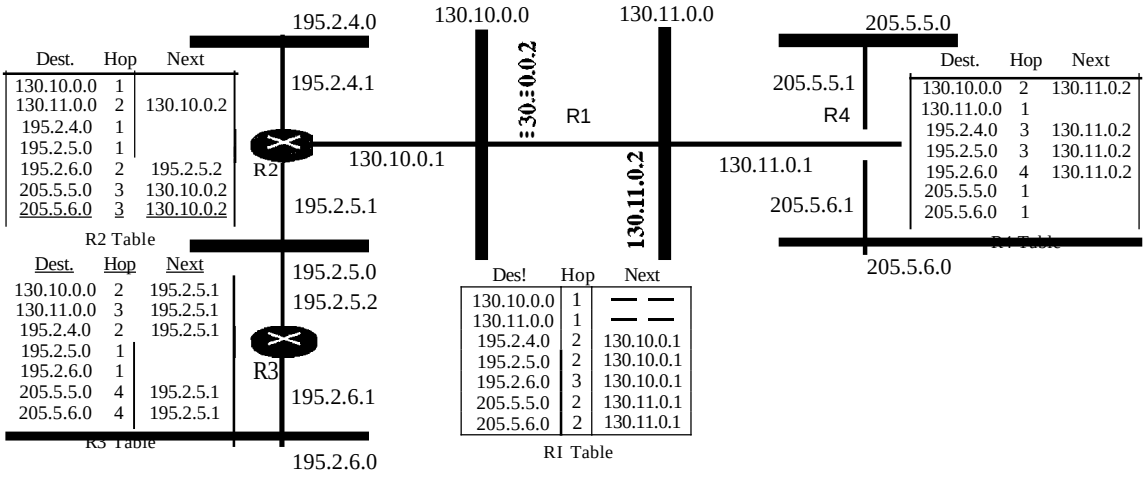


Figure 7.11 TCP congestion window adjustments: (a) on receipt of duplicate ACKs; (b) on expiry of a retransmission timer.

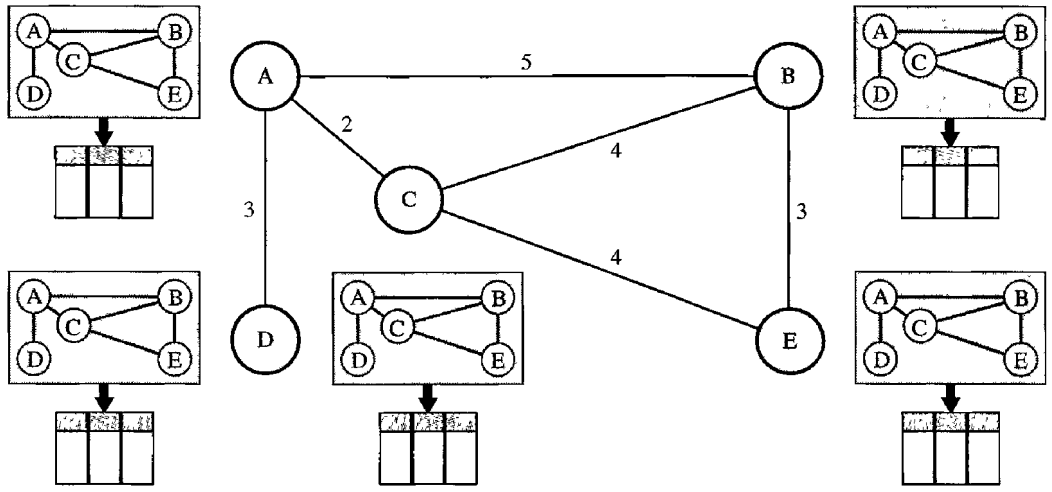
Figure 22.19 *Example of a domain using RIP*



Link State Routing

Link state routing has a different philosophy from that of distance vector routing. In link state routing, if each node in the domain has the entire topology of the domain-the list of nodes and links, how they are connected including the type, cost (metric), and condition of the links (up or down)-the node can use Dijkstra's algorithm to build a routing table. Figure 22.20 shows the concept.

Figure 22.20 *Concept of link state routing*

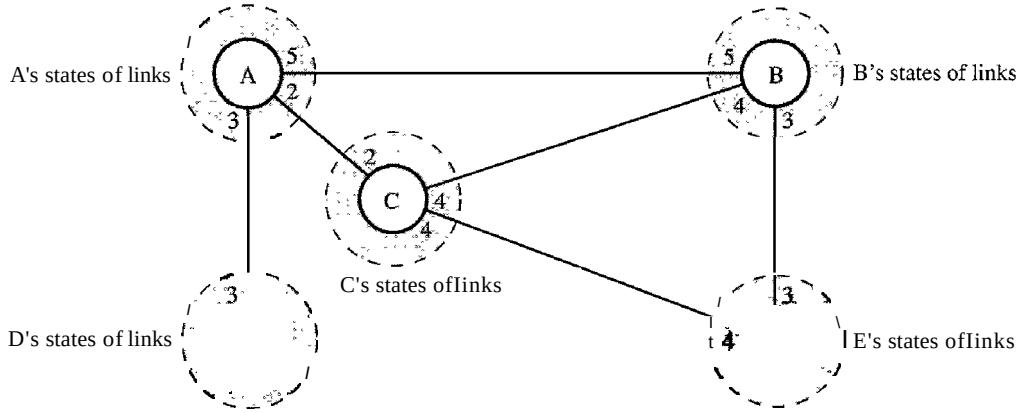


The figure shows a simple domain with five nodes. Each node uses the same topology to create a routing table, but the routing table for each node is unique because the calculations are based on different interpretations of the topology. This is analogous to a city map. While each person may have the same map, each needs to take a different route to reach her specific destination.

The topology must be dynamic, representing the latest state of each node and each link. If there are changes in any point in the network (a link is down, for example), the topology must be updated for each node.

How can a common topology be dynamic and stored in each node? No node can know the topology at the beginning or after a change somewhere in the network. Link state routing is based on the assumption that, although the global knowledge about the topology is not clear, each node has partial knowledge: it knows the state (type, condition, and cost) of its links. In other words, the whole topology can be compiled from the partial knowledge of each node. Figure 22.21 shows the same domain as in Figure 22.20, indicating the part of the knowledge belonging to each node.

Figure 22.21 *Link state knowledge*



Node A knows that it is connected to node B with metric 5, to node C with metric 2, and to node D with metric 3. Node C knows that it is connected to node A with metric 2, to node B with metric 4, and to node E with metric 4. Node D knows that it is connected only to node A with metric 3. And so on. Although there is an overlap in the knowledge, the overlap guarantees the creation of a common topology-a picture of the whole domain for each node.

Building Routing Tables

In link state routing, four sets of actions are required to ensure that each node has the routing table showing the least-cost node to every other node.

1. Creation of the states of the links by each node, called the link state packet (LSP).
2. Dissemination of LSPs to every other router, called **flooding,** in an efficient and reliable way.
3. Formation of a shortest path tree for each node.
4. Calculation of a routing table based on the shortest path tree.

Creation of Link State Packet (LSP) A link state packet can carry a large amount of information. For the moment, however, we assume that it carries a minimum amount

of data: the node identity, the list of links, a sequence number, and age. The first two, node identity and the list of links, are needed to make the topology. The third, sequence number, facilitates flooding and distinguishes new LSPs from old ones. The fourth, age, prevents old LSPs from remaining in the domain for a long time. LSPs are generated on two occasions:

1. *When there is a change in the topology of the domain.* Triggering of LSP dissemination is the main way of quickly informing any node in the domain to update its topology.
2. *On a periodic basis.* The period in this case is much longer compared to distance vector routing. As a matter of fact, there is no actual need for this type of LSP dissemination. It is done to ensure that old information is removed from the domain. The timer set for periodic dissemination is normally in the range of 60 min or 2 h based on the implementation. A longer period ensures that flooding does not create too much traffic on the network.

Flooding of LSPs After a node has prepared an LSP, it must be disseminated to all other nodes, not only to its neighbors. The process is called flooding and based on the following:

1. The creating node sends a copy of the LSP out of each interface.
2. A node that receives an LSP compares it 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, the node does the following:
   a. It discards the old LSP and keeps the new one.
   b. It 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 domain (where a node has only one interface).

Formation of Shortest Path Tree: Dijkstra Algorithm After receiving all LSPs, each node will have a copy of the whole topology. However, the topology is not sufficient to find the shortest path to every other node; a shortest path tree is needed.

A tree is a graph of nodes and links; one node is called the root. All other nodes can be reached from the root through only one single route. A shortest path tree is a tree in which the path between the root and every other node is the shortest. What we need for each node is a shortest path tree with that node as the root.

The Dijkstra algorithm creates a shortest path tree from a graph. The algorithm divides the nodes into two sets: tentative and permanent. It finds the neighbors of a current node, makes them tentative, examines them, and if they pass the criteria, makes them permanent. We can informally define the algorithm by using the flowchart in Figure 22.22.

Let us apply the algorithm to node A of our sample graph in Figure 22.23. To find the shortest path in each step, we need the cumulative cost from the root to each node, which is shown next to the node.

The following shows the steps. At the end of each step, we show the permanent (filled circles) and the tentative (open circles) nodes and lists with the cumulative costs.

Figure 22.22 *Dijkstra algorithm*

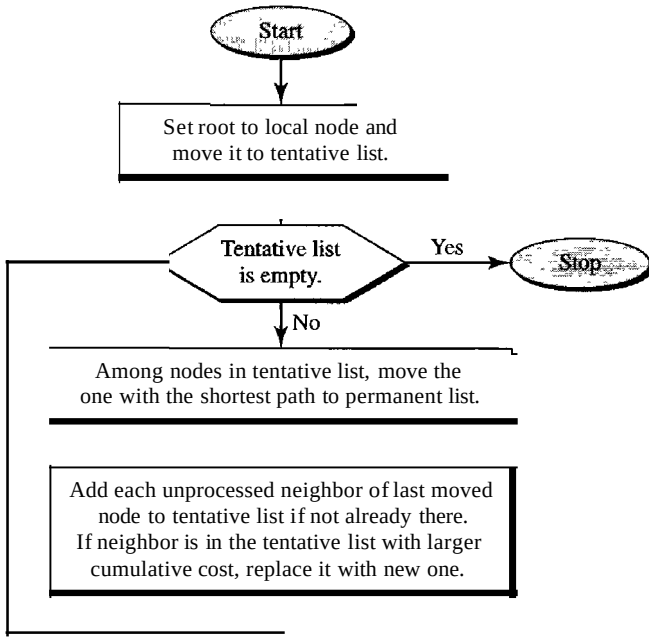
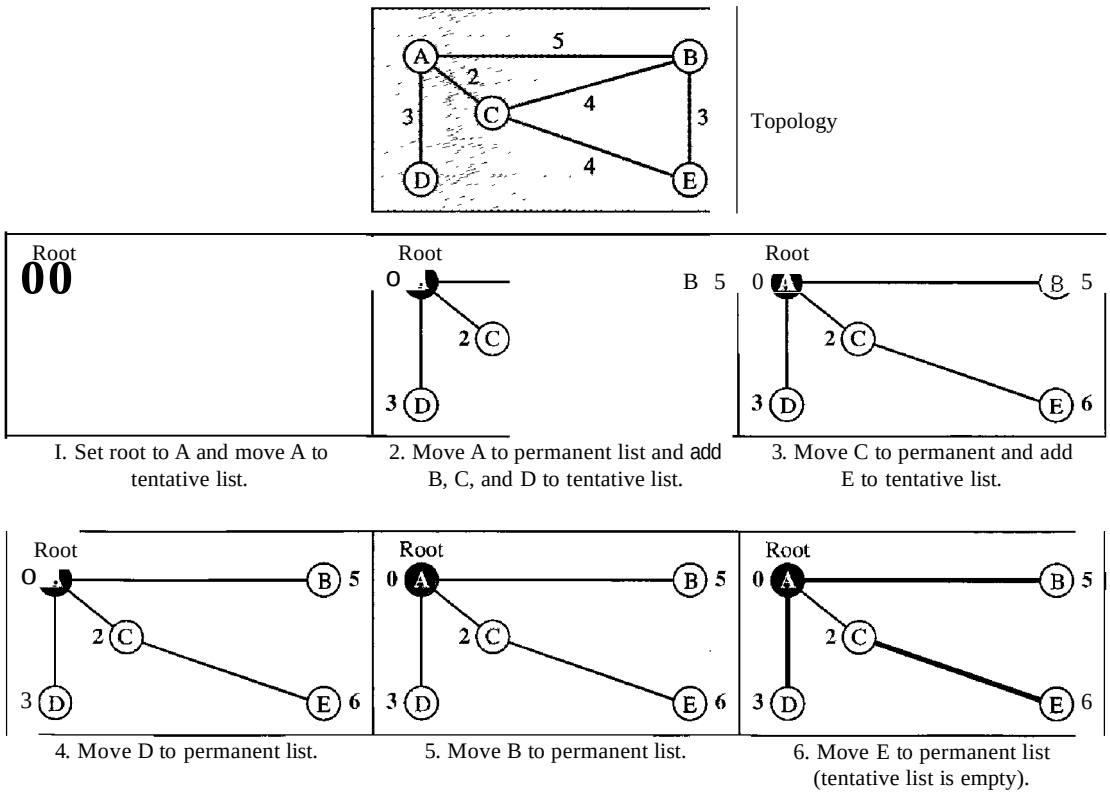


Figure 22.23 *Example of formation of shortest path tree*



1. We make node A the root of the tree and move it to the tentative list. Our two lists are

   Permanent list: empty Tentative list: A(O)

2. Node A has the shortest cumulative cost from all nodes in the tentative list. We move A to the pennanent list and add all neighbors of A to the tentative list. Our new lists are

   Permanent list: A(O) Tentative list: B(5), C(2), D(3)

3. Node C has the shortest cumulative cost from all nodes in the tentative list. We move C to the permanent list. Node C has three neighbors, but node A is already processed, which makes the unprocessed neighbors just B and E. However, B is already in the tentative list with a cumulative cost of 5. Node A could also reach node B through C with a cumulative cost of 6. Since 5 is less than 6, we keep node B with a cumulative cost of 5 in the tentative list and do not replace it. Our new lists are

   Permanent list: A(O), e(2) Tentative list: B(5), 0(3), E(6)

4. Node D has the shortest cumulative cost of all the nodes in the tentative list. We move D to the permanent list. Node D has no unprocessed neighbor to be added to the tentative list. Our new lists are

   Permanent list: A(O), C(2), 0(3) Tentative list: B(5), E(6)

5. Node B has the shortest cumulative cost of all the nodes in the tentative list. We move B to the permanent list. We need to add all unprocessed neighbors of B to the tentative list (this is just node E). However, E(6) is already in the list with a smaller cumulative cost. The cumulative cost to node E, as the neighbor of B, is 8. We keep node E(6) in the tentative list. Our new lists are

   Permanent list: A(O), B(5), C(2), 0(3) Tentative list: E(6)

6. Node E has the shortest cumulative cost from all nodes in the tentative list. We move E to the permanent list. Node E has no neighbor. Now the tentative list is empty. We stop; our shortest path tree is ready. The final lists are

   Permanent list: A(0), B(5), C(2), D(3), E(6) Tentative list: empty

Calculation of Routing Table from Shortest Path Tree Each node uses the shortest path tree protocol to construct its routing table. The routing table shows the cost of reaching each node from the root. Table 22.2 shows the routing table for node A.

Table 22.2 *Routing table for node A*

Node	*Cost*	*Next Router*
A	0	-
B	5	-
C	2	-
D	3	-
E	6	C

Compare Table 22.2 with the one in Figure 22.14. Both distance vector routing and link state routing end up with the same routing table for node A.

OSPF

The Open Shortest Path First or OSPF protocol is an intradomain routing protocol based on link state routing. Its domain is also an autonomous system.

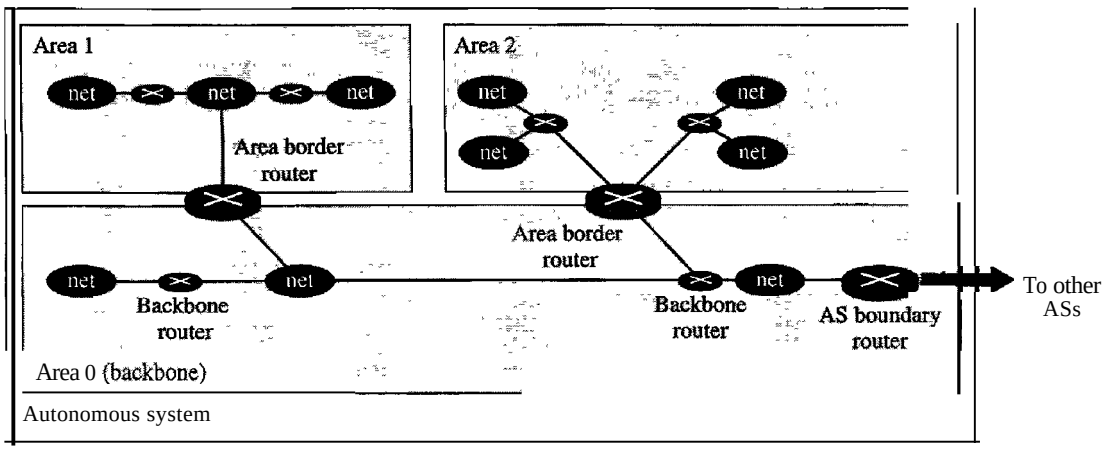
Areas To handle routing efficiently and in a timely manner, OSPF divides an autonomous system into areas. An area is a collection of networks, hosts, and routers all contained within an autonomous system. An autonomous system can be divided into many different areas. All networks inside an area must be connected.

Routers inside an area flood the area with routing information. At the border of an area, special routers called area border routers summarize the information about the area and send it to other areas. Among the areas inside an autonomous system is a special area called the *backbone;* all the areas inside an autonomous system must be connected to the backbone. In other words, the backbone serves as a primary area and the other areas as secondary areas. This does not mean that the routers within areas cannot be connected to each other, however. The routers inside the backbone are called the backbone routers. Note that a backbone router can also be an area border router.

If, because of some problem, the connectivity between a backbone and an area is broken, a virtual link between routers must be created by an administrator to allow continuity of the functions of the backbone as the primary area.

Each area has an area identification. The area identification of the backbone is zero. Figure 22.24 shows an autonomous system and its areas.

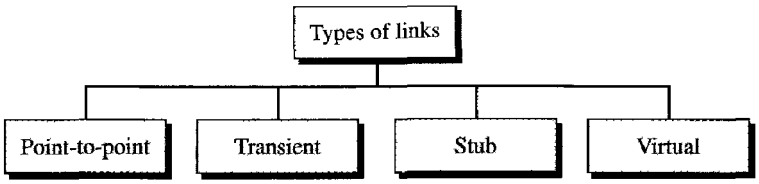
Figure 22.24 *Areas in an autonomous system*



Metric The OSPF protocol allows the administrator to assign a cost, called the metric, to each route. The metric can be based on a type of service (minimum delay, maximum throughput, and so on). As a matter of fact, a router can have multiple routing tables, each based on a different type of service.

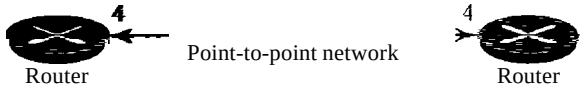
Types of Links In OSPF terminology, a connection is called a *link.* Four types of links have been defined: point-to-point, transient, stub, and virtual (see Figure 22.25).

Figure 22.25 *Types of links*



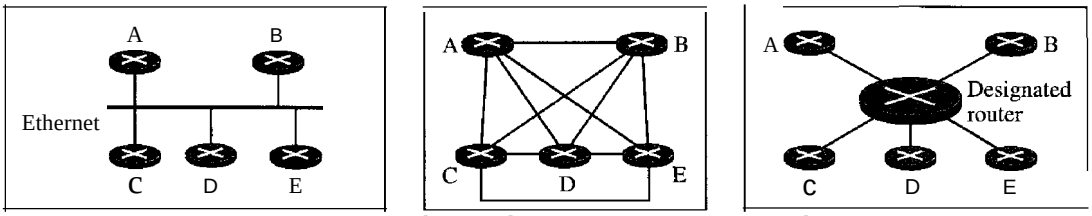
A point-to-point link connects two routers without any other host or router in between. In other words, the purpose of the link (network) is just to connect the two routers. An example of this type of link is two routers connected by a telephone line or a T line. There is no need to assign a network address to this type of link. Graphically, the routers are represented by nodes, and the link is represented by a bidirectional edge connecting the nodes. The metrics, which are usually the same, are shown at the two ends, one for each direction. In other words, each router has only one neighbor at the other side of the link (see Figure 22.26).

Figure 22.26 *Point-to-point link*



A transient link is a network with several routers attached to it. The data can enter through any of the routers and leave through any router. All LANs and some WANs with two or more routers are of this type. In this case, each router has many neighbors. For example, consider the Ethernet in Figure 22.27a. Router A has routers B, C, D, and E as neighbors. Router B has routers A, C, D, and E as neighbors. If we want to show the neighborhood relationship in this situation, we have the graph shown in Figure 22.27b.

Figure 22.27 *Transient link*



a. Transient network b. Unrealistic representation c. Realistic representation

This is neither efficient nor realistic. It is not efficient because each router needs to advertise the neighborhood to four other routers, for a total of 20 advertisements. It is

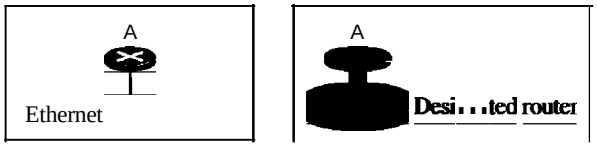
not realistic because there is no single network (link) between each pair of routers; there is only one network that serves as a crossroad between all five routers.

To show that each router is connected to every other router through one single network, the network itself is represented by a node. However, because a network is not a machine, it cannot function as a router. One of the routers in the network takes this responsibility. It is assigned a dual purpose; it is a true router and a designated router. We can use the topology shown in Figure 22.27c to show the connections of a transient network.

Now each router has only one neighbor, the designated router (network). On the other hand, the designated router (the network) has five neighbors. We see that thc number of neighbor announcements is reduced from 20 to 10. Still, the link is represented as a bidirectional edge between the nodes. However, while there is a metric from each node to the designated router, there is no metric from the designated router to any other node. The reason is that the designated router represents the network. We can only assign a cost to a packet that is passing through the network. We cannot charge for this twice. When a packet enters a network, we assign a cost; when a packet leaves the network to go to the router, there is no charge.

A **stub link** is a network that is connected to only one router. The data packets enter the network through this single router and leave the network through this same router. This is a special case of the transient network. We can show this situation using the router as a node and using the designated router for the network. However, the link is only one-directional, from the router to the network (see Figure 22.28).

Figure 22.28 *Stub link*



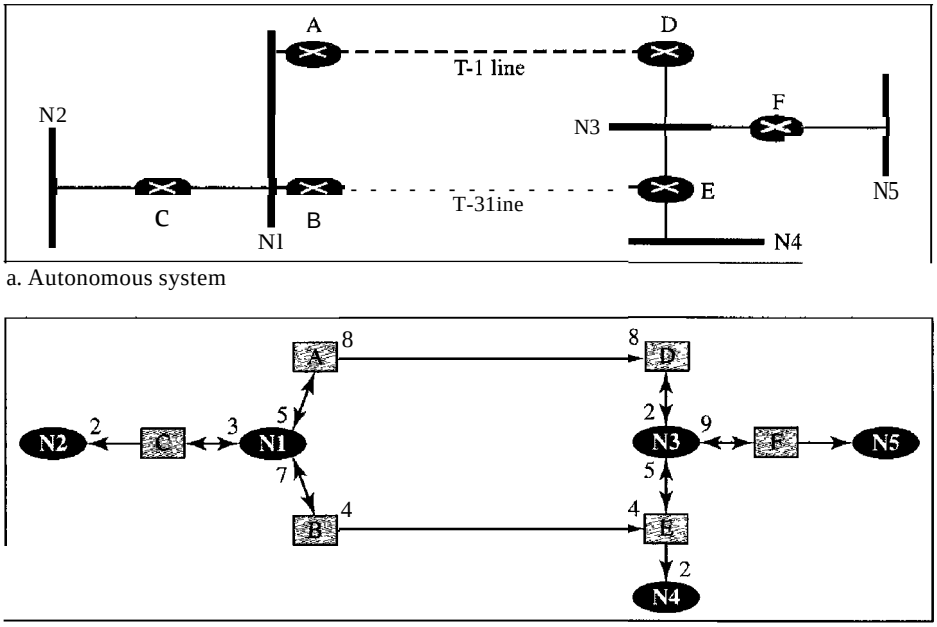
a. Stub network b. Representation

When the link between two routers is broken, the administration may create a **virtual link** between them, using a longer path that probably goes through several routers.

Graphical Representation Let us now examine how an AS can be represented graphically. Figure 22.29 shows a small AS with seven networks and six routers. Two of the networks are point-to-point networks. We use symbols such as Nl and N2 for transient and stub networks. There is no need to assign an identity to a point-to-point network. The figure also shows the graphical representation of the AS as seen by OSPF.

We have used square nodes for the routers and ovals for the networks (represented by designated routers). However, OSPF sees both as nodes. Note that we have three stub networks.

Figure 22.29 *Example of an AS and its graphical representation in OSPF*



a. Autonomous system

b. Graphical representation

Path Vector Routing

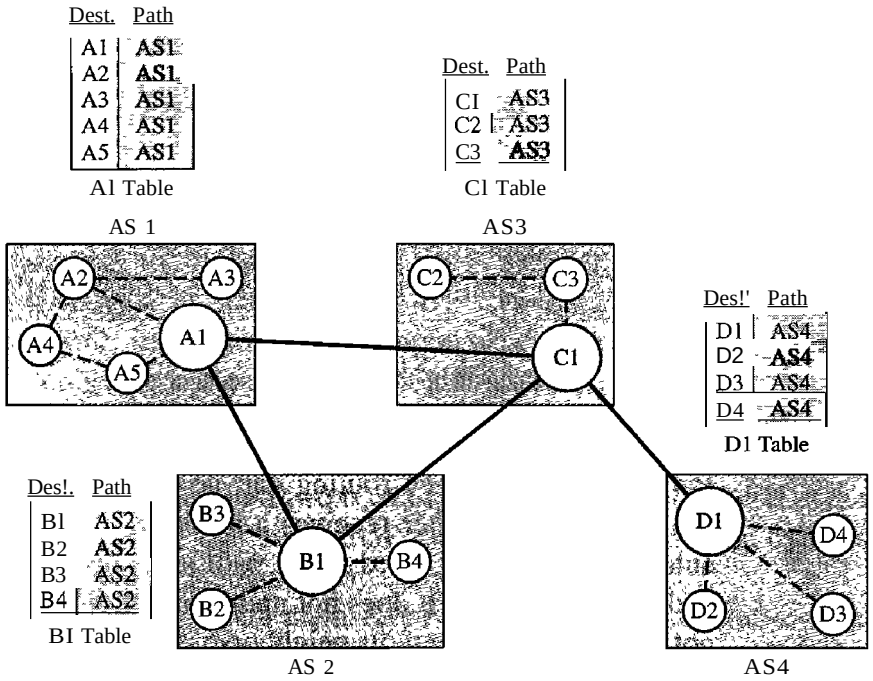
Distance vector and link state routing are both intradomain routing protocols. They can be used inside an autonomous system, but not between autonomous systems. These two protocols are not suitable for interdomain routing mostly because of scalability. Both of these routing protocols become intractable when the domain of operation becomes large. Distance vector routing is subject to instability if there are more than a few hops in the domain of operation. Link state routing needs a huge amount of resources to calculate routing tables. It also creates heavy traffic because of flooding. There is a need for a third routing protocol which we call path vector routing.

Path vector routing proved to be useful for interdomain routing. The principle of path vector routing is similar to that of distance vector routing. In path vector routing, we assume that there is one node (there can be more, but one is enough for our conceptual discussion) in each autonomous system that acts on behalf of the entire autonomous system. Let us call it the speaker node. The speaker node in an AS creates a routing table and advertises it to speaker nodes in the neighboring ASs. The idea is the same as for distance vector routing except that only speaker nodes in each AS can communicate with each other. However, what is advertised is different. A speaker node advertises the path, not the metric of the nodes, in its autonomous system or other autonomous systems.

Initialization

At the beginning, each speaker node can know only the reachability of nodes inside its autonomous system. Figure 22.30 shows the initial tables for each speaker node in a system made of four ASs.

Figure 22.30 *Initial routing tables in path vector routing*



Node A1 is the speaker node for AS1, B1 for AS2, C1 for AS3, and D1 for AS4. Node A1 creates an initial table that shows A1 to A5 are located in AS1 and can be reached through it. Node B1 advertises that B1 to B4 are located in AS2 and can be reached through B1. And so on.

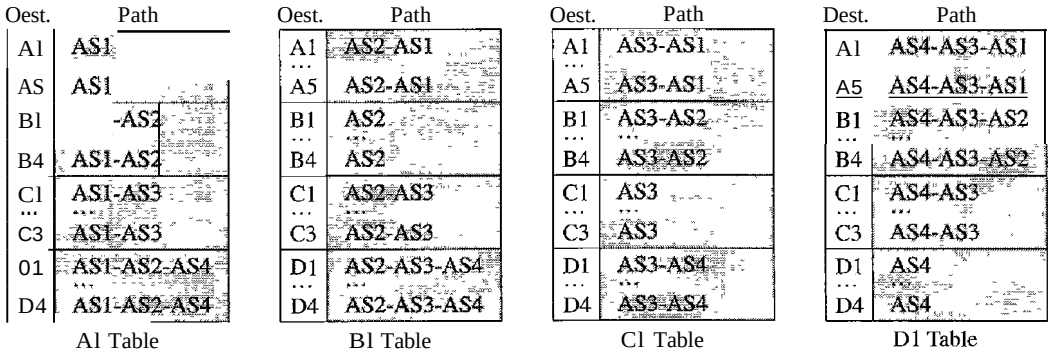
Sharing Just as in distance vector routing, in path vector routing, a speaker in an autonomous system shares its table with immediate neighbors. In Figure 22.30, node A1 shares its table with nodes B1 and C1. Node C1 shares its table with nodes D1, B1, and A1. Node B1 shares its table with C1 and A1. Node D1 shares its table with C1.

Updating When a speaker node receives a two-column table from a neighbor, it updates its own table by adding the nodes that are not in its routing table and adding its own autonomous system and the autonomous system that sent the table. After a while each speaker has a table and knows how to reach each node in other ASs. Figure 22.31 shows the tables for each speaker node after the system is stabilized.

According to the figure, if router A1 receives a packet for nodes A3, it knows that the path is in AS1 (the packet is at home); but if it receives a packet for D1, it knows that the packet should go from AS1, to AS2, and then to AS3. The routing table shows the path completely. On the other hand, if node D1 in AS4 receives a packet for node A2, it knows it should go through AS4, AS3, and AS 1.

- Loop prevention. The instability of distance vector routing and the creation of loops can be avoided in path vector routing. When a router receives a message, it checks to see if its autonomous system is in the path list to the destination. If it is, looping is involved and the message is ignored.

Figure 22.31 *Stabilized tables for three autonomous systems*



- **Policy routing.** Policy routing can be easily implemented through path vector routing. When a router receives a message, it can check the path. If one of the autonomous systems listed in the path is against its policy, it can ignore that path and that destination. It does not update its routing table with this path, and it does not send this message to its neighbors.
- **Optimum path.** What is the optimum path in path vector routing? We are looking for a path to a destination that is the best for the organization that runs the autonomous system. We definitely cannot include metrics in this route because each autonomous system that is included in the path may use a different criterion for the metric. One system may use, internally, RIP, which defines hop count as the metric; another may use OSPF with minimum delay defined as the metric. The optimum path is the path that fits the organization. In our previous figure, each autonomous system may have more than one path to a destination. For example, a path from AS4 to ASI can be AS4-AS3-AS2-AS1, or it can be AS4-AS3-ASI. For the tables, we chose the one that had the smaller number of autonomous systems, but this is not always the case. Other criteria, such as security, safety, and reliability, can also be applied.

BGP

Border Gateway Protocol (BGP) is an interdomain routing protocol using path vector routing. It first appeared in 1989 and has gone through four versions.

Types **of Autonomous** Systems As we said before, the Internet is divided into hierarchical domains called autonomous systems. For example, a large corporation that manages its own network and has full control over it is an autonomous system. A local ISP that provides services to local customers is an autonomous system. We can divide autonomous systems into three categories: stub, multihomed, and transit.

- **Stub** AS. A stub AS has only one connection to another AS. The interdomain data traffic in a stub AS can be either created or terminated in the AS. The hosts in the AS can send data traffic to other ASs. The hosts in the AS can receive data coming from hosts in other ASs. Data traffic, however, cannot pass through a stub AS. A stub AS

is either a source or a sink. A good example of a stub AS is a small corporation or a small local ISP.

- Multihomed AS. A multihomed AS has more than one connection to other ASs, but it is still only a source or sink for data traffic. It can receive data traffic from more than one AS. It can send data traffic to more than one AS, but there is no transient traffic. It does not allow data coming from one AS and going to another AS to pass through. A good example of a multihomed AS is a large corporation that is connected to more than one regional or national AS that does not allow transient traffic.
- Transit AS. A transit AS is a multihomed AS that also allows transient traffic. Good examples of transit ASs are national and international ISPs (Internet backbones).

Path Attributes In our previous example, we discussed a path for a destination network. The path was presented as a list of autonomous systems, but is, in fact, a list of attributes. Each attribute gives some information about the path. The list of attributes helps the receiving router make a more-informed decision when applying its policy.

Attributes are divided into two broad categories: well known and optional. A well-known attribute is one that every BGP router must recognize. An optional attribute is one that needs not be recognized by every router.

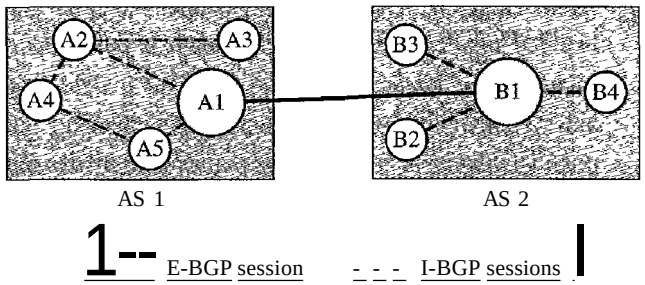
Well-known attributes are themselves divided into two categories: mandatory and discretionary. A *well-known mandatory attribute* is one that must appear in the description of a route. A *well-known discretionary attribute* is one that must be recognized by each router, but is not required to be included in every update message. One well-known mandatory attribute is ORIGIN. This defines the source of the routing information (RIP, OSPF, and so on). Another well-known mandatory attribute is AS_PATH. This defines the list of autonomous systems through which the destination can be reached. Still another well-known mandatory attribute is NEXT-HOP, which defines the next router to which the data packet should be sent.

The optional attributes can also be subdivided into two categories: transitive and nontransitive. An *optional transitive attribute* is one that must be passed to the next router by the router that has not implemented this attribute. An *optional nontransitive attribute* is one that must be discarded if the receiving router has not implemented it.

BGP Sessions The exchange of routing information between two routers using BGP takes place in a session. A session is a connection that is established between two BGP routers only for the sake of exchanging routing information. To create a reliable environment, BGP uses the services of TCP. In other words, a session at the BGP level, as an application program, is a connection at the TCP level. However, there is a subtle difference between a connection in TCP made for BGP and other application programs. When a TCP connection is created for BGP, it can last for a long time, until something unusual happens. For this reason, BGP sessions are sometimes referred to as *semi-pennanent connections.*

External and Internal BGP If we want to be precise, BGP can have two types of sessions: external BGP (E-BGP) and internal BGP (I-BGP) sessions. The E-BGP session is used to exchange information between two speaker nodes belonging to two different autonomous systems. The I-BGP session, on the other hand, is used to exchange routing information between two routers inside an autonomous system. Figure 22.32 shows the idea.

Figure 22.32 *Internal and external BGP sessions*



The session established between AS 1 and AS2 is an E-BOP session. The two speaker routers exchange information they know about networks in the Internet. However, these two routers need to collect information from other routers in the autonomous systems. This is done using I-BOP sessions.

22.4 MULTICAST ROUTING PROTOCOLS

In this section, we discuss multicasting and multicast routing protocols. We first define the term *multicasting* and compare it to unicasting and broadcasting. We also briefly discuss the applications of multicasting. Finally, we move on to multicast routing and the general ideas and goals related to it. We also discuss some common multicast routing protocols used in the Internet today.

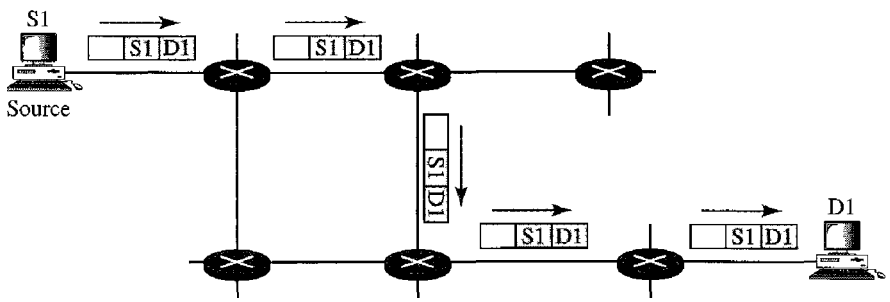
Unicast, Multicast, **and** Broadcast

A message can be unicast, multicast, or broadcast. Let us clarify these terms as they relate to the Internet.

Unicasting

In unicast communication, there is one source and one destination. The relationship between the source and the destination is one-to-one. In this type of communication, both the source and destination addresses, in the IP datagram, are the unicast addresses assigned to the hosts (or host interfaces, to be more exact). In Figure 22.33, a unicast

Figure 22.33 *Unicasting*



packet starts from the source S1 and passes through routers to reach the destination D1. We have shown the networks as a link between the routers to simplify the figure.

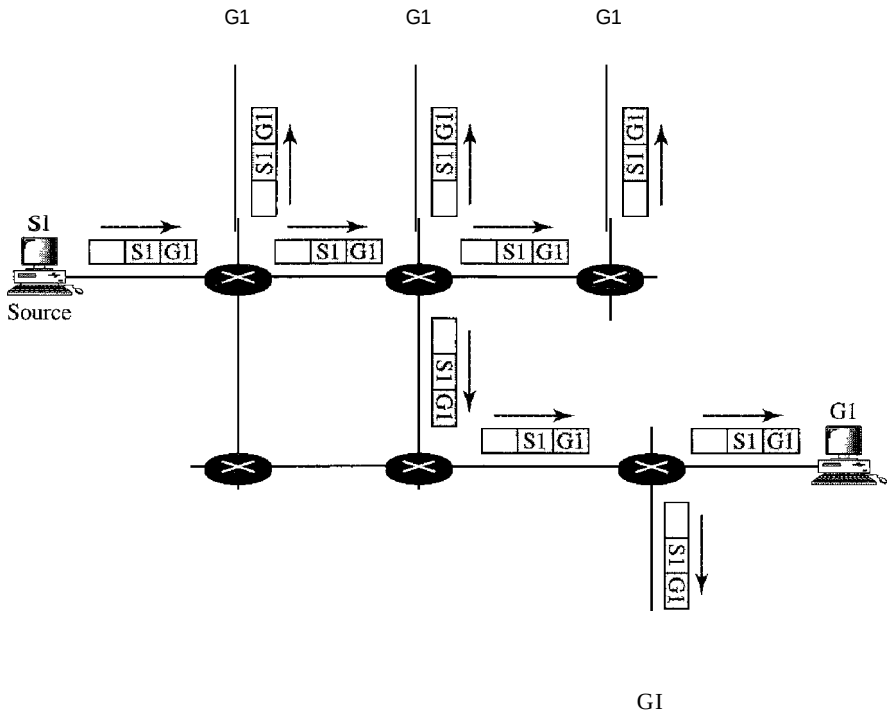
Note that in unicasting, when a router receives a packet, it forwards the packet through only one of its interfaces (the one belonging to the optimum path) as defined in the routing table. The router may discard the packet if it cannot find the destination address in its routing table.

In unicasting, the router forwards the received packet through only one of its interfaces.

Multicasting

In multicast communication, there is one source and a group of destinations. The relationship is one-to-many. In this type of communication, the source address is a unicast address, but the destination address is a group address, which defines one or more destinations. The group address identifies the members of the group. Figure 22.34 shows the idea behind multicasting.

Figure 22.34 *Multicasting*



A multicast packet starts from the source S1 and goes to all destinations that belong to group G1. In multicasting, when a router receives a packet, it may forward it through several of its interfaces.

In multicasting, the router may forward the received packet through several of its interfaces.

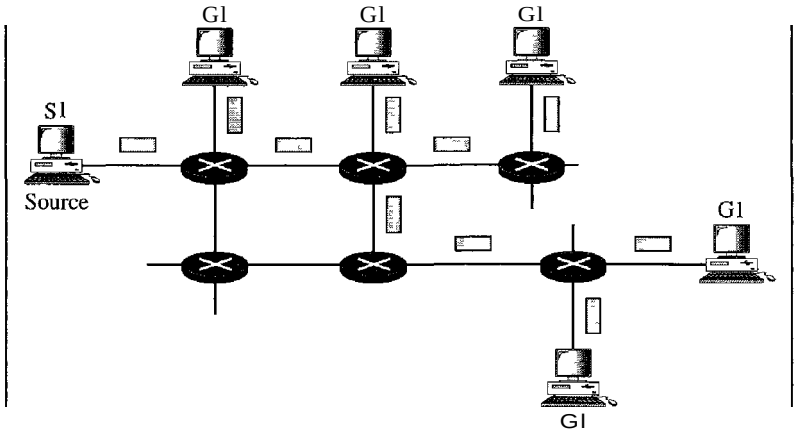
Broadcasting

In broadcast communication, the relationship between the source and the destination is one-to-all. There is only one source, but all the other hosts are the destinations. The Internet does not explicitly support broadcasting because of the huge amount of traffic it would create and because of the bandwidth it would need. Imagine the traffic generated in the Internet if one person wanted to send a message to everyone else connected to the Internet.

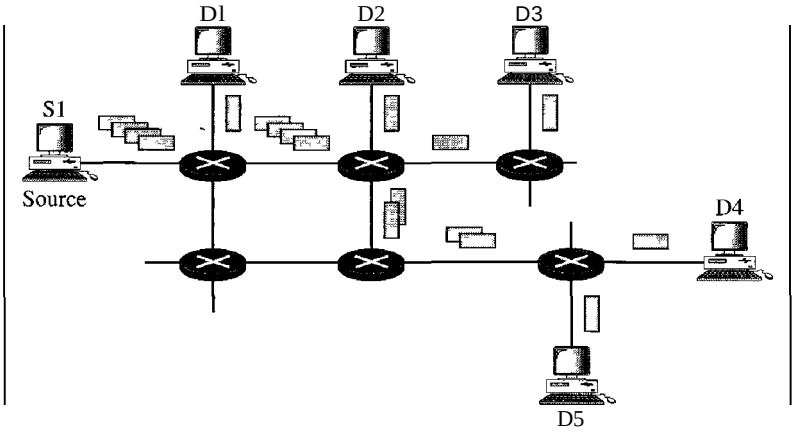
Multicasting Versus Multiple Unicasting

Before we finish this section, we need to distinguish between multicasting and multiple unicasting. Figure 22.35 illustrates both concepts.

Figure 22.35 *Multicasting versus multiple unicasting*



a. Multicasting



b. Multiple nnicasting

Multicasting starts with one single packet from the source that is duplicated by the routers. The destination address in each packet is the same for all duplicates. Note that only one single copy of the packet travels between any two routers.

In multiple unicasting, several packets start from the source. If there are five destinations, for example, the source sends five packets, each with a different unicast destination address. Note that there may be multiple copies traveling between two routers. For example, when a person sends an e-mail message to a group of people, this is multiple unicasting. The e-mail software creates replicas of the message, each with a different destination address and sends them one by one. This is not multicasting; it is multiple unicasting.

Emulation of Multicasting with Unicasting

You might wonder why we have a separate mechanism for multicasting, when it can be emulated with unicasting. There are two obvious reasons for this.

1. Multicasting is more efficient than multiple unicasting. In Figure 22.35, we can see how multicasting requires less bandwidth than does multiple unicasting. In multiple unicasting, some of the links must handle several copies.
2. In multiple unicasting, the packets are created by the source with a relative delay between packets. If there are 1000 destinations, the delay between the first and the last packet may be unacceptable. In multicasting, there is no delay because only one packet is created by the source.

Emulation of multicasting through multiple unicasting is not efficient and may create long delays, particularly with a large group.

Applications

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

Access to Distributed Databases

Most of the large databases today are distributed. That is, the information is stored in more than one location, usually at the time of production. The user who needs to access the database does not know the location of the information. A user's request is multicast to all the database locations, and the location that has the information responds.

Information Dissemination

Businesses often need to send information to their customers. If the nature of the information is the same for each customer, it can be multicast. In this way a business can send one message that can reach many customers. For example, a software update can be sent to all purchasers of a particular software package.

Dissemination of News

In a similar manner news can be easily disseminated through multicasting. One single message can be sent to those interested in a particular topic. For example, the statistics of the championship high school basketball tournament can be sent to the sports editors of many newspapers.

Teleconferencing

Teleconferencing involves multicasting. The individuals attending a teleconference all need to receive the same information at the same time. Temporary or permanent groups can be formed for this purpose. For example, an engineering group that holds meetings every Monday morning could have a permanent group while the group that plans the holiday party could form a temporary group.

Distance Learning

One growing area in the use of multicasting is distance learning. Lessons taught by one single professor can be received by a specific group of students. This is especially convenient for those students who find it difficult to attend classes on campus.

Multicast Routing

In this section, we first discuss the idea of optimal routing, common in all multicast protocols. We then give an overview of multicast routing protocols.

Optimal Routing: Shortest Path Trees

The process of optimal interdomain routing eventually results in the finding of the *shortest path tree.* The root of the tree is the source, and the leaves are the potential destinations. The path from the root to each destination is the shortest path. However, the number of trees and the formation of the trees in unicast and multicast routing are different. Let us discuss each separately.

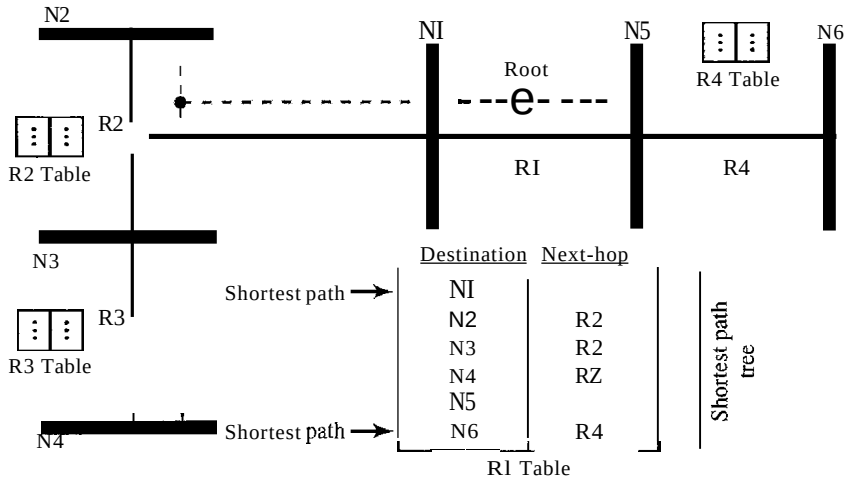
Unicast Routing In unicast routing, when a router receives a packet to forward, it needs to find the shortest path to the destination of the packet. The router consults its routing table for that particular destination. The next-hop entry corresponding to the destination is the start of the shortest path. The router knows the shortest path for each destination, which means that the router has a shortest path tree to optimally reach all destinations. In other words, each line of the routing table is a shortest path; the whole routing table is a shortest path tree. In unicast routing, each router needs only one shortest path tree to forward a packet; however, each router has its own shortest path tree. Figure 22.36 shows the situation.

The figure shows the details of the routing table and the shortest path tree for router Rl. Each line in the routing table corresponds to one path from the root to the corresponding network. The whole table represents the shortest path tree.

In unicast routing, each router in the domain has a table that defines a shortest path tree to possible destinations.

Multicast Routing When a router receives a multicast packet, the situation is different from when it receives a unicast packet. A multicast packet may have destinations in more than one network. Forwarding of a single packet to members of a group requires a shortest path tree. If we have *n* groups, we may need *n* shortest path trees. We can imagine the complexity of multicast routing. Two approaches have been used to solve the problem: source-based trees and group-shared trees.

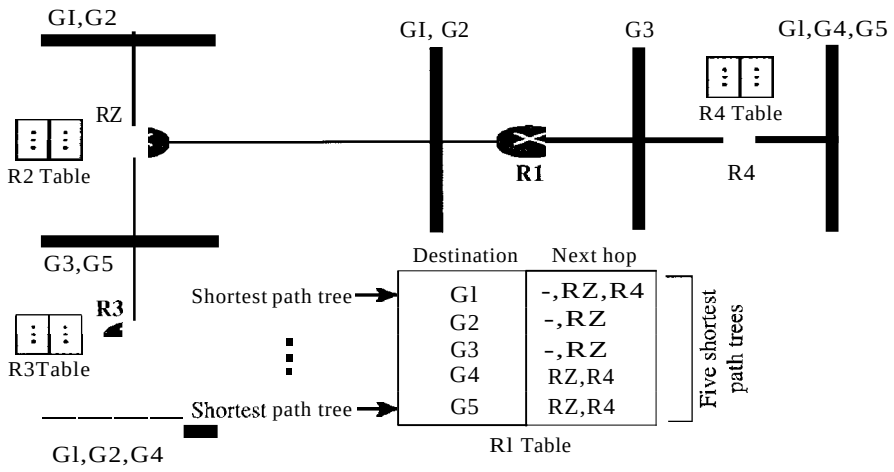
Figure 22.36 *Shortest path tree in unicast routing*



In multicast routing, each involved router needs to construct a shortest path tree for each group.

- Source-Based Tree. In the source-based tree approach, each router needs to have one shortest path tree for each group. The shortest path tree for a group defines the next hop for each network that has loyal member(s) for that group. In Figure 22.37, we assume that we have only five groups in the domain: GI, G2, G3, G4, and G5. At the moment GI has loyal members in four networks, G2 in three, G3 in two, G4 in two, and G5 in two. We have shown the names of the groups with loyal members on each network. Figure 22.37 also shows the multicast routing table for router RI. There is one shortest path tree for each group; therefore there are five shortest path trees for five groups. If router Rl receives a packet with destination

Figure 22.37 *Source-based tree approach*

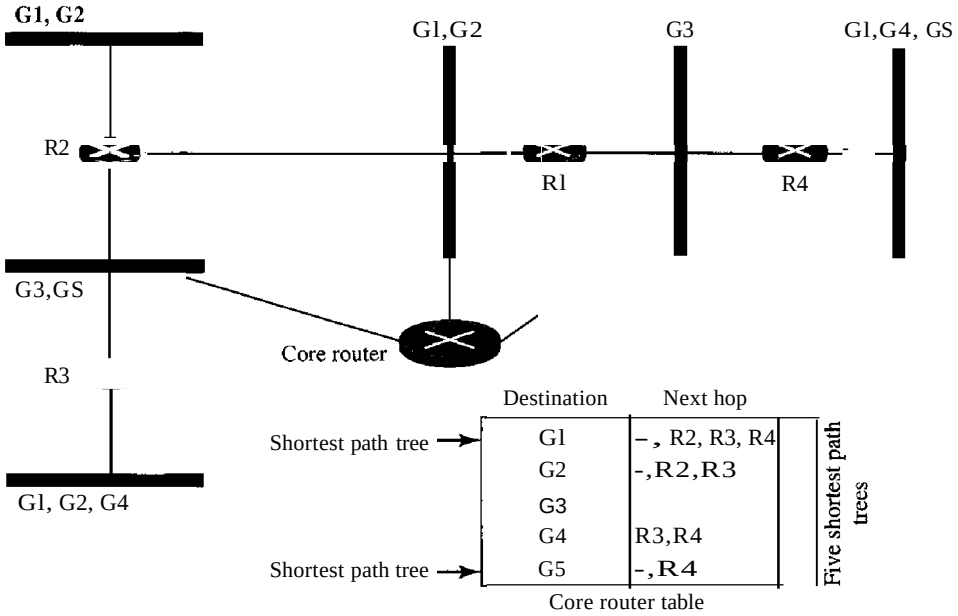


address G1, it needs to send a copy of the packet to the attached network, a copy to router R2, and a copy to router R4 so that all members of G1 can receive a copy. In this approach, if the number of groups is *m,* each router needs to have *m* shortest path trees, one for each group. We can imagine the complexity of the routing table if we have hundreds or thousands of groups. However, we will show how different protocols manage to alleviate the situation.

In the source-based tree approach, each router needs to have one shortest path tree for each group.

- Group-Shared Tree. In the group-shared tree approach, instead of each router having *m* shortest path trees, only one designated router, called the center core, or rendezvous router, takes the responsibility of distributing multicast traffic. The core has *m* shortest path trees in its routing table. The rest of the routers in the domain have none. If a router receives a multicast packet, it encapsulates the packet in a unicast packet and sends it to the core router. The core router removes the multicast packet from its capsule, and consults its routing table to route the packet. Figure 22.38 shows the idea.

Figure 22.38 *Group-shared tree approach*



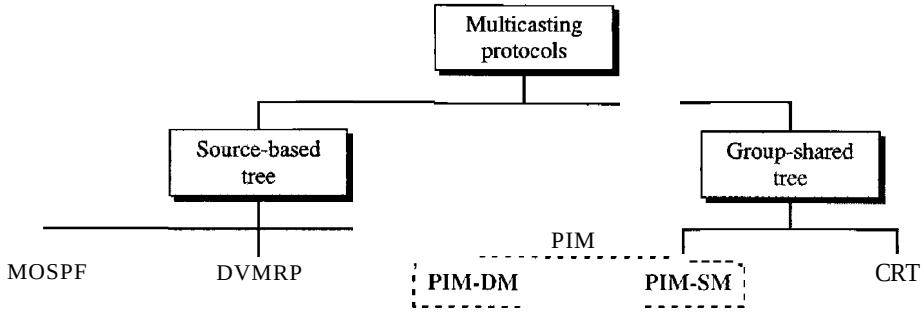
In the group-shared tree approach, only the core router, which has a shortest path tree for each group, is involved in multicasting.

Routing Protocols

During the last few decades, several multicast routing protocols have emerged. Some of these protocols are extensions of unicast routing protocols; others are totally new.

We discuss these protocols in the remainder of this chapter. Figure 22.39 shows the taxonomy of these protocols.

Figure 22.39 *Taxonomy of common multicast protocols*



Multicast Link State Routing: MOSPF

In this section, we briefly discuss multicast link state routing and its implementation in the Internet, MOSPF.

Multicast Link State Routing We discussed unicast link state routing in Section 22.3. We said that each router creates a shortest path tree by using Dijkstra's algorithm. The routing table is a translation of the shortest path tree. Multicast link state routing is a direct extension of unicast routing and uses a source-based tree approach. Although unicast routing is quite involved, the extension to multicast routing is very simple and straightforward.

Multicast link state routing uses the source-based tree approach.

Recall that in unicast routing, each node needs to advertise the state of its links. For multicast routing, a node needs to revise the interpretation of *state.* A node advertises every group which has any loyal member on the link. Here the meaning of state is "what groups are active on this link." The information about the group comes from IGMP (see Chapter 21). Each router running IGMP solicits the hosts on the link to find out the membership status.

When a router receives all these LSPs, it creates *n (n* is the number of groups) topologies, from which *n* shortest path trees are made by using Dijkstra's algorithm. So each router has a routing table that represents as many shortest path trees as there are groups.

The only problem with this protocol is the time and space needed to create and save the many shortest path trees. The solution is to create the trees only when needed. When a router receives a packet with a multicast destination address, it runs the Dijkstra algorithm to calculate the shortest path tree for that group. The result can be cached in case there are additional packets for that destination.

MOSPF Multicast Open Shortest Path First (MOSPF) protocol is an extension of the OSPF protocol that uses multicast link state routing to create source-based trees.

The protocol requires a new link state update packet to associate the unicast address of a host with the group address or addresses the host is sponsoring. This packet is called the group-membership LSA. In this way, we can include in the tree only the hosts (using their unicast addresses) that belong to a particular group. In other words, we make a tree that contains all the hosts belonging to a group, but we use the unicast address of the host in the calculation. For efficiency, the router calculates the shortest path trees on demand (when it receives the first multicast packet). In addition, the tree can be saved in cache memory for future use by the same source/group pair. MOSPF is a data-driven protocol; the first time an MOSPF router sees a datagram with a given source and group address, the router constructs the Dijkstra shortest path tree.

Multicast Distance Vector: DVMRP

In this section, we briefly discuss multicast distance vector routing and its implementation in the Internet, DVMRP.

Multicast Distance Vector Routing Unicast distance vector routing is very simple; extending it to support multicast routing is complicated. Multicast routing does not allow a router to send its routing table to its neighbors. The idea is to create a table from scratch by using the information from the unicast distance vector tables.

Multicast distance vector routing uses source-based trees, but the router never actually makes a routing table. When a router receives a multicast packet, it forwards the packet as though it is consulting a routing table. We can say that the shortest path tree is evanescent. After its use (after a packet is forwarded) the table is destroyed.

To accomplish this, the multicast distance vector algorithm uses a process based on four decision-making strategies. Each strategy is built on its predecessor. We explain them one by one and see how each strategy can improve the shortcomings of the previous one.

- Flooding. Flooding is the first strategy that comes to mind. A router receives a packet and, without even looking at the destination group address, sends it out from every interlace except the one from which it was received. Flooding accomplishes the first goal of multicasting: every network with active members receives the packet. However, so will networks without active members. This is a broadcast, not a multicast. There is another problem: it creates loops. A packet that has left the router may come back again from another interlace or the same interlace and be forwarded again. Some flooding protocols keep a copy of the packet for a while and discard any duplicates to avoid loops. The next strategy, reverse path forwarding, corrects this defect.

Flooding broadcasts packets, but creates loops in the systems.

- Reverse Path Forwarding (RPF). RPF is a modified flooding strategy. To prevent loops, only one copy is forwarded; the other copies are dropped. In RPF, a router forwards only the copy that has traveled the shortest path from the source to the router. To find this copy, RPF uses the unicast routing table. The router receives a packet and extracts the source address (a unicast address). It consults its unicast routing table as though it wants to send a packet to the source address. The routing table tells the

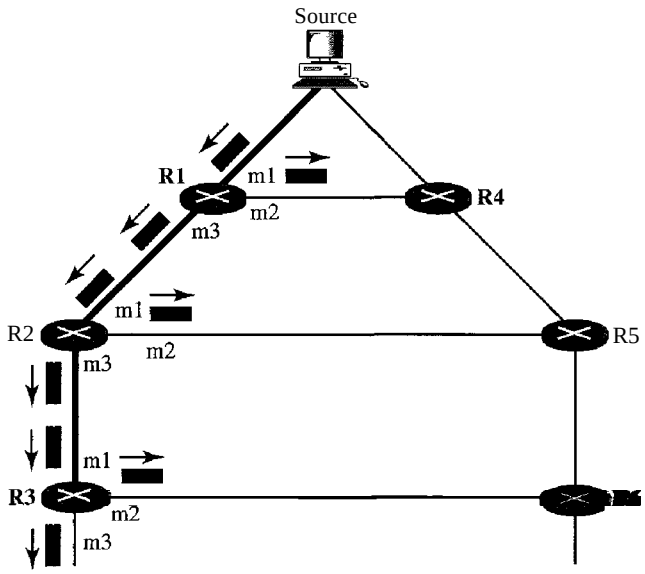
router the next hop. If the multicast packet has just come from the hop defined in the table, the packet has traveled the shortest path from the source to the router because the shortest path is reciprocal in unicast distance vector routing protocols. If the path from A to B is the shortest, then it is also the shortest from B to A. The router forwards the packet if it has traveled from the shortest path; it discards it otherwise.

This strategy prevents loops because there is always one shortest path from the source to the router. If a packet leaves the router and comes back again, it has not traveled the shortest path. To make the point clear, let us look at Figure 22.40.

Figure 22.40 shows part of a domain and a source. The shortest path tree as calculated by routers RI, R2, and R3 is shown by a thick line. When RI receives a packet from the source through the interface rnl, it consults its routing table and finds that the shortest path from RI to the source is through interface mI. The packet is forwarded. However, if a copy of the packet has arrived through interface m2, it is discarded because m2 does not define the shortest path from RI to the source. The story is the same with R2 and R3. You may wonder what happens if a copy of a packet that arrives at the ml interface of R3, travels through R6, R5, R2, and then enters R3 through interface ml. This interface is the correct interface for R3. Is the copy of the packet forwarded? The answer is that this scenario never happens because when the packet goes from R5 to R2, it will be discarded by R2 and never reaches R3. The upstream routers toward the source always discard a packet that has not gone through the shortest path, thus preventing confusion for the downstream routers.

RPF eliminates the loop in the flooding process.

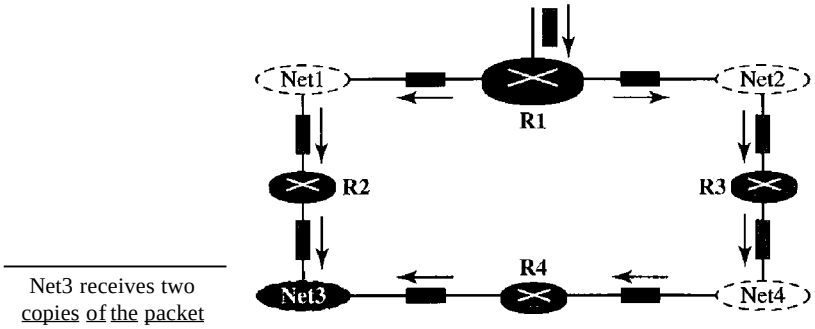
Figure 22.40 *Reverse path forwarding (RPF)*



- Reverse Path Broadcasting (RPB). RPF guarantees that each network receives a copy of the multicast packet without formation of loops. However, RPF does not

guarantee that each network receives only one copy; a network may receive two or more copies. The reason is that RPF is not based on the destination address (a group address); forwarding is based on the source address. To visualize the problem, let us look at Figure 22.41.

Figure 22.41 *Problem with RPF*

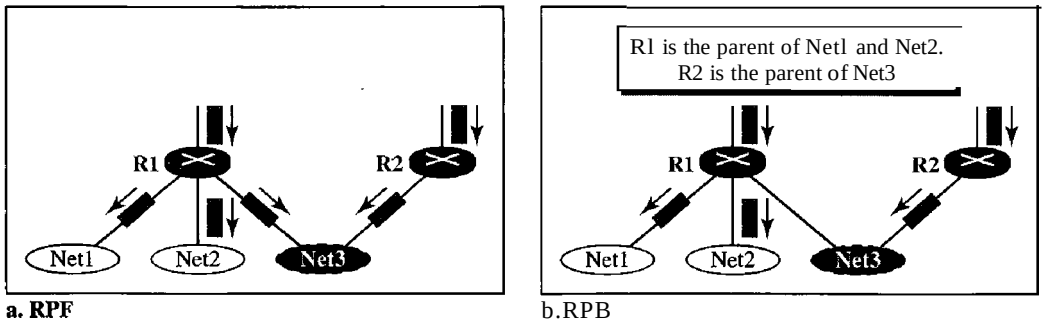


Net3 in this figure receives two copies of the packet even though each router just sends out one copy from each interface. There is duplication because a tree has not been made; instead of a tree we have a graph. Net3 has two parents: routers R2 andR4.

To eliminate duplication, we must define only one parent router for each network. We must have this restriction: A network can receive a multicast packet from a particular source only through a designated parent router.

Now the policy is clear. For each source, the router sends the packet only out of those interfaces for which it is the designated parent. This policy is called reverse path broadcasting (RPB). RPB guarantees that the packet reaches every network and that every network receives only one copy. Figure 22.42 shows the difference between RPF and RPB.

Figure 22.42 *RPF Versus RPB*



a. RPF b.RPB

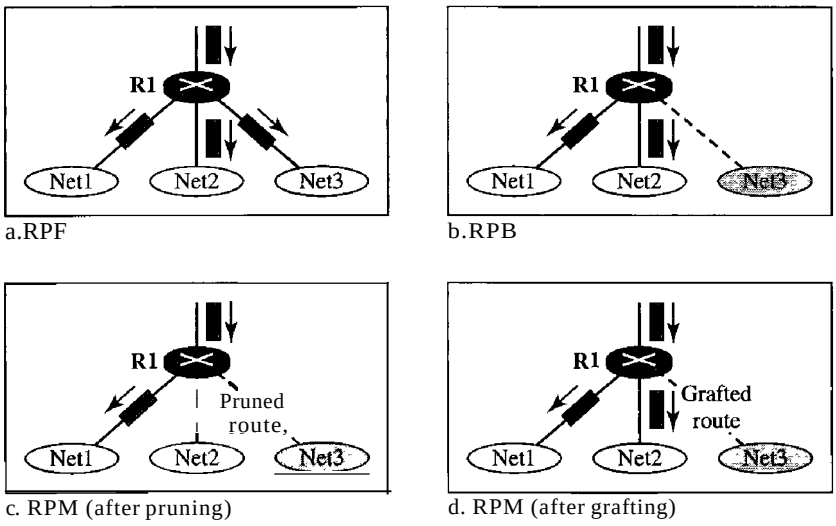
The reader may ask how the designated parent is determined. The designated parent router can be the router with the shortest path to the source. Because routers periodically

send updating packets to each other (in RIP), they can easily determine which router in the neighborhood has the shortest path to the source (when interpreting the source as the destination), If more than one router qualifies, the router with the smallest IP address is selected.

RPB creates a shortest path broadcast tree from the source to each destination. It guarantees that each destination receives one and only one copy of the packet.

- Reverse Path Multicasting (RPM). As you may have noticed, RPB does not multicast the packet, it broadcasts it. This is not efficient. To increase efficiency, the multicast packet must reach only those networks that have active members for that particular group. This is called reverse path multicasting (RPM). To convert broadcasting to multicasting, the protocol uses two procedures, pruning and grafting. Figure 22.43 shows the idea of pruning and grafting.

Figure 22.43 *RPF, RPB, and RPM*



The designated parent router of each network is responsible for holding the membership information. This is done through the IGMP protocol described in Chapter 21. The process starts when a router connected to a network finds that there is no interest in a multicast packet. The router sends a prune message to the upstream router so that it can exclude the corresponding interface. That is, the upstream router can stop sending multicast messages for this group through that interface. Now if this router receives prune messages from all downstream routers, it, in turn, sends a prune message to its upstream router.

What if a leaf router (a router at the bottom of the tree) has sent a prune message but suddenly realizes, through IGMP, that one of its networks is again interested in receiving the multicast packet? It can send a graft message. The graft message forces the upstream router to resume sending the multicast messages.

RPM adds pruning and grafting to RPB to create a multicast shortest path tree that supports dynamic membership changes.

DVMRP The Distance Vector Multicast Routing Protocol (DVMRP) is an implementation of multicast distance vector routing. It is a source-based routing protocol, based on RIP.

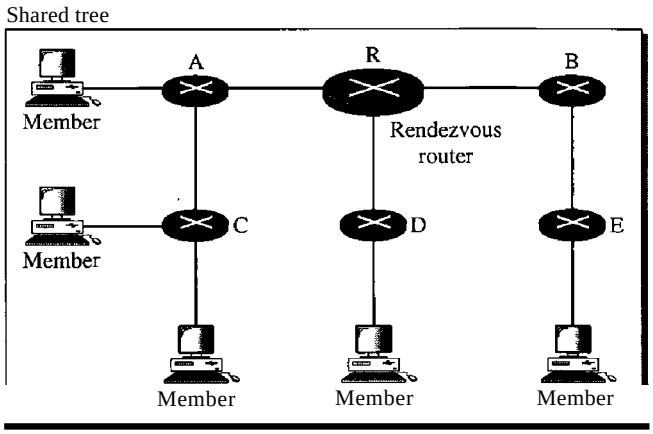
CRT

The Core-Based Tree (CBT) protocol is a group-shared protocol that uses a core as the root of the tree. The autonomous system is divided into regions, and a core (center router or rendezvous router) is chosen for each region.

Formation of the Tree After the rendezvous point is selected, every router is infonned of the unicast address of the selected router. Each router then sends a unicast join message (similar to a grafting message) to show that it wants to join the group. This message passes through all routers that are located between the sender and the rendezvous router. Each intermediate router extracts the necessary infonnation from the message, such as the unicast address of the sender and the interface through which the packet has arrived, and forwards the message to the next router in the path. When the rendezvous router has received all join messages from every member of the group, the tree is formed. Now every router knows its upstream router (the router that leads to the root) and the downstream router (the router that leads to the leaf).

If a router wants to leave the group, it sends a leave message to its upstream router. The upstream router removes the link to that router from the tree and forwards the message to its upstream router, and so on. Figure 22.44 shows a group-shared tree with its rendezvous router.

Figure 22.44 *Group-shared tree with rendezvous router*

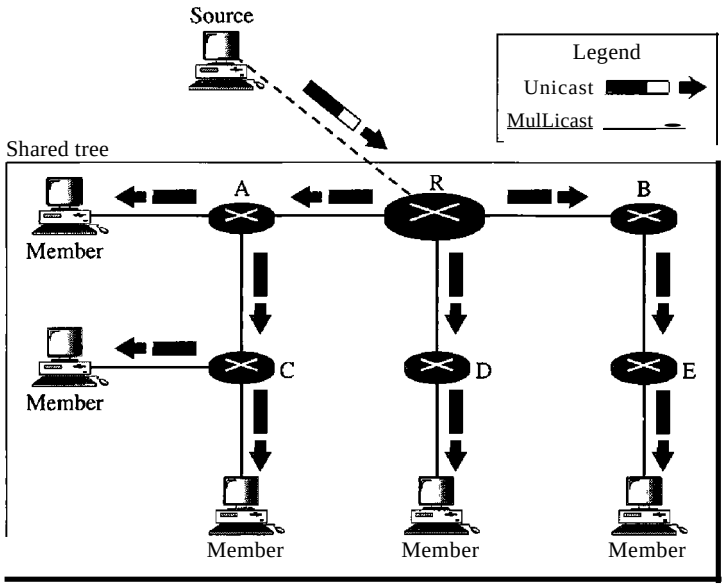


The reader may have noticed two differences between DVMRP and MOSPF, on one hand, and CBT, on the other. First, the tree for the first two is made from the root up; the tree for CBT is fonned from the leaves down. Second, in DVMRP, the tree is

first made (broadcasting) and then pruned; in CBT, there is no tree at the beginning; the joining (grafting) gradually makes the tree.

Sending Multicast Packets After formation of the tree, any source (belonging to the group or not) can send a multicast packet to all members of the group. It simply sends the packet to the rendezvous router, using the unicast address of the rendezvous router; the rendezvous router distributes the packet to all members of the group. Figure 22.45 shows how a host can send a multicast packet to all members of the group. Note that the source host can be any of the hosts inside the shared tree or any host outside the shared tree. In Figure 22.45 we show one located outside the shared tree.

Figure 22.45 *Sending a multicast packet to the rendezvous router*



Selecting the Rendezvous Router This approach is simple except for one point. How do we select a rendezvous router to optimize the process and multicasting as well? Several methods have been implemented. However, this topic is beyond the scope of this book, and we leave it to more advanced books.

In summary, the Core-Based Tree (CBT) is a group-shared tree, center-based protocol using one tree per group. One of the routers in the tree is called the core. A packet is sent from the source to members of the group following this procedure:

1. The source, which mayor may not be part of the tree, encapsulates the multicast packet inside a unicast packet with the unicast destination address of the core and sends it to the core. This part of delivery is done using a unicast address; the only recipient is the core router.
2. The core decapsulates the unicast packet and forwards it to all interested intetfaces.
3. Each router that receives the multicast packet, in tum, forwards it to all interested interfaces.

In CRT, the source sends the multicast packet (encapsulated in a unicast packet) to the core router. The core router decapsulates the packet and forwards it to all interested interfaces.

PIM

Protocol Independent Multicast (PIM) is the name given to two independent multicast routing protocols: Protocol Independent Multicast, Dense Mode (PIM-DM) and Protocol Independent Multicast, Sparse Mode (PIM-SM). Both protocols are unicast-protocol-dependent, but the similarity ends here. We discuss each separately.

PIM-DM PIM-DM is used when there is a possibility that each router is involved in multicasting (dense mode). In this environment, the use of a protocol that broadcasts the packet is justified because almost all routers are involved in the process.

PIM-DM is used in a dense multicast environment, such as a LAN.

PIM-DM is a source-based tree routing protocol that uses RPF and pruning and grafting strategies for multicasting. Its operation is like that of DVMRP; however, unlike DVMRP, it does not depend on a specific unicasting protocol. It assumes that the autonomous system is using a unicast protocol and each router has a table that can find the outgoing interface that has an optimal path to a destination. This unicast protocol can be a distance vector protocol (RIP) or link state protocol (OSPF).

PIM-DM uses RPF and pruning and grafting strategies to handle multicasting. However, it is independent of the underlying unicast protocol.

PIM-SM PIM-SM is used when there is a slight possibility that each router is involved in multicasting (sparse mode). In this environment, the use of a protocol that broadcasts the packet is not justified; a protocol such as CBT that uses a group-shared tree is more appropriate.

PIM-SM is used in a sparse multicast environment such as a WAN.

PIM-SM is a group-shared tree routing protocol that has a rendezvous point (RP) as the source of the tree. Its operation is like CBT; however, it is simpler because it does not require acknowledgment from a join message. In addition, it creates a backup set of RPs for each region to cover RP failures.

One of the characteristics of PIM-SM is that it can switch from a group-shared tree strategy to a source-based tree strategy when necessary. This can happen if there is a dense area of activity far from the RP. That area can be more efficiently handled with a source-based tree strategy instead of a group-shared tree strategy.

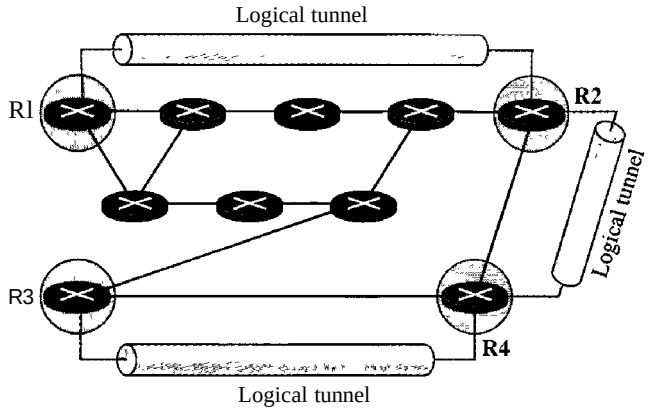
PIM-SM is similar to CRT but uses a simpler procedure.

MBONE

Multimedia and real-time communication have increased the need for multicasting in the Internet. However, only a small fraction of Internet routers are multicast routers. In

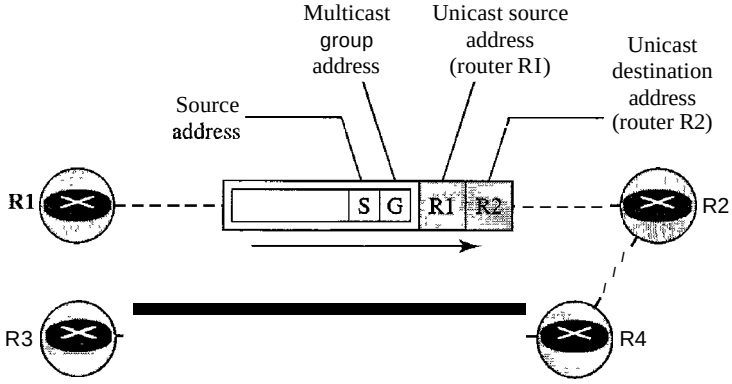
other words, a multicast router may not find another multicast router in the neighborhood to forward the multicast packet. Although this problem may be solved in the next few years by adding more and more multicast routers, there is another solution to this problem. The solution is tunneling. The multicast routers are seen as a group of routers on top of unicast routers. The multicast routers may not be connected directly, but they are connected logically. Figure 22.46 shows the idea. In Figure 22.46, only the routers enclosed in the shaded circles are capable of multicasting. Without tunneling, these routers are isolated islands. To enable multicasting, we make a multicast backbone (MBONE) out of these isolated routers by using the concept of tunneling.

Figure 22.46 *Logical tunneling*



A logical tunnel is established by encapsulating the multicast packet inside a unicast packet. The multicast packet becomes the payload (data) of the unicast packet. The intermediate (nonmulticast) routers forward the packet as unicast routers and deliver the packet from one island to another. It's as if the unicast routers do not exist and the two multicast routers are neighbors. Figure 22.47 shows the concept. So far the only protocol that supports MBONE and tunneling is DVMRP.

Figure 22.47 *MBONE*



22.5 RECOMMENDED READING

For more details about subjects discussed in this chapter, we recommend the following books and sites. The items in brackets [...] refer to the reference list at the end of the text.

Books

Delivery and forwarding are discussed in Chapter 6 of [For06]. Unicast routing protocols are discussed in Chapter 14 of [For06]. Multicasting and multicast routing are discussed in Chapter 15 of [For06]. For a complete discussion of multicasting see [WZOl]. For routing protocols see [HuiOO]. OSPF is discussed in [Moy98].

Sites

- www.ietf.org/rfc.html Information about RFCs

RFCs

A discussion of RIP can be found in the following RFCs:

1131,1245,1246,1247,1370,1583,1584,1585, 1586,1587, 1722,1723,2082,2453

A discussion of OSPF can be found in the following RFCs:

1131,1245,1246,1247,1370,1583,1584,1585, 1586, 1587,2178,2328,2329,2370

A discussion of BGP can be found in the following RFCs:

1092,1105,1163,1265,1266,1267,1364,1392,1403, 1565,1654,1655,1665,1771,1772, 1745,1774,2283

22.6 KEY TERMS

address aggregation
area
area border routers
area identification
autonomous system (AS)
backbone router
Border Gateway Protocol (BGP)
broadcasting
Core-Based Tree (CBT) protocol
data-driven
default method
delivery
designated parent router
Dijkstra's algorithm
direct delivery
distance learning
Distance Vector Multicast Routing Protocol (DVMRP)
distance vector routing
distributed database
dynamic routing method
dynamic routing table
flooding
forwarding
graft message
group-shared tree

hierarchical routing
hop count
host-specific method
ifconfig
immediate neighbors
indirect delivery
interdomain routing
intradomain routing
least-cost tree
link state routing
logical tunnel
longest mask matching
metric
multicast backbone (MBONE)
Multicast Open Shortest Path First (MOSPF)
multicast router
multicast routing
multicasting
multiple unicasting
netstat
network-specific method
next-hop address
next-hop method
Open Shortest Path First (OSPF)
optional attribute
OSPF protocol
path vector routing
point-to-point link
poison reverse
policy routing
Protocol Independent Multicast (PIM)
Protocol Independent Multicast, Dense Mode (PIM-DM)
Protocol Independent Multicast, Sparse Mode (PIM-SM)
prune message
rendezvous router
rendezvous-point tree
reverse path broadcasting (RPB)
reverse path forwarding (RPF)
reverse path multicasting (RPM)
route method
routing
Routing Information Protocol (RIP)
routing protocols
shortest path tree
slow convergence
source-based tree
speaker node
split horizon
static routing table
stub link
switching fabric
teleconferencing
transient link
triggered update
tunneling
unicasting
update message
virtual link
well-known attribute

22.7 SUMMARY

- The delivery of a packet is called direct if the deliverer (host or router) and the destination are on the same network; the delivery of a packet is called indirect if the deliverer (host or router) and the destination are on different networks.
- In the next-hop method, instead of a complete list of the stops the packet must make, only the address of the next hop is listed in the routing table; in the network-specific method, all hosts on a network share one entry in the routing table.

- In the host-specific method, the full IP address of a host is given in the routing table.
- In the default method, a router is assigned to receive all packets with no match in the routing table.
- The routing table for classless addressing needs at least four columns.
- Address aggregation simplifies the forwarding process in classless addressing.
- Longest mask matching is required in classless addressing.
- Classless addressing requires hierarchical and geographical routing to prevent immense routing tables.
- A static routing table's entries are updated manually by an administrator; a dynamic routing table's entries are updated automatically by a routing protocol.
- A metric is the cost assigned for passage of a packet through a network.
- An autonomous system (AS) is a group of networks and routers under the authority of a single administration.
- RIP is based on distance vector routing, in which each router shares, at regular intervals, its knowledge about the entire AS with its neighbors.
- Two shortcomings associated with the RIP protocol are slow convergence and instability. Procedures to remedy RIP instability include triggered update, split horizons, and poison reverse.
- OSPF divides an AS into areas, defined as collections of networks, hosts, and routers.
- OSPF is based on link state routing, in which each router sends the state of its neighborhood to every other router in the area. A packet is sent only if there is a change in the neighborhood.
- OSPF routing tables are calculated by using Dijkstra's algorithm.
- BGP is an interautonomous system routing protocol used to update routing tables.
- BGP is based on a routing protocol called path vector routing. In this protocol, the ASs through which a packet must pass are explicitly listed.
- In a source-based tree approach to multicast routing, the source/group combination determines the tree; in a group-shared tree approach to multicast routing, the group determines the tree.
- MOSPF is a multicast routing protocol that uses multicast link state routing to create a source-based least-cost tree.
- In reverse path forwarding (RPF), the router forwards only the packets that have traveled the shortest path from the source to the router.
- Reverse path broadcasting (RPB) creates a shortest path broadcast tree from the source to each destination. It guarantees that each destination receives one and only one copy of the packet.
- Reverse path multicasting (RPM) adds pruning and grafting to RPB to create a multicast shortest path tree that supports dynamic membership changes.
- DVMRP is a multicast routing protocol that uses the distance routing protocol to create a source-based tree.
- The Core-Based Tree (CBT) protocol is a multicast routing protocol that uses a router as the root of the tree.

- PIM-DM is a source-based tree routing protocol that uses RPF and pruning and grafting strategies to handle multicasting.
- PIM-SM is a group-shared tree routing protocol that is similar to CBT and uses a rendezvous router as the source of the tree.
- For multicasting between two noncontiguous multicast routers, we make a multicast backbone (MBONE) to enable tunneling.

22.8 PRACTICE SET

Review Questions

1. What is the difference between a direct and an indirect delivery?
2. List three forwarding techniques discussed in this chapter and give a brief description of each.
3. Contrast two different routing tables discussed in this chapter.
4. What is the purpose of RIP?
5. What are the functions of a RIP message?
6. Why is the expiration timer value 6 times that of the periodic timer value?
7. How does the hop count limit alleviate RIP's problems?
8. List RIP shortcomings and their corresponding fixes.
9. What is the basis of classification for the four types of links defined by OSPF?
10. Why do OSPF messages propagate faster than RIP messages?
11. What is the purpose of BGP?
12. Give a brief description of two groups of multicast routing protocols discussed in this chapter.

Exercises

13. Show a routing table for a host that is totally isolated.
14. Show a routing table for a host that is connected to a LAN without being connected to the Internet.
15. Find the topology of the network if Table 22.3 is the routing table for router R1.

Table 22.3 *Routing table for Exercise 15*

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

16. Can router R1 in Figure 22.8 receive a packet with destination address 140.24.7.194? Explain your answer.

17. Can router Rl in Figure 22.8 receive a packet with destination address l40.24.7.42? Explain your answer.
18. Show the routing table for the regional ISP in Figure 22.9.
19. Show the routing table for local ISP 1 in Figure 22.9.
20. Show the routing table for local ISP 2 in Figure 22.9.
21. Show the routing table for local ISP 3 in Figure 22.9.
22. Show the routing table for small ISP 1 in Figure 22.9.
23. Contrast and compare distance vector routing with link state routing.
24. A router has the following RIP routing table:

Net!	4	B
Net2	2	C
Net3	!	F
Net4	5	G

What would be the contents of the table if the router received the following RIP message from router C?

Net!	2
Net2	!
Net3	3
Net4	7

25. How many bytes are empty in a RIP message that advertises *N* networks?
26. A router has the following RIP routing table:

Net!	4	B
Net2	2	C
Net3	1	F
Net4	5	G

Show the response message sent by this router.

27. Show the autonomous system with the following specifications:
    a. There are eight networks (Nl to N8).
    b. There are eight routers (Rl to R8).
    c. Nl, N2, N3, N4, NS, and N6 are Ethernet LANs.
    d. N7 and N8 are point-to-point WANs.
    e. Rl connects Nl and N2.
    f. R2 connects Nl and N7.
    g. R3 connects N2 and N8.
    h. R4 connects N7 and N6.
    i. RS connects N6 and N3.
    j. R6 connects N6 and N4.
    k. R7 connects N6 and NS.
    1. R8 connects N8 and N5.

28. Draw the graphical representation of the autonomous system of Exercise 27 as seen by OSPF.
29. Which of the networks in Exercise 27 is a transient network? Which is a stub network?
30. A router using DVMRP receives a packet with source address 10.14.17.2 from interface 2. If the router forwards the packet, what are the contents of the entry related to this address in the unicast routing table?
31. Does RPF actually create a shortest path tree? Explain.
32. Does RPB actually create a shortest path tree? Explain. What are the leaves of the tree?
33. Does RPM actually create a shortest path tree? Explain. What are the leaves of the tree?

Research Activities

34. If you have access to UNIX (or LINUX), use *netstat* and *ifconfig* to find the routing table for the server to which you are connected.
35. Find out how your ISP uses address aggregation and longest mask match principles.
36. Find out whether your IP address is part of the geographical address allocation.
37. If you are using a router, find the number and names of the columns in the routing table.

PART 5

Transport Layer

Objectives

The transport layer is responsible for process-to-process delivery of the entire message. A process is an application program running on a host. Whereas the network layer oversees source-to-destination delivery of individual packets, it does not recognize any relationship between those packets. It treats each one independently, as though each piece belonged to a separate message, whether or not it does. The transport layer, on the other hand, ensures that the whole message arrives intact and in order, overseeing both error control and flow control at the source-to-destination level.

The transport layer is responsible for the delivery of a message from one process to another.

Computers often run several programs at the same time. For this reason, source-to-destination delivery means delivery not only from one computer to the next but also from a specific process on one computer to a specific process on the other. The transport layer header must therefore include a type of address called a *service-point address* in the OSI model and port number or port addresses in the Internet and TCP/IP protocol suite.

A transport layer protocol can be either connectionless or connection-oriented. A connectionless transport layer treats each segment as an independent packet and delivers it to the transport layer at the destination machine. A connection-oriented transport layer makes a connection with the transport layer at the destination machine first before delivering the packets. After all the data is transferred, the connection is terminated.

In the transport layer, a message is normally divided into transmittable segments. A connectionless protocol, such as UDP, treats each segment separately. A connection-oriented protocol, such as TCP and SCTP, creates a relationship between the segments using sequence numbers.

Like the data link layer, the transport layer may be responsible for flow and error control. However, flow and error control at this layer is performed end to end rather than across a single link. We will see that one of the protocols discussed in this part of

the book, UDP, is not involved in flow or error control. On the other hand, the other two protocols, TCP and SCTP, use sliding windows for flow control and an acknowledgment system for error control.

Part 5 of the book is devoted to the transport layer and the services provided by this layer.

Chapters

This part consists of two chapters: Chapters 23 and 24.

Chapter 23

Chapter 23 discusses three transport layer protocols in the Internet: UDP, TCP, and SCTP. The first, User Datagram Protocol (UDP), is a connectionless, unreliable protocol that is used for its efficiency. The second, Transmission Control Protocol (TCP), is a connection-oriented, reliable protocol that is a good choice for data transfer. The third, Stream Control Transport Protocol (SCTP) is a new transport-layer protocol designed for multimedia applications.

Chapter 24

Chapter 24 discuss two related topics: congestion control and quality of service. Although these two issues can be related to any layer, we discuss them here with some references to other layers.

CHAPTER 23

Process-la-Process Delivery: UDp, TCp, and SCTP

We begin this chapter by giving the rationale for the existence of the transport layer-the need for process-to-process delivery. We discuss the issues arising from this type of delivery, and we discuss methods to handle them.

The Internet model has three protocols at the transport layer: UDP, TCP, and SCTP. First we discuss UDP, which is the simplest of the three. We see how we can use this very simple transport layer protocol that lacks some of the features of the other two.

We then discuss TCP, a complex transport layer protocol. We see how our previously presented concepts are applied to TCP. We postpone the discussion of congestion control and quality of service in TCP until Chapter 24 because these two topics apply to the data link layer and network layer as well.

We finally discuss SCTP, the new transport layer protocol that is designed for multihomed, multistream applications such as multimedia.

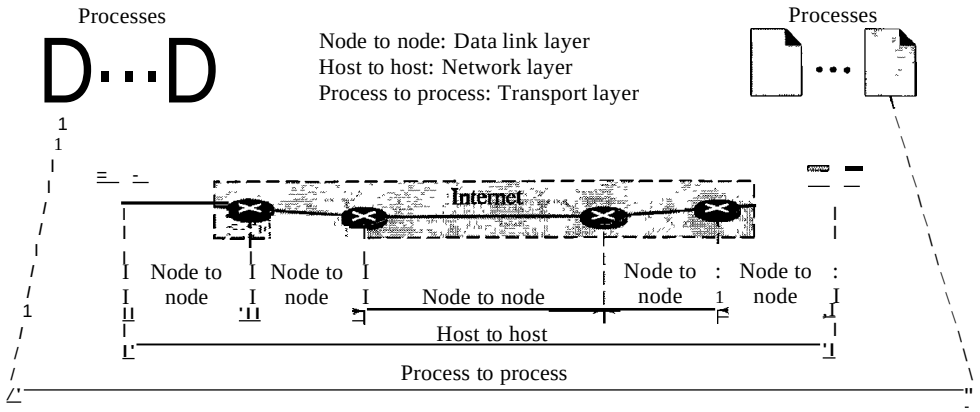
23.1 PROCESS-TO-PROCESS DELIVERY

The data link layer is responsible for delivery of frames between two neighboring nodes over a link. This is called *node-to-node delivery.* The network layer is responsible for delivery of datagrams between two hosts. This is called *host-to-host delivery.* Communication on the Internet is not defined as the exchange of data between two nodes or between two hosts. Real communication takes place between two processes (application programs). We need process-to-process delivery. However, at any moment, several processes may be running on the source host and several on the destination host. To complete the delivery, we need a mechanism to deliver data from one of these processes running on the source host to the corresponding process running on the destination host.

The transport layer is responsible for process-to-process delivery-the delivery of a packet, part of a message, from one process to another. Two processes communicate in a client/server relationship, as we will see later. Figure 23.1 shows these three types of deliveries and their domains.

The transport layer is responsible for process-to-process delivery.

Figure 23.1 *Types ofdata deliveries*



Client/Server Paradigm

Although there are several ways to achieve process-to-process communication, the most common one is through the client/server paradigm. A process on the local host, called a client, needs services from a process usually on the remote host, called a server.

Both processes (client and server) have the same name. For example, to get the day and time from a remote machine, we need a Daytime client process running on the local host and a Daytime server process running on a remote machine.

Operating systems today support both multiuser and multiprogramming environments. A remote computer can run several server programs at the same time, just as local computers can run one or more client programs at the same time. For communication, we must define the following:

1. Local host
2. Local process
3. Remote host
4. Remote process

Addressing

Whenever we need to deliver something to one specific destination among many, we need an address. At the data link layer, we need a MAC address to choose one node among several nodes if the connection is not point-to-point. A frame in the data link layer needs a destination MAC address for delivery and a source address for the next node's reply.

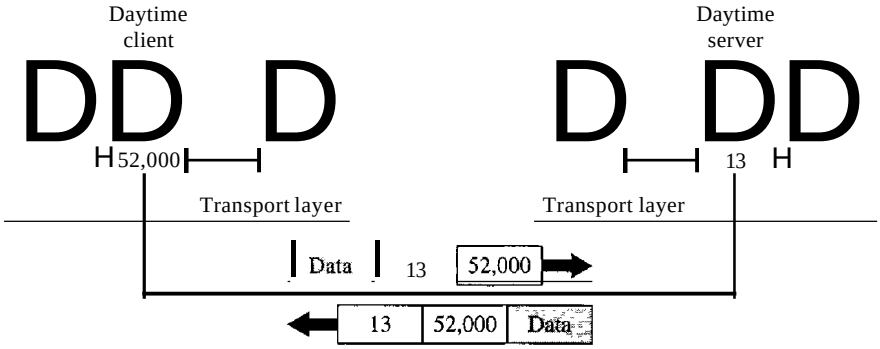
At the network layer, we need an IP address to choose one host among millions. A datagram in the network layer needs a destination IP address for delivery and a source IP address for the destination's reply.

At the transport layer, we need a transport layer address, called a port number, to choose among multiple processes running on the destination host. The destination port number is needed for delivery; the source port number is needed for the reply.

In the Internet model, the port numbers are 16-bit integers between 0 and 65,535. The client program defines itself with a port number, chosen randomly by the transport layer software running on the client host. This is the ephemeral port number.

The server process must also define itself with a port number. This port number, however, cannot be chosen randomly. **If** the computer at the server site runs a server process and assigns a random number as the port number, the process at the client site that wants to access that server and use its services will not know the port number. Of course, one solution would be to send a special packet and request the port number of a specific server, but this requires more overhead. The Internet has decided to use universal port numbers for servers; these are called well-known port numbers. There are some exceptions to this rule; for example, there are clients that are assigned well-known port numbers. Every client process knows the well-known port number of the corresponding server process. For example, while the Daytime client process, discussed above, can use an ephemeral (temporary) port number 52,000 to identify itself, the Daytime server process must use the well-known (permanent) port number 13. Figure 23.2 shows this concept.

Figure 23.2 *Port numbers*



It should be clear by now that the IP addresses and port numbers play different roles in selecting the final destination of data. The destination IP address defines the host among the different hosts in the world. After the host has been selected, the port number defines one of the processes on this particular host (see Figure 23.3).

lANA Ranges

The lANA (Internet Assigned Number Authority) has divided the port numbers into three ranges: well known, registered, and dynamic (or private), as shown in Figure 23.4.

- Well-known ports. The ports ranging from 0 to 1023 are assigned and controlled by lANA. These are the well-known ports.
- Registered ports. The ports ranging from 1024 to 49,151 are not assigned or controlled by lANA. They can only be registered with lANA to prevent duplication.
- Dynamic ports. The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used by any process. These are the ephemeral ports.

Figure 23.3 *IP addresses versus port numbers*

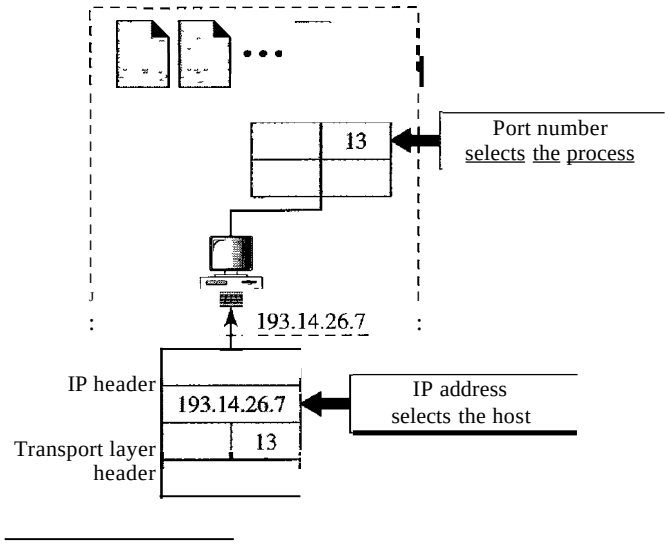
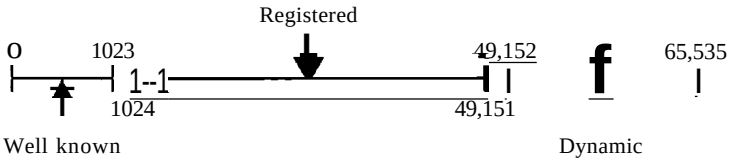


Figure 23.4 *lANA ranges*

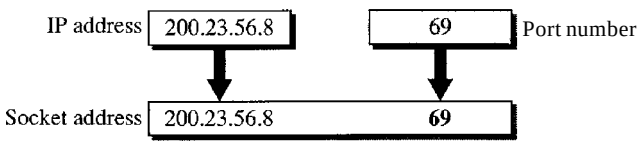


Socket Addresses

Process-to-process delivery needs two identifiers, IP address and the port number, at each end to make a connection. The combination of an IP address and a port number is called a socket address. The client socket address defines the client process uniquely just as the server socket address defines the server process uniquely (see Figure 23.5).

A transport layer protocol needs a pair of socket addresses: the client socket address and the server socket address. These four pieces of information are part of the IP header and the transport layer protocol header. The IP header contains the IP addresses; the UDP or TCP header contains the port numbers.

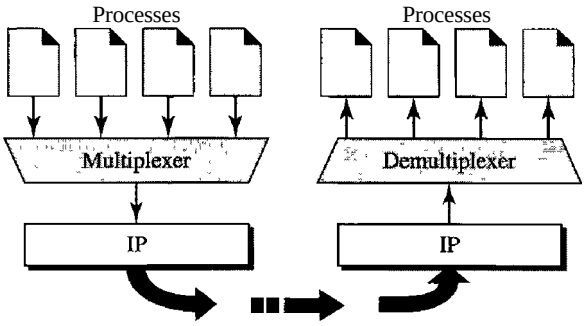
Figure 23.5 *Socket address*



Multiplexing and Demultiplexing

The addressing mechanism allows multiplexing and demultiplexing by the transport layer, as shown in Figure 23.6.

Figure 23.6 *Multiplexing and demultiplexing*



Multiplexing

At the sender site, there may be several processes that need to send packets. However, there is only one transport layer protocol at any time. This is a many-to-one relationship and requires multiplexing. The protocol accepts messages from different processes, differentiated by their assigned port numbers. After adding the header, the transport layer passes the packet to the network layer.

Demultiplexing

At the receiver site, the relationship is one-to-many and requires demultiplexing. The transport layer receives datagrams from the network layer. After error checking and dropping of the header, the transport layer delivers each message to the appropriate process based on the port number.

Connectionless Versus Connection-Oriented Service

A transport layer protocol can either be connectionless or connection-oriented.

Connectionless Service

In a connectionless service, the packets are sent from one party to another with no need for connection establishment or connection release. The packets are not numbered; they may be delayed or lost or may arrive out of sequence. There is no acknowledgment either. We will see shortly that one of the transport layer protocols in the Internet model, UDP, is connectionless.

Connection-Oriented Service

In a connection-oriented service, a connection is first established between the sender and the receiver. Data are transferred. At the end, the connection is released. We will see shortly that TCP and SCTP are connection-oriented protocols.

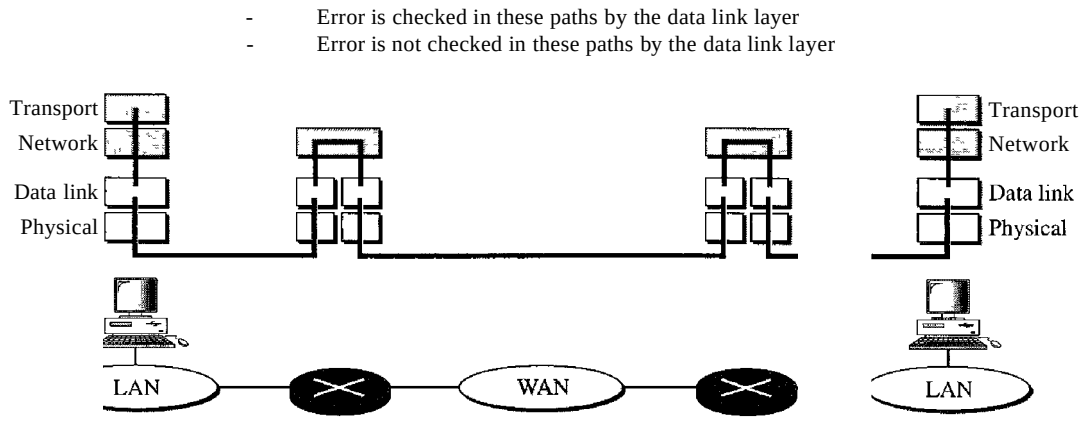
Reliable Versus Unreliable

The transport layer service can be reliable or unreliable. If the application layer program needs reliability, we use a reliable transport layer protocol by implementing flow and error control at the transport layer. This means a slower and more complex service. On the other hand, if the application program does not need reliability because it uses its own flow and error control mechanism or it needs fast service or the nature of the service does not demand flow and error control (real-time applications), then an unreliable protocol can be used.

In the Internet, there are three common different transport layer protocols, as we have already mentioned. UDP is connectionless and unreliable; TCP and SCTP are connection-oriented and reliable. These three can respond to the demands of the application layer programs.

One question often comes to the mind. If the data link layer is reliable and has flow and error control, do we need this at the transport layer, too? The answer is yes. Reliability at the data link layer is between two nodes; we need reliability between two ends. Because the network layer in the Internet is unreliable (best-effort delivery), we need to implement reliability at the transport layer. To understand that error control at the data link layer does not guarantee error control at the transport layer, let us look at Figure 23.7.

Figure 23.7 *Error control*

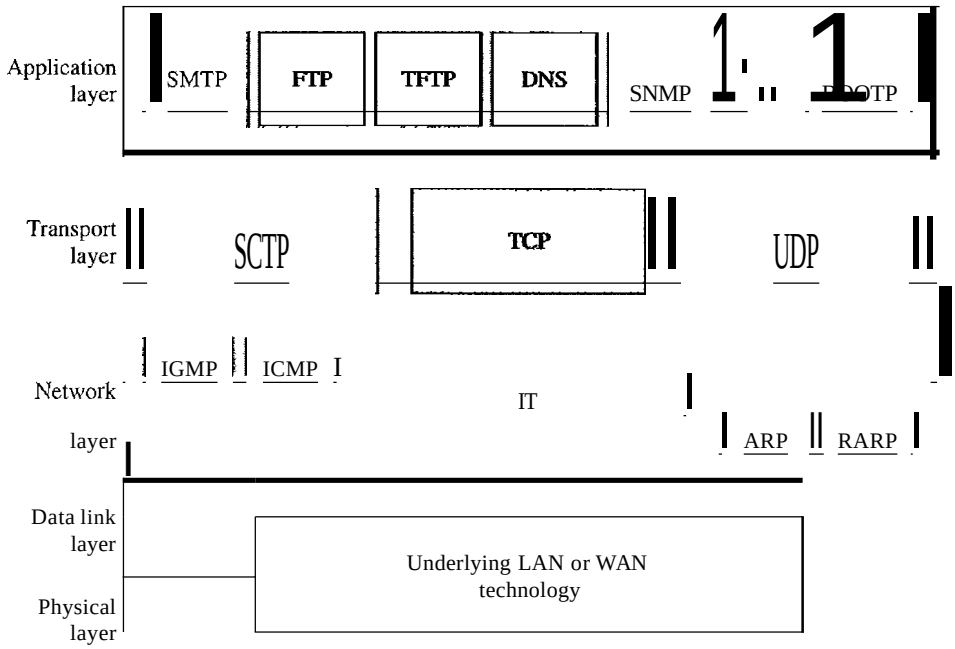


As we will see, flow and error control in TCP is implemented by the sliding window protocol, as discussed in Chapter 11. The window, however, is character-oriented, instead of frame-oriented.

Three Protocols

The original TCP/IP protocol suite specifies two protocols for the transport layer: UDP and TCP. We first focus on UDP, the simpler of the two, before discussing TCP. A new transport layer protocol, SCTP, has been designed, which we also discuss in this chapter. Figure 23.8 shows the position of these protocols in the TCP/IP protocol suite.

Figure 23.8 *Position of UDp, TCp, and SCTP in TCPIIP suite*



23.2 USER DATAGRAM PROTOCOL (UDP)

The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to-process communication instead of host-to-host communication. Also, it performs very limited error checking.

If UDP is so powerless, why would a process want to use it? With the disadvantages come some advantages. UDP is a very simple protocol using a minimum of overhead. If a process wants to send a small message and does not care much about reliability, it can use UDP. Sending a small message by using UDP takes much less interaction between the sender and receiver than using TCP or SCTP.

Well-Known Ports for UDP

Table 23.1 shows some well-known port numbers used by UDP. Some port numbers can be used by both UDP and TCP. We discuss them when we talk about TCP later in the chapter.

Table 23.1 *Well-known ports used with UDP*

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users

Table 23.1 *Well-known ports used with UDP (continued)*

Port	*Protocol*	*Description*
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
III	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

Example 23.1

In UNIX, the well-known ports are stored in a file called fetcfservices. Each line in this file gives the name of the server and the well-known port number. We can use the *grep* utility to extract the line corresponding to the desired application. The following shows the port for FTP. Note that FTP can use port 21 with either UDP or TCP.

```
$grep    ftp    fetclservices
ftp          21ftcp
fip          211udp
```

SNMP uses two port numbers (161 and 162), each for a different purpose, as we will see in Chapter 28.

```
$grep      snmp fetclservices
snmp            161ftcp          #Simple Net  Mgmt Proto
snmp            1611udp          #Simple Net  Mgmt Proto
snmptrap        162/udp          #Traps for SNMP
```

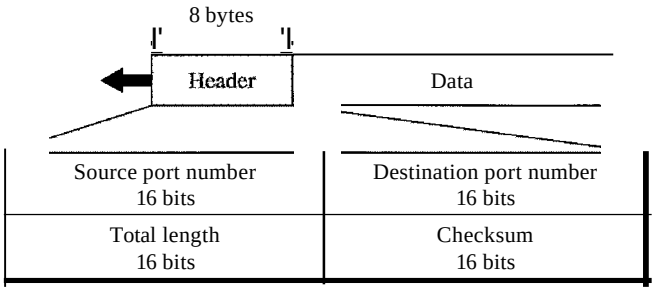
User Datagram

UDP packets, called user datagrams, have a fixed-size header of 8 bytes. Figure 23.9 shows the format of a user datagram.

The fields are as follows:

- Source port number. This is the port number used by the process running on the source host. It is 16 bits long, which means that the port number can range from 0 to 65,535. If the source host is the client (a client sending a request), the port number, in most cases, is an ephemeral port number requested by the process and chosen by the UDP software running on the source host. If the source host is the server (a server sending a response), the port number, in most cases, is a well-known port number.

Figure 23.9 *User datagram format*



- Destination port number. This is the port number used by the process running on the destination host. It is also 16 bits long. If the destination host is the server (a client sending a request), the port number, in most cases, is a well-known port number. If the destination host is the client (a server sending a response), the port number, in most cases, is an ephemeral port number. In this case, the server copies the ephemeral port number it has received in the request packet.
- Length. This is a 16-bit field that defines the total length of the user datagram, header plus data. The 16 bits can define a total length of 0 to 65,535 bytes. However, the total length needs to be much less because a UDP user datagram is stored in an IP datagram with a total length of 65,535 bytes.

  The length field in a UDP user datagram is actually not necessary. A user datagram is encapsulated in an IP datagram. There is a field in the IP datagram that defines the total length. There is another field in the IP datagram that defines the length of the header. So if we subtract the value of the second field from the first, we can deduce the length of a UDP datagram that is encapsulated in an IP datagram.

$$\text{UDP length} = \text{IP length} - \text{IP header's length}$$

  However, the designers of the UDP protocol felt that it was more efficient for the destination UDP to calculate the length of the data from the information provided in the UDP user datagram rather than ask the IP software to supply this information. We should remember that when the IP software delivers the UDP user datagram to the UDP layer, it has already dropped the IP header.
- Checksum. This field is used to detect errors over the entire user datagram (header plus data). The checksum is discussed next.

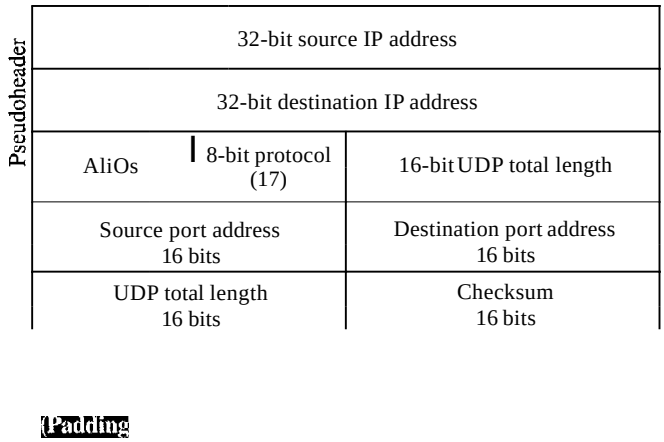
Checksum

We have already talked about the concept of the checksum and the way it is calculated in Chapter 10. We have also shown how to calculate the checksum for the IP and ICMP packet. We now show how this is done for UDP.

The UDP checksum calculation is different from the one for IP and ICMP. Here the checksum includes three sections: a pseudoheader, the UDP header, and the data coming from the application layer.

The pseudoheader is the part of the header of the IP packet in which the user datagram is to be encapsulated with some fields filled with Os (see Figure 23.10).

Figure 23.10 *Pseudoheader for checksum calculation*



If the checksum does not include the pseudoheader, a user datagram may arrive safe and sound. However, if the IP header is corrupted, it may be delivered to the wrong host.

The protocol field is added to ensure that the packet belongs to UDP, and not to other transport-layer protocols. We will see later that if a process can use either UDP or TCP, the destination port number can be the same. The value of the protocol field for UDP is 17. If this value is changed during transmission, the checksum calculation at the receiver will detect it and UDP drops the packet. It is not delivered to the wrong protocol.

Note the similarities between the pseudoheader fields and the last 12 bytes of the IP header.

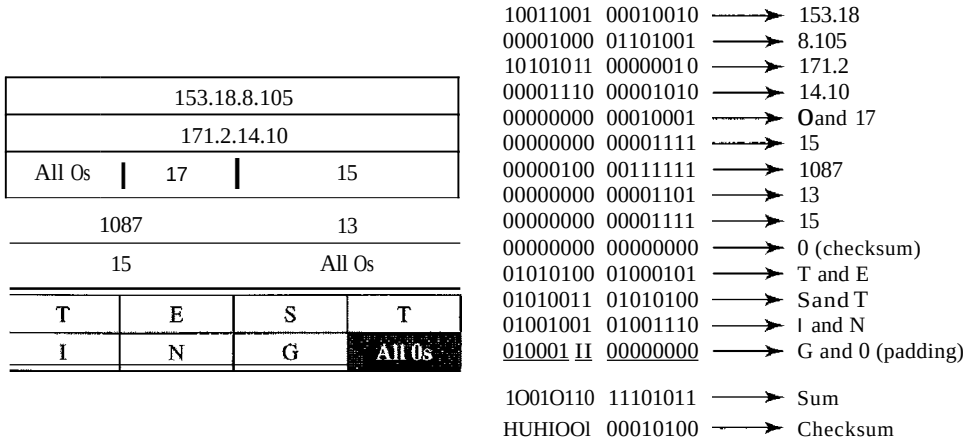
Example 23.2

Figure 23.11 shows the checksum calculation for a very small user datagram with only 7 bytes of data. Because the number of bytes of data is odd, padding is added for checksum calcUlation. The pseudoheader as well as the padding will be dropped when the user datagram is delivered to IP.

Optional Use of the Checksum

The calculation of the checksum and its inclusion in a user datagram are optional. If the checksum is not calculated, the field is filled with 1s. Note that a calculated checksum can never be allIs because this implies that the sum is all Os, which is impossible because it requires that the value of fields to be Os.

Figure 23.11 *Checksum calculation of a simple UDP user datagram*



UDP Operation

UDP uses concepts common to the transport layer. These concepts will be discussed here briefly, and then expanded in the next section on the TCP protocol.

Connectionless Services

As mentioned previously, UDP provides a connectionless service. This means that each user datagram sent by UDP is an independent datagram. There is no relationship between the different user datagrams even if they are coming from the same source process and going to the same destination program. The user datagrams are not numbered. Also, there is no connection establishment and no connection termination, as is the case for TCP. This means that each user datagram can travel on a different path.

One of the ramifications of being connectionless is that the process that uses UDP cannot send a stream of data to UDP and expect UDP to chop them into different related user datagrams. Instead each request must be small enough to fit into one user datagram. Only those processes sending short messages should use UDP.

Flow and Error Control

UDP is a very simple, unreliable transport protocol. There is no flow control and hence no window mechanism. The receiver may overflow with incoming messages.

There is no error control mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded.

The lack of flow control and error control means that the process using UDP should provide these mechanisms.

Encapsulation and Decapsulation

To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages in an IP datagram.