The answer is that hosts not designated to be routers should *not* route datagrams that they receive; they should discard them.

There are four reasons why a host not designated to serve as a router should refrain from performing any router functions. First, when such a host receives a datagram intended for some other machine, something has gone wrong with internet addressing, routing, or delivery. The problem may not be revealed if the host takes corrective action by routing the datagram. Second, routing will cause unnecessary network traffic (and may steal CPU time from legitimate uses of the host). Third, simple errors can cause chaos. Suppose that every host routes traffic, and imagine what happens if one machine accidentally broadcasts a datagram that is destined for some host, *H*. Because it has been broadcast, every host on the network receives a copy of the datagram. Every host forwards its copy to *H*, which will be bombarded with many copies. Fourth, as later chapters show, routers do more than merely route traffic. As the next chapter explains, routers use a special protocol to report errors, while hosts do not (again, to avoid having multiple error reports bombard a source). Routers also propagate routing information to ensure that their routing tables are consistent. If hosts route datagrams without participating fully in all router functions, unexpected anomalies can arise.

8.11 Establishing Routing Tables

We have discussed how IP routes datagrams based on the contents of routing tables, without saying how systems initialize their routing tables or update them as the network changes. Later chapters deal with these questions and discuss protocols that allow routers to keep routes consistent. For now, it is only important to understand that IP software uses the routing table whenever it decides how to forward a datagram, so changing routing tables will change the paths datagrams follow.

8.12 Summary

IP uses routing information to forward datagrams; the computation consists of deciding where to send a datagram based on its destination IP address. Direct delivery is possible if the destination machine lies on a network to which the sending machine attaches; we think of this as the final step in datagram transmission. If the sender cannot reach the destination directly, the sender must forward the datagram to a router. The general paradigm is that hosts send indirectly routed datagrams to the nearest router; the datagrams travel through the internet from router to router until they can be delivered directly across one physical network.

When IP software looks up a route, the algorithm produces the IP address of the next machine (i.e., the address of the next hop) to which the datagram should be sent; IP passes the datagram and next hop address to network interface software. Transmission of a datagram from one machine to the next always involves encapsulating the datagram in a physical frame, mapping the next hop internet address to a physical address, and sending the frame using the underlying hardware.

The internet routing algorithm is table driven and uses only IP addresses. Although it is possible for a routing table to contain a host-specific destination address, most routing tables contain only network addresses, keeping routing tables small. Using a default route can also help keep a routing table small, especially for hosts that can access only one router.

FOR FURTHER STUDY

Routing is an important topic. Frank and Chou [1971] and Schwartz and Stern [1980] discuss routing in general; Postel [1980] discusses internet routing. Braden and Postel [RFC 1009] provides a summary of how Internet routers handle IP datagrams. Narten [1989] contains a survey of Internet routing. Fultz and Kleinrock [1971] analyzes adaptive routing schemes; and McQuillan, Richer, and Rosen [1980] describes the ARPANET adaptive routing algorithm.

The idea of using policy statements to formulate rules about routing has been considered often. Leiner [RFC 1124] considers policies for interconnected networks. Braun [RFC 1104] discusses models of policy routing for internets, Rekhter [RFC 1092] relates policy routing to the second NSFNET backbone, and Clark [RFC 1102] describes using policy routing with IP.

EXERCISES

8.1 Complete routing tables for all routers in Figure 8.2. Which routers will benefit most from using a default route?

8.2 Examine the routing algorithm used on your local system. Are all the cases mentioned in the chapter covered? Does the algorithm allow anything not mentioned?

8.3 What does a router do with the *time to live* value in an IP header?

8.4 Consider a machine with two physical network connections and two IP addresses I_1 and I_2. Is it possible for that machine to receive a datagram destined for I_2 over the network with address I_1? Explain.

8.5 Consider two hosts, A and B, that both attach to a common physical network, N. Is it ever possible, when using our routing algorithm, for A to receive a datagram destined for B? Explain.

8.6 Modify the routing algorithm to accommodate the IP source route options discussed in Chapter 7.

8.7 An IP router must perform a computation that takes time proportional to the length of the datagram header each time it processes a datagram. Explain.

8.8 A network administrator argues that to make monitoring and debugging his local network easier, he wants to rewrite the routing algorithm so it tests host-specific routes *before* it tests for direct delivery. How can he use the revised algorithm to build a network monitor?

8.9 Is it possible to address a datagram to a router's IP address? Does it make sense to do so?

8.10 Consider a modified routing algorithm that examines host-specific routes before testing for delivery on directly connected networks. Under what circumstances might such an algorithm be desirable? undesirable?

8.11 Play detective: after monitoring IP traffic on a local area network for 10 minutes one evening, someone notices that all frames destined for machine *A* carry IP datagrams that have destination equal to *A*'s IP address, while all frames destined for machine *B* carry IP datagrams with destination *not* equal to *B*'s IP address. Users report that both *A* and *B* can communicate. Explain.

8.12 How could you change the IP datagram format to support high-speed packet switching at routers? Hint: a router must recompute a header checksum after decrementing the time-to-live field.

8.13 Compare CLNP, the ISO connectionless delivery protocol (ISO standard 8473) with IP. How well will the ISO protocol support high-speed switching? Hint: variable length fields are expensive.

9

Internet Protocol: Error And Control Messages (ICMP)

9.1 Introduction

The previous chapter shows how the Internet Protocol software provides an unreliable, connectionless datagram delivery service by arranging for each router to forward datagrams. A datagram travels from router to router until it reaches one that can deliver the datagram directly to its final destination. If a router cannot route or deliver a datagram, or if the router detects an unusual condition that affects its ability to forward the datagram (e.g., network congestion), the router needs to inform the original source to take action to avoid or correct the problem. This chapter discusses a mechanism that internet routers and hosts use to communicate such control or error information. We will see that routers use the mechanism to report problems and hosts use it to test whether destinations are reachable.

9.2 The Internet Control Message Protocol

In the connectionless system we have described so far, each router operates autonomously, routing or delivering datagrams that arrive without coordinating with the original sender. The system works well if all machines operate correctly and agree on routes. Unfortunately, no large communication system works correctly all the time. Besides failures of communication lines and processors, IP fails to deliver datagrams when the destination machine is temporarily or permanently disconnected from the network, when the time-to-live counter expires, or when intermediate routers become so

congested that they cannot process the incoming traffic. The important difference between having a single network implemented with dedicated hardware and an internet implemented with software is that in the former, the designer can add special hardware to inform attached hosts when problems arise. In an internet, which has no such hardware mechanism, a sender cannot tell whether a delivery failure resulted from a local malfunction or a remote one. Debugging becomes extremely difficult. The IP protocol itself contains nothing to help the sender test connectivity or learn about such failures.

To allow routers in an internet to report errors or provide information about unexpected circumstances, the designers added a special-purpose message mechanism to the TCP/IP protocols. The mechanism, known as the *Internet Control Message Protocol (ICMP)*, is considered a required part of IP and must be included in every IP implementation.

Like all other traffic, ICMP messages travel across the internet in the data portion of IP datagrams. The ultimate destination of an ICMP message is not an application program or user on the destination machine, however, but the Internet Protocol software on that machine. That is, when an ICMP error message arrives, the ICMP software module handles it. Of course, if ICMP determines that a particular higher-level protocol or application program has caused a problem, it will inform the appropriate module. We can summarize:

> *The Internet Control Message Protocol allows routers to send error or control messages to other routers or hosts; ICMP provides communication between the Internet Protocol software on one machine and the Internet Protocol software on another.*

Initially designed to allow routers to report the cause of delivery errors to hosts, ICMP is not restricted to routers. Although guidelines restrict the use of some ICMP messages, an arbitrary machine can send an ICMP message to any other machine. Thus, a host can use ICMP to correspond with a router or another host. The chief advantage of allowing hosts to use ICMP is that it provides a single mechanism used for all control and information messages.

9.3 Error Reporting vs. Error Correction

Technically, ICMP is an *error reporting mechanism.* It provides a way for routers that encounter an error to report the error to the original source. Although the protocol specification outlines intended uses of ICMP and suggests possible actions to take in response to error reports, ICMP does not fully specify the action to be taken for each possible error. In short,

When a datagram causes an error, ICMP can only report the error condition back to the original source of the datagram; the source must relate the error to an individual application program or take other action to correct the problem.

Most errors stem from the original source, but others do not. Because ICMP reports problems to the original source, however, it cannot be used to inform intermediate routers about problems. For example, suppose a datagram follows a path through a sequence of routers, R_1, R_2, ..., R_k. If R_k has incorrect routing information and mistakenly routes the datagram to router R_E, R_E cannot use ICMP to report the error back to router R_k; ICMP can only send a report back to the original source. Unfortunately, the original source has no responsibility for the problem or control over the misbehaving router. In fact, the source may not be able to determine which router caused the problem.

Why restrict ICMP to communication with the original source? The answer should be clear from our discussion of datagram formats and routing in the previous chapters. A datagram only contains fields that specify the original source and the ultimate destination; it does not contain a complete record of its trip through the internet (except for unusual cases where the record route option is used). Furthermore, because routers can establish and change their own routing tables, there is no global knowledge of routes. Thus, when a datagram reaches a given router, it is impossible to know the path it has taken to arrive there. If the router detects a problem, it cannot know the set of intermediate machines that processed the datagram, so it cannot inform them of the problem. Instead of silently discarding the datagram, the router uses ICMP to inform the original source that a problem has occurred, and trusts that host administrators will cooperate with network administrators to locate and repair the problem.

9.4 ICMP Message Delivery

ICMP messages require two levels of encapsulation as Figure 9.1 shows. Each ICMP message travels across the internet in the data portion of an IP datagram, which itself travels across each physical network in the data portion of a frame. Datagrams carrying ICMP messages are routed exactly like datagrams carrying information for users; there is no additional reliability or priority. Thus, error messages themselves may be lost or discarded. Furthermore, in an already congested network, the error message may cause additional congestion. An exception is made to the error handling procedures if an IP datagram carrying an ICMP message causes an error. The exception, established to avoid the problem of having error messages about error messages, specifies that ICMP messages are not generated for errors that result from datagrams carrying ICMP error messages.

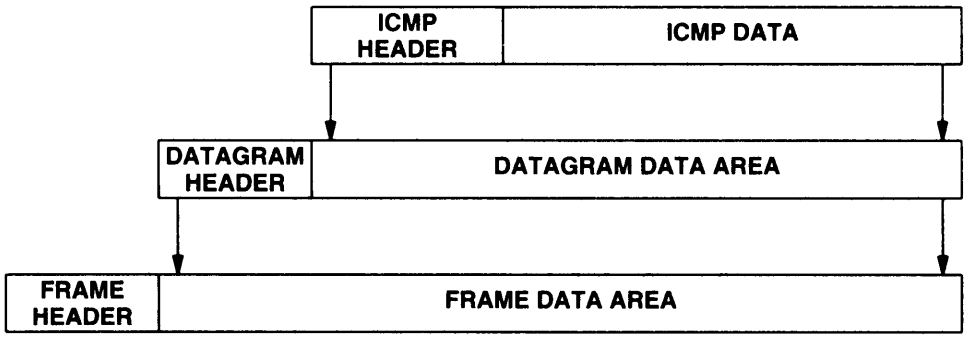


Figure 9.1 Two levels of ICMP encapsulation. The ICMP message is encapsulated in an IP datagram, which is further encapsulated in a frame for transmission. To identify ICMP, the datagram protocol field contains the value *1*.

It is important to keep in mind that even though ICMP messages are encapsulated and sent using IP, ICMP is not considered a higher level protocol — it is a required part of IP. The reason for using IP to deliver ICMP messages is that they may need to travel across several physical networks to reach their final destination. Thus, they cannot be delivered by the physical transport alone.

9.5 ICMP Message Format

Although each ICMP message has its own format, they all begin with the same three fields: an 8-bit integer message *TYPE* field that identifies the message, an 8-bit *CODE* field that provides further information about the message type, and a 16-bit *CHECKSUM* field (ICMP uses the same additive checksum algorithm as IP, but the ICMP checksum only covers the ICMP message). In addition, ICMP messages that report errors always include the header and first 64 data bits of the datagram causing the problem.

The reason for returning more than the datagram header alone is to allow the receiver to determine more precisely which protocol(s) and which application program were responsible for the datagram. As we will see later, higher-level protocols in the TCP/IP suite are designed so that crucial information is encoded in the first 64 bits.

The ICMP *TYPE* field defines the meaning of the message as well as its format. The types include:

Type Field	ICMP Message Type
0	Echo Reply
3	Destination Unreachable
4	Source Quench
5	Redirect (change a route)
8	Echo Request
9	Router Advertisement
10	Router Solicitation
11	Time Exceeded for a Datagram
12	Parameter Problem on a Datagram
13	Timestamp Request
14	Timestamp Reply
15	Information Request (obsolete)
16	Information Reply (obsolete)
17	Address Mask Request
18	Address Mask Reply

The next sections describe each of these messages, giving details of the message format and its meaning.

9.6 Testing Destination Reachability And Status (Ping)

TCP/IP protocols provide facilities to help network managers or users identify network problems. One of the most frequently used debugging tools invokes the ICMP *echo request* and *echo reply* messages. A host or router sends an ICMP echo request message to a specified destination. Any machine that receives an echo request formulates an echo reply and returns it to the original sender. The request contains an optional data area; the reply contains a copy of the data sent in the request. The echo request and associated reply can be used to test whether a destination is reachable and responding. Because both the request and reply travel in IP datagrams, successful receipt of a reply verifies that major pieces of the transport system work. First, IP software on the source computer must route the datagram. Second, intermediate routers between the source and destination must be operating and must route the datagram correctly. Third, the destination machine must be running (at least it must respond to interrupts), and both ICMP and IP software must be working. Finally, all routers along the return path must have correct routes.

On many systems, the command users invoke to send ICMP echo requests is named *ping*†. Sophisticated versions of ping send a series of ICMP echo requests, capture responses, and provide statistics about datagram loss. They allow the user to specify the length of the data being sent and the interval between requests. Less sophisticated versions merely send one ICMP echo request and await a reply.

†Dave Mills once suggested that *PING* is an acronym for *Packet InterNet Groper*.

9.7 Echo Request And Reply Message Format

Figure 9.2 shows the format of echo request and reply messages.

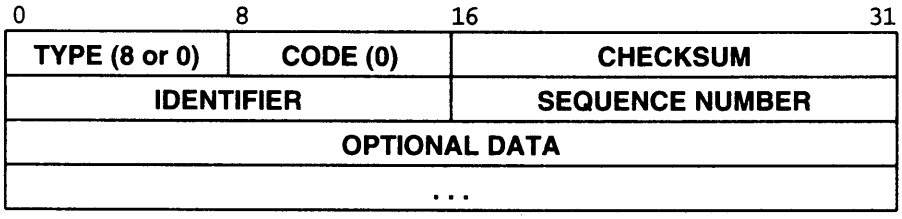


Figure 9.2 ICMP echo request or reply message format.

The field listed as *OPTIONAL DATA* is a variable length field that contains data to be returned to the sender. An echo reply always returns exactly the same data as was received in the request. Fields *IDENTIFIER* and *SEQUENCE NUMBER* are used by the sender to match replies to requests. The value of the *TYPE* field specifies whether the message is a request (*8*) or a reply (*0*).

9.8 Reports Of Unreachable Destinations

When a router cannot forward or deliver an IP datagram, it sends a *destination unreachable* message back to the original source, using the format shown in Figure 9.3.

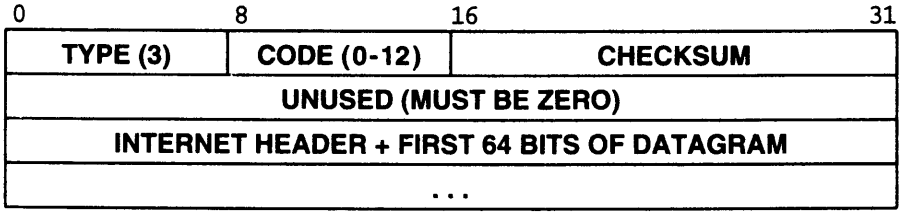


Figure 9.3 ICMP destination unreachable message format.

The *CODE* field in a destination unreachable message contains an integer that further describes the problem. Possible values are:

Code Value	Meaning
0	Network unreachable
1	Host unreachable
2	Protocol unreachable
3	Port unreachable
4	Fragmentation needed and DF set
5	Source route failed
6	Destination network unknown
7	Destination host unknown
8	Source host isolated
9	Communication with destination network administratively prohibited
10	Communication with destination host administratively prohibited
11	Network unreachable for type of service
12	Host unreachable for type of service

Although IP is a best-effort delivery mechanism, discarding datagrams should not be taken lightly. Whenever an error prevents a router from routing or delivering a datagram, the router sends a destination unreachable message back to the source and then *drops* (i.e., discards) the datagram. Network unreachable errors usually imply routing failures; host unreachable errors imply delivery failures†. Because the ICMP error message contains a short prefix of the datagram that caused the problem, the source will know exactly which address is unreachable.

Destinations may be unreachable because hardware is temporarily out of service, because the sender specified a nonexistent destination address, or (in rare circumstances) because the router does not have a route to the destination network. Note that although routers report failures they encounter, they may not know of all delivery failures. For example, if the destination machine connects to an Ethernet network, the network hardware does not provide acknowledgements. Therefore, a router can continue to send packets to a destination after the destination is powered down without receiving any indication that the packets are not being delivered. To summarize:

> *Although a router sends a destination unreachable message when it encounters a datagram that cannot be forwarded or delivered, a router cannot detect all such errors.*

The meaning of protocol and port unreachable messages will become clear when we study how higher level protocols use abstract destination points called *ports*. Most of the remaining messages are self explanatory. If the datagram contains the source route option with an incorrect route, it may trigger a *source route* failure message. If a router needs to fragment a datagram but the "don't fragment" bit is set, the router sends a *fragmentation needed* message back to the source.

†An exception occurs for routers using the subnet addressing scheme of Chapter 10. They report a subnet routing failure with an ICMP host unreachable message.

9.9 Congestion And Datagram Flow Control

Because IP is connectionless, a router cannot reserve memory or communication resources in advance of receiving datagrams. As a result, routers can be overrun with traffic, a condition known as *congestion*. It is important to understand that congestion can arise for two entirely different reasons. First, a high-speed computer may be able to generate traffic faster than a network can transfer it. For example, imagine a supercomputer generating internet traffic. The datagrams may eventually need to cross a slower-speed wide area network (WAN) even though the supercomputer itself attaches to a high-speed local area net. Congestion will occur in the router that attaches the LAN to the WAN because datagrams arrive faster than they can be sent. Second, if many computers simultaneously need to send datagrams through a single router, the router can experience congestion, even though no single source causes the problem.

When datagrams arrive too quickly for a host or router to process, it enqueues them in memory temporarily. If the datagrams are part of a small burst, such buffering solves the problem. If the traffic continues, the host or router eventually exhausts memory and must discard additional datagrams that arrive. A machine uses ICMP *source quench* messages to report congestion to the original source. A source quench message is a request for the source to reduce its current rate of datagram transmission. Usually, congested routers send one source quench message for every datagram that they discard. Routers may also use more sophisticated congestion control techniques. Some monitor incoming traffic and quench sources that have the highest datagram transmission rates. Others attempt to avoid congestion altogether by arranging to send quench requests as their queues start to become long, but before they overflow.

There is no ICMP message to reverse the effect of a source quench. Instead, a host that receives source quench messages for a destination, *D*, lowers the rate at which it sends datagrams to *D* until it stops receiving source quench messages; it then gradually increases the rate as long as no further source quench requests are received.

9.10 Source Quench Format

In addition to the usual ICMP *TYPE*, *CODE*, *CHECKSUM* fields, and an unused 32-bit field, source quench messages have a field that contains a datagram prefix. Figure 9.4 illustrates the format. As with most ICMP messages that report an error, the datagram prefix field contains a prefix of the datagram that triggered the source quench request.

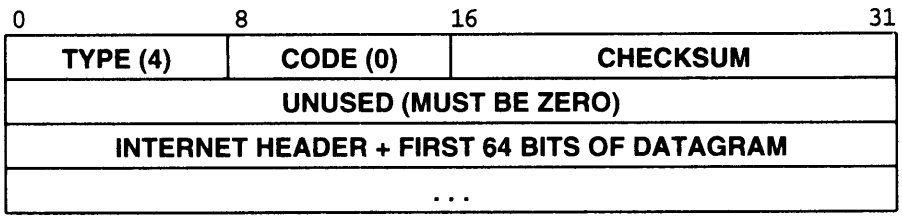


Figure 9.4 ICMP source quench message format. A congested router sends one source quench message each time it discards a datagram; the datagram prefix identifies the datagram that was dropped.

9.11 Route Change Requests From Routers

Internet routing tables usually remain static over long periods of time. Hosts initialize them from a configuration file at system startup, and system administrators seldom make routing changes during normal operations. If the network topology changes, routing tables in a router or host may become incorrect. A change can be temporary (e.g., when hardware needs to be repaired) or permanent (e.g., when a new network is added to the internet). As we will see in later chapters, routers exchange routing information periodically to accommodate network changes and keep their routes up-to-date. Thus, as a general rule:

> *Routers are assumed to know correct routes; hosts begin with minimal routing information and learn new routes from routers.*

To help follow this rule and to avoid duplicating routing information in the configuration file on each host, the initial host route configuration specifies the minimum possible routing information needed to communicate (e.g., the address of a single router). Thus, the host begins with minimal information and relies on routers to update its routing table. In one special case, when a router detects a host using a nonoptimal route, it sends the host an ICMP message, called a *redirect*, requesting that the host change its route. The router also forwards the original datagram on to its destination.

The advantage of the ICMP redirect scheme is simplicity: it allows a host to boot knowing the address of only one router on the local network. The initial router returns ICMP redirect messages whenever a host sends a datagram for which there is a better route. The host routing table remains small but still contains optimal routes for all destinations in use.

Redirect messages do not solve the problem of propagating routes in a general way, however, because they are limited to interactions between a router and a host on a directly connected network. Figure 9.5 illustrates the limitation. In the figure, assume source S sends a datagram to destination D. Assume that router R_1 incorrectly routes the datagram through router R_2 instead of through router R_4 (i.e., R_1 incorrectly chooses

a longer path than necessary). When router R_5 receives the datagram, it cannot send an ICMP redirect message to R_1 because it does not know R_1's address. Later chapters explore the problem of how to propagate routes across multiple networks.

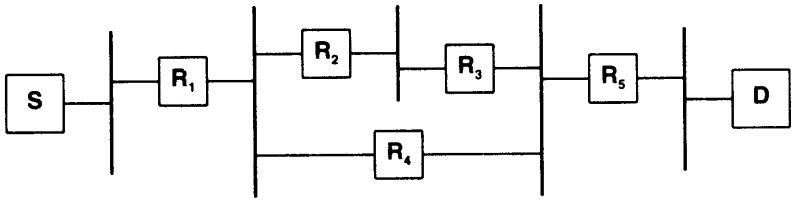


Figure 9.5 ICMP redirect messages do not provide routing changes among routers. In this example, router R_5 cannot redirect R_1 to use the shorter path for datagrams from *S* to *D*.

In addition to the requisite *TYPE*, *CODE*, and *CHECKSUM* fields, each redirect message contains a 32-bit *ROUTER INTERNET ADDRESS* field and an *INTERNET HEADER* field, as Figure 9.6 shows.

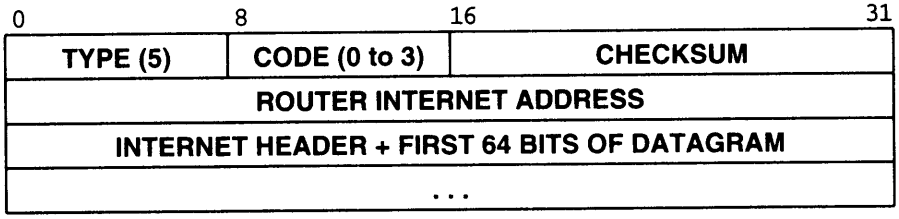


Figure 9.6 ICMP redirect message format.

The *ROUTER INTERNET ADDRESS* field contains the address of a router that the host is to use to reach the destination mentioned in the datagram header. The *INTERNET HEADER* field contains the IP header plus the next 64 bits of the datagram that triggered the message. Thus, a host receiving an ICMP redirect examines the datagram prefix to determine the datagram's destination address. The *CODE* field of an ICMP redirect message further specifies how to interpret the destination address, based on values assigned as follows:

Code Value	Meaning
0	Redirect datagrams for the Net (now obsolete)
1	Redirect datagrams for the Host
2	Redirect datagrams for the Type of Service† and Net
3	Redirect datagrams for the Type of Service and Host

As a general rule, routers only send ICMP redirect requests to hosts and not to other routers. We will see in later chapters that routers use other protocols to exchange routing information.

9.12 Detecting Circular Or Excessively Long Routes

Because internet routers compute a next hop using local tables, errors in routing tables can produce a *routing cycle* for some destination, *D*. A routing cycle can consist of two routers that each route a datagram for destination *D* to the other, or it can consist of several routers. When several routers form a cycle, they each route a datagram for destination *D* to the next router in the cycle. If a datagram enters a routing cycle, it will pass around the cycle endlessly. As mentioned previously, to prevent datagrams from circling forever in a TCP/IP internet, each IP datagram contains a time-to-live counter, sometimes called a *hop count*. A router decrements the time-to-live counter whenever it processes the datagram and discards the datagram when the count reaches zero.

Whenever a router discards a datagram because its hop count has reached zero or because a timeout occurred while waiting for fragments of a datagram, it sends an ICMP *time exceeded* message back to the datagram's source, using the format shown in Figure 9.7.

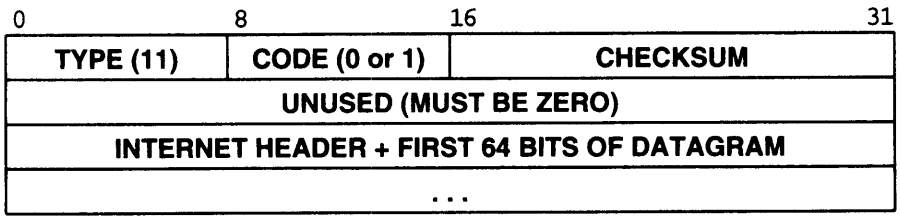


Figure 9.7 ICMP time exceeded message format. A router sends this message whenever a datagram is discarded because the time-to-live field in the datagram header has reached zero or because its reassembly timer expired while waiting for fragments.

ICMP uses the *CODE* field in each time exceeded message (value zero or one) to explain the nature of the timeout being reported:

†Recall that each IP header specifies a type of service used for routing.

Code Value	Meaning
0	Time-to-live count exceeded
1	Fragment reassembly time exceeded

Fragment reassembly refers to the task of collecting all the fragments from a datagram. When the first fragment of a datagram arrives, the receiving host starts a timer and considers it an error if the timer expires before all the pieces of the datagram arrive. Code value *1* is used to report such errors to the sender; one message is sent for each such error.

9.13 Reporting Other Problems

When a router or host finds problems with a datagram not covered by previous ICMP error messages (e.g., an incorrect datagram header), it sends a *parameter problem* message to the original source. One possible cause of such problems occurs when arguments to an option are incorrect. The message, formatted as shown in Figure 9.8, is only sent when the problem is so severe that the datagram must be discarded.

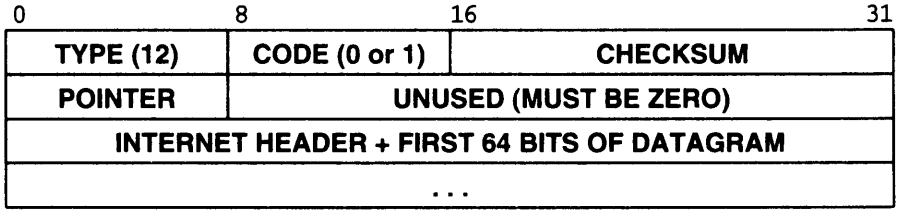


Figure 9.8 ICMP parameter problem message format. Such messages are only sent when the problem causes the datagram to be dropped.

To make the message unambiguous, the sender uses the *POINTER* field in the message header to identify the octet in the datagram that caused the problem. Code *1* is used to report that a required option is missing (e.g., a security option in the military community); the *POINTER* field is not used for code *1*.

9.14 Clock Synchronization And Transit Time Estimation

Although machines on an internet can communicate, they usually operate independently, with each machine maintaining its own notion of the current time. Clocks that differ widely can confuse users of distributed systems software. The TCP/IP protocol suite includes several protocols that can be used to synchronize clocks. One of the simplest techniques uses an ICMP message to obtain the time from another machine. A re-

questing machine sends an ICMP *timestamp request* message to another machine, asking that the second machine return its current value for the time of day. The receiving machine returns a *timestamp reply* back to the machine making the request. Figure 9.9 shows the format of timestamp request and reply messages.

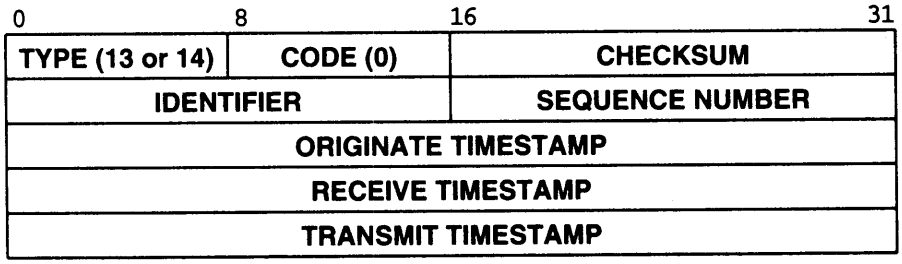


Figure 9.9 ICMP timestamp request or reply message format.

The *TYPE* field identifies the message as a request (*13*) or a reply (*14*); the *IDENTIFIER* and *SEQUENCE NUMBER* fields are used by the source to associate replies with requests. Remaining fields specify times, given in milliseconds since midnight, Universal Time†. The *ORIGINATE TIMESTAMP* field is filled in by the original sender just before the packet is transmitted, the *RECEIVE TIMESTAMP* field is filled immediately upon receipt of a request, and the *TRANSMIT TIMESTAMP* field is filled immediately before the reply is transmitted.

Hosts use the three timestamp fields to compute estimates of the delay time between them and to synchronize their clocks. Because the reply includes the *ORIGINATE TIMESTAMP* field, a host can compute the total time required for a request to travel to a destination, be transformed into a reply, and return. Because the reply carries both the time at which the request entered the remote machine, as well as the time at which the reply left, the host can compute the network transit time, and from that, estimate the differences in remote and local clocks.

In practice, accurate estimation of round-trip delay can be difficult and substantially restricts the utility of ICMP timestamp messages. Of course, to obtain an accurate estimate of round trip delay, one must take many measurements and average them. However, the round-trip delay between a pair of machines that connect to a large internet can vary dramatically, even over short periods of time. Furthermore, recall that because IP is a best-effort technology, datagrams can be dropped, delayed, or delivered out of order. Thus, merely taking many measurements may not guarantee consistency; sophisticated statistical analysis is needed to produce precise estimates.

† Universal Time was formerly called Greenwich Mean Time; it is the time of day at the prime meridian.

9.15 Information Request And Reply Messages

The ICMP *information request* and *information reply* messages (types *15* and *16*) are now considered obsolete and should not be used. They were originally intended to allow hosts to discover their internet address at system startup. The current protocols for address determination are RARP, described in Chapter 6, and BOOTP, described in Chapter 23.

9.16 Obtaining A Subnet Mask

Chapter 10 discusses the motivation for subnet addressing as well as the details of how subnets operate. For now, it is only important to understand that when hosts use subnet addressing, some bits in the hostid portion of their IP address identify a physical network. To participate in subnet addressing, a host needs to know which bits of the 32-bit internet address correspond to the physical network and which correspond to host identifiers. The information needed to interpret the address is represented in a 32-bit quantity called the *subnet mask.*

To learn the subnet mask used for the local network, a machine can send an *address mask request* message to a router and receive an *address mask reply.* The machine making the request can either send the message directly, if it knows the router's address, or broadcast the message if it does not. Figure 9.10 shows the format of address mask messages.

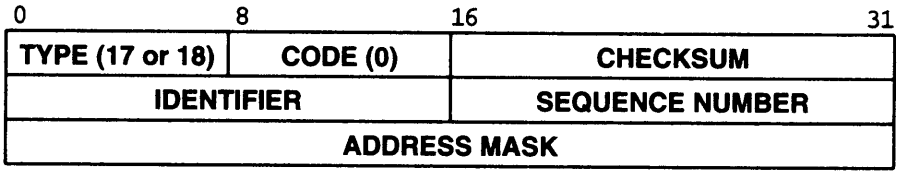


Figure 9.10 ICMP address mask request or reply message format. Usually, hosts broadcast a request without knowing which specific router will respond.

The *TYPE* field in an address mask message specifies whether the message is a request (*17*) or a reply (*18*). A reply contains the network's subnet address mask in the *ADDRESS MASK* field. As usual, the *IDENTIFIER* and *SEQUENCE NUMBER* fields allow a machine to associate replies with requests.

9.17 Router Discovery

After a host boots, it must learn the address of at least one router on the local network before it can send datagrams to destinations on other networks. ICMP supports a *router discovery* scheme that allows a host to discover a router address.

ICMP router discovery is not the only mechanism a host can use to find a router address. The BOOTP and DHCP protocols described in Chapter 23 provide the main alternative — each of the protocols provides a way for a host to obtain the address of a default router along with other bootstrap information. However, BOOTP and DHCP have a serious deficiency: the information they return comes from a database that network administrators configure manually. Thus, the information cannot change quickly.

Of course, static router configuration does work well in some situations. For example, consider a network that has only a single router connecting it to the rest of the Internet. There is no need for a host on such a network to dynamically discover routers or change routes. However, if a network has multiple routers connecting it to the rest of the Internet, a host that obtains a default route at startup can lose connectivity if a single router crashes. More important, the host cannot detect the crash.

The ICMP router discovery scheme helps in two ways. First, instead of providing a statically configured router address via a bootstrap protocol, the scheme allows a host to obtain information directly from the router itself. Second, the mechanism uses a *soft state* technique with timers to prevent hosts from retaining a route after a router crashes — routers advertise their information periodically, and a host discards a route if the timer for a route expires.

Figure 9.11 illustrates the format of the advertisement message a router sends.

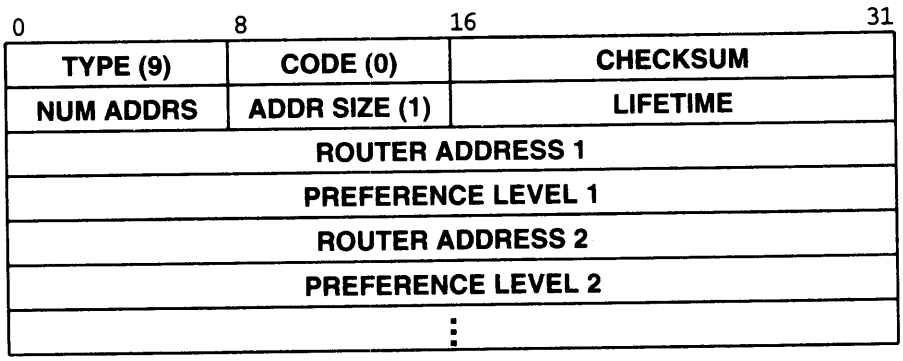


Figure 9.11 ICMP router advertisement message format used with IPv4. Routers send these messages periodically.

Besides the *TYPE*, *CODE*, and *CHECKSUM* fields, the message contains a field labeled *NUM ADDRS* that specifies the number of address entries which follow (often 1), an *ADDR SIZE* field that specifies the size of an address in 32-bit units (1 for IPv4

addresses), and a *LIFETIME* field that specifies the time in seconds a host may use the advertised address(es). The default value for *LIFETIME* is 30 minutes, and the default value for periodic retransmission is 10 minutes, which means that a host will not discard a route if the host misses a single advertisement message.

The remainder of the message consists of *NUM ADDRS* pairs of fields, where each pair contains a *ROUTER ADDRESS* and an integer *PRECEDENCE LEVEL* for the route. The precedence value is a two's complement integer; a host chooses the route with highest precedence.

If the router and the network support multicast as described in Chapter 17, a router multicasts ICMP router advertisement messages to the all-systems multicast address (i.e., 224.0.0.1). If not, the router sends the messages to the limited broadcast address (i.e., the all 1's address). Of course, a host must never send a router advertisement message.

9.18 Router Solicitation

Although the designers provided a range of values to be used as the delay between successive router advertisements, they chose the default of 10 minutes. The value was selected as a compromise between rapid failure detection and low overhead. A smaller value would allow more rapid detection of router failure, but would increase network traffic; a larger value would decrease traffic, but would delay failure detection. One of the issues the designers considered was how to accommodate a large number of routers on the same network.

From the point of view of a host, the default delay has a severe disadvantage: a host cannot afford to wait many minutes for an advertisement when it first boots. To avoid such delays, the designers included an *ICMP router solicitation message* that allows a host to request an immediate advertisement. Figure 9.12 illustrates the message format.

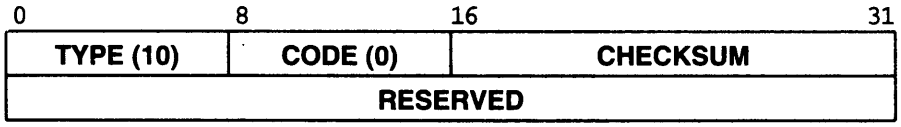


Figure 9.12 ICMP router solicitation message. A host sends a solicitation after booting to request that routers on the local net immediately respond with an ICMP router advertisement.

If a host supports multicasting, the host sends the solicitation to the all-routers multicast address (i.e., 224.0.0.2); otherwise the host sends the solicitation to the limited broadcast address (i.e., the all 1's address). The arrival of a solicitation message causes a router to send a normal router advertisement. As the figure shows, the solicitation does not need to carry information beyond the *TYPE*, *CODE*, and *CHECKSUM* fields.

9.19 Summary

Normal communication across an internet involves sending messages from an application on one host to an application on another host. Routers may need to communicate directly with the network software on a particular host to report abnormal conditions or to send the host new routing information.

The Internet Control Message Protocol provides for extranormal communication among routers and hosts; it is an integral, required part of IP. ICMP includes *source quench* messages that retard the rate of transmission, *redirect* messages that request a host to change its routing table, *echo request/reply* messages that hosts can use to determine whether a destination can be reached, and *router solicitation* and *advertisement* messages that hosts use to dynamically maintain a default route. An ICMP message travels in the data area of an IP datagram and has three fixed-length fields at the beginning of the message: an ICMP message *type* field, a *code* field, and an ICMP *checksum* field. The message type determines the format of the rest of the message as well as its meaning.

FOR FURTHER STUDY

Both Tanenbaum [1981] and Stallings [1985] discuss control messages in general and relate them to various network protocols. The central issue is not how to send control messages but when. Grange and Gien [1979], as well as Driver, Hopewell, and Iaquinto [1979], concentrate on a problem for which control messages are essential, namely, flow control. Gerla and Kleinrock [1980] compares flow control strategies analytically. For a discussion of clock synchronization protocols see Mills [RFCs 956, 957, and 1305].

The Internet Control Message Protocol described here is a TCP/IP standard defined by Postel [RFC 792] and updated by Braden [RFC [1122]. Nagle [RFC 896] discusses ICMP source quench messages and shows how routers should use them to handle congestion control. Prue and Postel [RFC 1016] discusses a more recent technique routers use in response to source quench. Nagle [1987] argues that congestion is always a concern in packet switched networks. Mogul and Postel [RFC 950] discusses subnet mask request and reply messages, and Deering [RFC 1256] discusses the solicitation and advertisement messages used in router discovery. Jain, Ramakrishnan and Chiu [1987] considers how routers and transport protocols could cooperate to avoid congestion.

EXERCISES

9.1 Devise an experiment to record how many of each ICMP message type appear on your local network during a day.

9.2 Experiment to see if you can send packets through a router fast enough to trigger an ICMP source quench message.

9.3 Devise an algorithm that synchronizes clocks using ICMP timestamp messages.

9.4 See if your local computer system contains a *ping* command. How does the program interface with protocols in the operating system? In particular, does the mechanism allow an arbitrary user to create a *ping* program, or does such a program require special privilege? Explain.

9.5 Assume that all routers send ICMP time-exceeded messages, and that your local TCP/IP software will return such messages to an application program. Use the facility to build a *traceroute* command that reports the list of routers between the source and a particular destination.

9.6 If you connect to the global Internet, try to ping host 128.10.2.1 (a machine at Purdue).

9.7 Should a router give ICMP messages priority over normal traffic? Why or why not?

9.8 Consider an Ethernet that has one conventional host, *H*, and 12 routers connected to it. Find a single (slightly illegal) frame carrying an IP packet that, when sent by host *H*, causes *H* to receive exactly 24 packets.

9.9 Compare ICMP source quench packets with Jain's 1-bit scheme used in DECNET. Which is a more effective strategy for dealing with congestion? Why?

9.10 There is no ICMP message that allows a machine to inform the source that transmission errors are causing datagrams to arrive corrupted. Explain why.

9.11 In the previous question, under what circumstances might such a message be useful?

9.12 Should ICMP error messages contain a timestamp that specifies when they are sent? Why or why not?

9.13 If routers at your site participate in ICMP router discovery, find out how many addresses each router advertises on each interface.

9.14 Try to reach a server on a nonexistent host on your local network. Also try to communicate with a nonexistent host on a remote network. In which case do you receive an error message? Why?

9.15 Try using *ping* with a network broadcast address. How many computers answer? Read the protocol documents to determine whether answering a broadcast request is required, recommended, not recommended, or prohibited.

10

Classless And Subnet Address Extensions (CIDR)

10.1 Introduction

Chapter 4 discusses the original Internet addressing scheme and presents the three primary forms of IP addresses. This chapter examines five extensions of the IP address scheme all designed to conserve network prefixes. The chapter considers the motivation for each extension and describes the basic mechanisms used. In particular, it presents the details of the address subnet scheme that is now part of the TCP/IP standards, and the classless address scheme that is an elective standard.

10.2 Review Of Relevant Facts

Chapter 4 discusses addressing in internetworks and presents the fundamentals of the IP address scheme. We said that the 32-bit addresses are carefully assigned to make the IP addresses of all hosts on a given physical network share a common prefix. In the original IP address scheme, designers thought of the common prefix as defining the network portion of an internet address and the remainder as a host portion. The consequence of importance to us is:

> *In the original IP addressing scheme, each physical network is assigned a unique network address; each host on a network has the network address as a prefix of the host's individual address.*

The chief advantage of dividing an IP address into two parts arises from the size of the routing tables required in routers. Instead of keeping one routing entry per destination host, a router can keep one routing entry per network, and examine only the network portion of a destination address when making routing decisions.

Recall that the original IP addressing scheme accommodated diverse network sizes by dividing host addresses into three primary classes. Networks assigned class *A* addresses partition the 32 bits into an 8-bit network portion and a 24-bit host portion. Class *B* addresses partition the 32 bits into 16-bit network and host portions, while class *C* partitions the address into a 24-bit network portion and an 8-bit host portion.

To understand some of the address extensions in this chapter, it will be important to realize that individual sites have the freedom to modify addresses and routes as long as the modifications remain invisible to other sites. That is, a site can choose to assign and use IP addresses in unusual ways internally as long as:

- All hosts and routers at the site agree to honor the site's addressing scheme.
- Other sites on the Internet can treat addresses as a network prefix and a host suffix.

10.3 Minimizing Network Numbers

The original classful IP addressing scheme seems to handle all possibilities, but it has a minor weakness. How did the weakness arise? What did the designers fail to envision? The answer is simple: growth. Because they worked in a world of expensive mainframe computers, the designers envisioned an internet with hundreds of networks and thousands of hosts. They did not foresee tens of thousands of small networks of personal computers that would suddenly appear in the decade after TCP/IP was designed.

Growth has been most apparent in the connected Internet, where the size has been doubling every nine to fifteen months. The large population of networks with trivial size stresses the entire Internet design because it means (1) immense administrative overhead is required merely to manage network addresses, (2) the routing tables in routers are extremely large, and (3) the address space will eventually be exhausted†. The second problem is important because it means that when routers exchange information from their routing tables, the load on the Internet is high, as is the computational effort required in participating routers. The third problem is crucial because the original address scheme could not accommodate the number of networks currently in the global Internet. In particular, insufficient class B prefixes exist to cover all the medium-size networks in the Internet. So the question is, "How can one minimize the number of assigned network addresses, especially class B, without abandoning the 32-bit addressing scheme?"

To minimize the number of addresses used, we must avoid assigning network prefixes whenever possible, and the same IP network prefix must be shared by multiple physical networks. To minimize the use of class B addresses, class C addresses must be used instead. Of course, the routing procedures must be modified, and all machines that connect to the affected networks must understand the conventions used.

†Although there were many predictions that the IPv4 address space would be exhausted before the year 2000, it now appears that with careful allocation and the techniques described in this chapter, IPv4 addresses will suffice until around the year 2019.

The idea of sharing one network address among multiple physical networks is not new and has taken several forms. We will examine three: transparent routers, proxy ARP, and standard IP subnets. In addition, we will explore anonymous point-to-point networks, a special case in which no network prefix needs to be assigned. Finally, we will consider classless addressing, which abandons the rigid class system and allows the address space to be divided in arbitrary ways.

10.4 Transparent Routers

The *transparent router* scheme is based on the observation that a network assigned a class *A* IP address can be extended through a simple trick illustrated in Figure 10.1.

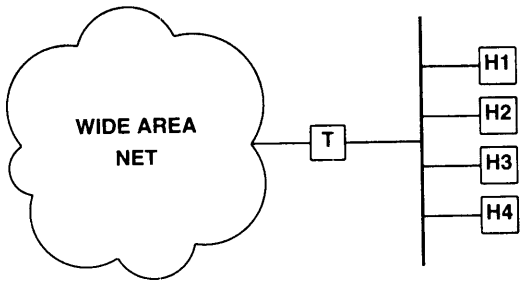


Figure 10.1 Transparent router *T* extending a wide area network to multiple hosts at a site. Each host appears to have an IP address on the WAN.

The trick consists of arranging for a physical network, usually a WAN, to multiplex several host connections through a single host port. As Figure 10.1 shows, a special purpose router, *T*, connects the single host port from the wide area net to a local area network. *T* is called a *transparent router* because other hosts and routers on the WAN do not know it exists.

The local area network does not have its own IP prefix; hosts attached to it are assigned addresses as if they connected directly to the WAN. The transparent router demultiplexes datagrams that arrive from the WAN by sending them to the appropriate host (e.g., by using a table of addresses). The transparent router also accepts datagrams from hosts on the local area network and routes them across the WAN toward their destination.

To make demultiplexing efficient, transparent routers often divide the IP address into multiple parts and encode information in unused parts. For example, the ARPANET was assigned class *A* network address *10.0.0.0*. Each packet switch node (PSN) on the ARPANET had a unique integer address. Internally, the ARPANET treated any 4-octet IP address of the form *10.p.u.i* as four separate octets that specify a

network (*10*), a specific port on the destination PSN (*p*), and a destination PSN (*i*). Octet *u* remained uninterpreted. Thus, the ARPANET addresses *10.2.5.37* and *10.2.9.37* both refer to host *2* on PSN *37*. A transparent router connected to PSN *37* on port *2* can use octet *u* to decide which real host should receive a datagram. The WAN itself need not be aware of the multiple hosts that lie beyond the PSN.

Transparent routers have advantages and disadvantages when compared to conventional routers. The chief advantage is that they require fewer network addresses because the local area network does not need a separate IP prefix. Another is that they can support load balancing. That is, if two transparent routers connect to the same local area network, traffic to hosts on that network can be split between them. By comparison, conventional routers can only advertise one route to a given network.

One disadvantage of transparent routers is that they only work with networks that have a large address space from which to choose host addresses. Thus, they work best with class *A* networks, and they do not work well with class *C* networks. Another disadvantage is that because they are not conventional routers, transparent routers do not provide all the same services as standard routers. In particular, transparent routers may not participate fully in ICMP or network management protocols like SNMP. Therefore, they do not return ICMP echo requests (i.e., one cannot easily "ping" a transparent router to determine if it is operating).

10.5 Proxy ARP

The terms *proxy ARP*, *promiscuous ARP*, and *the ARP hack* refer to a second technique used to map a single IP network prefix into two physical addresses. The technique, which only applies to networks that use ARP to bind internet addresses to physical addresses, can best be explained with an example. Figure 10.2 illustrates the situation.

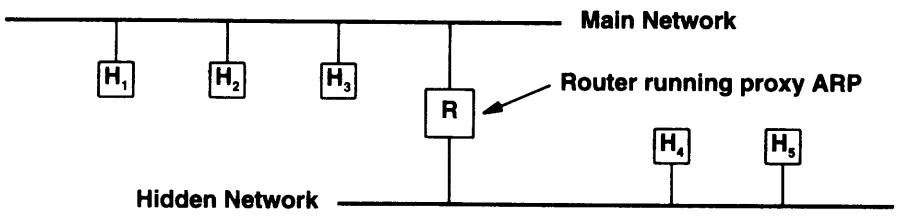


Figure 10.2 Proxy ARP technique (the ARP hack) allows one network address to be shared between two physical nets. Router *R* answers ARP requests on each network for hosts on the other network, giving its hardware address and then routing datagrams correctly when they arrive. In essence, *R* lies about IP-to-physical address bindings.

In the figure, two networks share a single IP network address. Imagine that the network labeled *Main Network* was the original network, and that the second, labeled *Hidden Network*, was added later. The router connecting the two networks, *R*, knows which hosts lie on which physical network and uses ARP to maintain the illusion that only one network exists. To make the illusion work, *R* keeps the location of hosts completely hidden, allowing all other machines on the network to communicate as if directly connected. In our example, when host H_1 needs to communicate with host H_4, it first invokes ARP to map H_4's IP address into a physical address. Once it has a physical address, H_1 can send the datagram directly to that physical address.

Because *R* runs proxy ARP software, it captures the broadcast ARP request from H_1, decides that the machine in question lies on the other physical network, and responds to the ARP request by sending its own physical address. H_1 receives the ARP response, installs the mapping in its ARP table, and then uses the mapping to send datagrams destined for H_4 to *R*. When *R* receives a datagram, it searches a special routing table to determine how to route the datagram. *R* must forward datagrams destined for H_4 over the hidden network. To allow hosts on the hidden network to reach hosts on the main network, *R* performs the proxy ARP service on that network as well.

Routers using the proxy ARP technique are taking advantage of an important feature of the ARP protocol, namely, trust. ARP is based on the idea that all machines cooperate and that any response is legitimate. Most hosts install mappings obtained through ARP without checking their validity and without maintaining consistency. Thus, it may happen that the ARP table maps several IP addresses to the same physical address, but that does not violate the protocol specification.

Some implementations of ARP are not as lax as others. In particular, ARP implementations designed to alert managers to possible security violations will inform them whenever two distinct IP addresses map to the same physical hardware address. The purpose of alerting the manager is to warn about *spoofing*, a situation in which one machine claims to be another in order to intercept packets. Host implementations of ARP that warn managers of possible spoofing cannot be used on networks that have proxy ARP routers because the software will generate messages frequently.

The chief advantage of proxy ARP is that it can be added to a single router on a network without disturbing the routing tables in other hosts or routers on that network. Thus, proxy ARP completely hides the details of physical connections.

The chief disadvantage of proxy ARP is that it does not work for networks unless they use ARP for address resolution. Furthermore, it does not generalize to more complex network topology (e.g., multiple routers interconnecting two physical networks), nor does it support a reasonable form of routing. In fact, most implementations of proxy ARP rely on managers to maintain tables of machines and addresses manually, making it both time consuming and prone to errors.

10.6 Subnet Addressing

The third technique used to allow a single network address to span multiple physical networks is called *subnet addressing*, *subnet routing*, or *subnetting*. Subnetting is the most widely used of the three techniques because it is the most general and because it has been standardized. In fact, subnetting is a required part of IP addressing.

The easiest way to understand subnet addressing is to imagine that a site has a single class *B* IP network address assigned to it, but it has two or more physical networks. Only local routers know that there are multiple physical nets and how to route traffic among them; routers in other autonomous systems route all traffic as if there were a single physical network. Figure 10.3 shows an example.

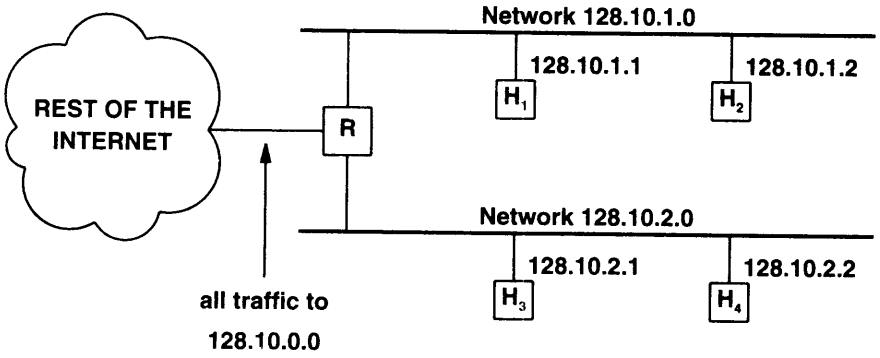


Figure 10.3 A site with two physical networks using subnet addressing to label them with a single class *B* network address. Router *R* accepts all traffic for net 128.10.0.0 and chooses a physical network based on the third octet of the address.

In the example, the site is using the single class *B* network address *128.10.0.0* for two networks. Except for router *R*, all routers in the internet route as if there were a single physical net. Once a packet reaches *R*, it must be sent across the correct physical network to its destination. To make the choice of physical network efficient, the local site has chosen to use the third octet of the address to distinguish between the two networks. The manager assigns machines on one physical net addresses of the form *128.10.1.X*, and machines on the other physical net addresses of the form *128.10.2.X*, where *X*, the final octet of the address, contains a small integer used to identify a specific host. To choose a physical network, *R* examines the third octet of the destination address and routes datagrams with value *1* to the network labeled *128.10.1.0* and those with value *2* to the network labeled *128.10.2.0*.

Conceptually, adding subnets only changes the interpretation of IP addresses slightly. Instead of dividing the 32-bit IP address into a network prefix and a host suffix, subnetting divides the address into a *network portion* and a *local portion*. The interpre-

tation of the network portion remains the same as for networks that do not use subnetting. As before, reachability to the network must be advertised to outside autonomous systems; all traffic destined for the network will follow the advertised route. The interpretation of the local portion of an address is left up to the site (within the constraints of the formal standard for subnet addressing). To summarize:

> *We think of a 32-bit IP address as having an internet portion and a local portion, where the internet portion identifies a site, possibly with multiple physical networks, and the local portion identifies a physical network and host at that site.*

The example of Figure 10.3 showed subnet addressing with a class *B* address that had a 2-octet internet portion and a 2-octet local portion. To make routing among the physical networks efficient, the site administrator in our example chose to use one octet of the local portion to identify a physical network, and the other octet of the local portion to identify a host on that network, as Figure 10.4 shows.

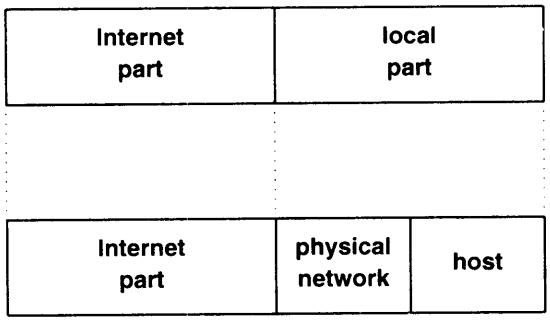


Figure 10.4 (a) Conceptual interpretation of a 32-bit IP address in the original IP address scheme, and (b) conceptual interpretation of addresses using the subnet scheme shown in Figure 10.3. The local portion is divided into two parts that identify a physical network and a host on that network.

The result is a form of *hierarchical addressing* that leads to corresponding *hierarchical routing*. The top level of the routing hierarchy (i.e., other autonomous systems in the internet) uses the first two octets when routing, and the next level (i.e., the local site) uses an additional octet. Finally, the lowest level (i.e., delivery across one physical network) uses the entire address.

Hierarchical addressing is not new; many systems have used it before. The best example is the U.S. telephone system, where a 10-digit phone number is divided into a 3-digit area code, 3-digit exchange, and 4-digit connection. The advantage of using

hierarchical addressing is that it accommodates large growth because it means a given router does not need to know as much detail about distant destinations as it does about local ones. One disadvantage is that choosing a hierarchical structure is difficult, and it often becomes difficult to change a hierarchy once it has been established.

10.7 Flexibility In Subnet Address Assignment

The TCP/IP standard for subnet addressing recognizes that not every site will have the same needs for an address hierarchy; it allows sites flexibility in choosing how to assign them. To understand why such flexibility is desirable, imagine a site with five networks interconnected, as Figure 10.5 shows. Suppose the site has a single class *B* network address that it wants to use for all physical networks. How should the local part be divided to make routing efficient?

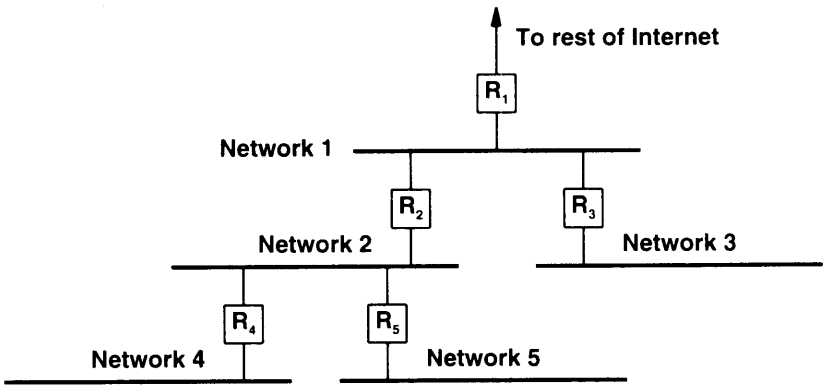


Figure 10.5 A site with five physical networks arranged in three "levels." The simplistic division of addresses into physical net and host parts may not be optimal for such cases.

In our example, the site will choose a partition of the local part of the IP address based on how it expects to grow. Dividing the 16-bit local part into an 8-bit network identifier and an 8-bit host identifier as shown in Figure 10.4 allows up to 256 networks, with up to 256 hosts per network†. Figure 10.6 illustrates the possible choices if a site uses the *fixed-length subnetting* scheme described above and avoids the all 0s and all 1s subnet and host addresses.

†In practice, the limit is 254 subnets of 254 hosts per subnet because the all 1s and all 0s host addresses are reserved for broadcast, and the all 1s or all 0s subnet is not recommended.

Subnet Bits	Number of Subnets	Hosts per Subnet
0	1	65534
2	2	16382
3	6	8190
4	14	4094
5	30	2046
6	62	1022
7	126	510
8	254	254
9	510	126
10	1022	62
11	2046	30
12	4094	14
13	8190	6
14	16382	2

Figure 10.6 The possible fixed-length subnets sizes for a class B number, with 8 subnet bits being the most popular choice; an organization must choose one line in the table.

As the figure shows, an organization that adopts fixed-length subnetting must choose a compromise. If the organization has a large number of physical networks, the networks cannot contain many hosts; if the number of hosts on a network is large, the number of physical networks must be small. For example, allocating 3 bits to identify a physical network results in up to 6 networks that each support up to 8190 hosts. Allocating 12 bits results in up to 4094 networks, but restricts the size of each to 62 hosts.

10.8 Variable-Length Subnets

We have implied that choosing a subnet addressing scheme is synonymous with choosing how to partition the local portion of an IP address into physical net and host parts. Indeed, most sites that implement subnetting use a fixed-length assignment. It should be clear that the designers did not choose a specific division for subnetting because no single partition of the local part of the address works for all organizations — some need many networks with few hosts per network, while others need a few networks with many hosts attached to each. The designers realized that the same problem can exist within a single organization. To allow maximum autonomy, the TCP/IP subnet standard provides even more flexibility than indicated above. An organization may select a subnet partition on a per-network basis. Although the technique is known as *variable-length subnetting*, the name is slightly misleading because the value does not "vary" over time — once a partition has been selected for a particular network, the partition never changes. All hosts and routers attached to that network must follow the decision; if they do not, datagrams can be lost or misrouted. We can summarize:

> *To allow maximum flexibility in choosing how to partition subnet addresses, the TCP/IP subnet standard permits variable-length subnetting in which the partition can be chosen independently for each physical network. Once a subnet partition has been selected, all machines on that network must honor it.*

The chief advantage of variable-length subnetting is flexibility: an organization can have a mixture of large and small networks, and can achieve higher utilization of the address space. However, variable-length subnetting has serious disadvantages. Most important, values for subnets must be assigned carefully to avoid *address ambiguity*, a situation in which an address is interpreted differently depending on the physical network. For example, an address can appear to match two different subnets. As a result, invalid variable-length subnets may make it impossible for all pairs of hosts to communicate. Routers cannot resolve such ambiguity, which means that an invalid assignment can only be repaired by renumbering. Thus, network managers are discouraged from using variable-length subnetting.

10.9 Implementation Of Subnets With Masks

The subnet technology makes configuration of either fixed or variable length easy. The standard specifies that a 32-bit mask is used to specify the division. Thus, a site using subnet addressing must choose a 32-bit *subnet mask* for each network. Bits in the subnet mask are set to *1* if machines on the network treat the corresponding bit in the IP address as part of the subnet prefix, and *0* if they treat the bit as part of the host identifier. For example, the 32-bit subnet mask:

11111111 11111111 11111111 00000000

specifies that the first three octets identify the network and the fourth octet identifies a host on that network. A subnet mask should have *1*s for all bits that correspond to the network portion of the address (e.g., the subnet mask for a class *B* network will have *1*s for the first two octets plus one or more bits in the last two octets).

The interesting twist in subnet addressing arises because the standard does not restrict subnet masks to select contiguous bits of the address. For example, a network might be assigned the mask:

11111111 11111111 00011000 01000000

which selects the first two octets, two bits from the third octet, and one bit from the fourth. Although such flexibility makes it possible to arrange interesting assignments of addresses to machines, doing so makes assigning host addresses and understanding routing tables tricky. Thus, it is recommended that sites use contiguous subnet masks and

that they use the same mask throughout an entire set of physical nets that share an IP address.

10.10 Subnet Mask Representation

Specifying subnet masks in binary is both awkward and prone to errors. Therefore, most software allows alternative representations. Sometimes, the representation follows whatever conventions the local operating system uses for representation of binary quantities, (e.g., hexadecimal notation).

Most IP software uses dotted decimal representation for subnet masks; it works best when sites choose to align subnetting on octet boundaries. For example, many sites choose to subnet class *B* addresses by using the third octet to identify the physical net and the fourth octet to identify hosts as on the previous page. In such cases, the subnet mask has dotted decimal representation *255.255.255.0*, making it easy to write and understand.

The literature also contains examples of subnet addresses and subnet masks represented in braces as a 3-tuple:

{ <network number> , <subnet number> , <host number> }

In this representation, the value *-1* means "all ones." For example, if the subnet mask for a class *B* network is *255.255.255.0*, it can be written *{-1, -1, 0}*.

The chief disadvantage of the 3-tuple representation is that it does not accurately specify how many bits are used for each part of the address; the advantage is that it abstracts away from the details of bit fields and emphasizes the values of the three parts of the address. To see why address values are sometimes more important than bit fields, consider the 3-tuple:

{ 128.10 , -1, 0 }

which denotes an address with a network number *128.10*, all ones in the subnet field, and all zeroes in the host field. Expressing the same address value using other representations requires a 32-bit subnet mask as well as a 32-bit IP address, and forces readers to decode bit fields before they can deduce the values of individual fields. Furthermore, the 3-tuple representation is independent of the IP address class or the size of the subnet field. Thus, the 3-tuple can be used to represent sets of addresses or abstract ideas. For example, the 3-tuple:

{ <network number>, -1, -1 }

denotes "addresses with a valid network number, a subnet field containing all ones, and a host field containing all ones." We will see additional examples later in this chapter.

10.11 Routing In The Presence Of Subnets

The standard IP routing algorithm must be modified to work with subnet addresses. All hosts and routers attached to a network that uses subnet addressing must use the modified algorithm, which is called *subnet routing*. What may not be obvious is that unless restrictions are added to the use of subnetting, other hosts and routers at the site may also need to use subnet routing. To see how a problem arises without restrictions, consider the example set of networks shown in Figure 10.7.

In the figure, physical networks *2* and *3* have been (illegally) assigned subnet addresses of a single IP network address, *N*. Although host *H* does not directly attach to a network that has a subnet address, it must use subnet routing to decide whether to send datagrams destined for network *N* to router R_1 or router R_2. It could be argued that *H* can send to either router and let them handle the problem, but that solution means not all traffic will follow a shortest path. In larger examples, the difference between an optimal and nonoptimal path can be significant.

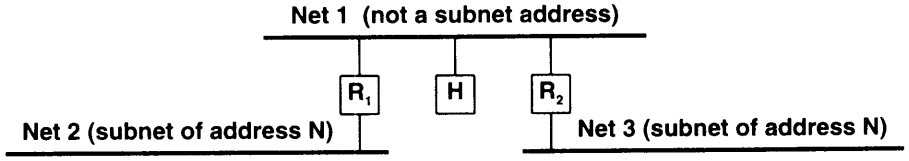


Figure 10.7 An example (illegal) topology with three networks where Nets *2* and *3* are subnets of a single IP network address, *N*. If such topologies were allowed, host *H* would need to use subnet routing even though Net *1* does not have a subnet address.

In theory, a simple rule determines when machines need to use subnet routing. The subnet rule is:

> *To achieve optimal routing, a machine* M *must use subnet routing for an IP network address* N, *unless there is a single path* P *such that P is a shortest path between* M *and every physical network that is a subnet of* N.

Unfortunately, understanding the theoretical restriction does not help in assigning subnets. First, shortest paths can change if hardware fails or if routing algorithms redirect traffic around congestion. Such dynamic changes make it difficult to use the subnet rule except in trivial cases. Second, the subnet rule fails to consider the boundaries of sites or the difficulties involved in propagating subnet masks. It is impossible to propagate subnet routes beyond the boundary of a given organization because the routing protocols discussed later do not provide for it. Realistically, it becomes extremely difficult to propagate subnet information beyond a given physical network. Therefore, the designers recommend that if a site uses subnet addressing, that site should keep subnet-

ting as simple as possible. In particular, network administrators should adhere to the following guidelines:

> *All subnets of a given network IP address must be contiguous, the subnet masks should be uniform across all networks, and all machines should participate in subnet routing.*

The guidelines pose special difficulty for a large corporation that has multiple sites each connected to the Internet, but not connected directly to one another. Such a corporation cannot use subnets of a single address for all its sites because the physical networks are not contiguous.

10.12 The Subnet Routing Algorithm

Like the standard IP routing algorithm, the algorithm used with subnets searches a table of routes. Recall that in the standard algorithm, per-host routes and default routes are special cases that must be checked explicitly; table lookup is used for all others. A conventional routing table contains entries of the form:

(network address, next hop address)

where the *network address* field specifies the IP address of a destination network, *N*, and the *next hop address* field specifies the address of a router to which datagrams destined for *N* should be sent. The standard routing algorithm compares the network portion of a destination address to the *network address* field of each entry in the routing table until a match is found. Because the *next hop address* field is constrained to specify a machine that is reachable over a directly connected network, only one table lookup is ever needed.

The standard algorithm knows how an address is partitioned into network portion and local portion because the first three bits encode the address type and format (i.e., class *A*, *B*, *C*, or *D*). With subnets, it is not possible to decide which bits correspond to the network and which to the host from the address alone. Instead, the modified algorithm used with subnets maintains additional information in the routing table. Each table entry contains one additional field that specifies the subnet mask used with the network in that entry:

(subnet mask, network address, next hop address)

When choosing routes, the modified algorithm uses the *subnet mask* to extract bits of the destination address for comparison with the table entry. That is, it performs a bitwise Boolean *and* of the full 32-bit destination IP address and the *subnet mask* field from an entry, and it then checks to see if the result equals the value in the *network address* field of that entry. If so, it routes the datagram to the address specified in the *next hop address* field† of the entry.

---

†As in the standard routing algorithm, the next hop router must be reachable by a directly connected network.

10.13 A Unified Routing Algorithm

Observant readers may have guessed that if we allow arbitrary masks, the subnet routing algorithm can subsume all the special cases of the standard algorithm. It can handle routes to individual hosts, a default route, and routes to directly connected networks using the same masking technique it uses for subnets. In addition, masks can handle routes to conventional classful addresses. The flexibility comes from the ability to combine arbitrary 32-bit values in a *subnet mask* field and arbitrary 32-bit addresses in a *network address* field. For example, to install a route for a single host, one uses a mask of all *1*s and a network address equal to the host's IP address. To install a default route, one uses a subnet mask of all *0*s and a network address of all *0*s (because any destination address *and* zero equals zero). To install a route to a (nonsubnetted) class *B* network, one specifies a mask with two octets of *1*s and two octets of *0*s. Because the table contains more information, the routing algorithm contains fewer special cases as Figure 10.8 shows.

Algorithm:

Route_IP_Datagram (datagram, routing_table)

```
Extract destination IP address, ID, from datagram;
If prefix of ID matches address of any directly connected
   network send datagram to destination over that network
   (This involves resolving ID to a physical address,
   encapsulating the datagram, and sending the frame.)
else
   for each entry in routing table do
      Let N be the bitwise-and of ID and the subnet mask
      If N equals the network address field of the entry then
        route the datagram to the specified next hop address
   endforloop
If no matches were found, declare a routing error;
```

Figure 10.8 The unified IP routing algorithm. Given an IP datagram and a routing table with masks, this algorithm selects a next hop router to which the datagram should be sent. The next hop must lie on a directly connected network.

In fact, most implementations eliminate the explicit test for destinations on directly connected networks. To do so, one must add a table entry for each directly connected network. Like other entries, each entry for a directly connected network contains a mask that specifies the number of bits in the prefix.

10.14 Maintenance Of Subnet Masks

How do subnet masks get assigned and propagated? Chapter 9 answered the second part of the question by showing that a host can obtain the subnet mask for a given network by sending an ICMP *subnet mask request* to a router on that network. The request can be broadcast if the host does not know the specific address of a router. Later chapters will complete the answer to the second part by explaining that some of the protocols routers use to exchange routing information pass subnet masks along with each network address.

The first part of the question is more difficult to answer. Each site is free to choose subnet masks for its networks. When making assignments, managers attempt to balance sizes of networks, numbers of physical networks, expected growth, and ease of maintenance. Difficulty arises because nonuniform masks give the most flexibility, but make possible assignments that lead to ambiguous routes. Or worse, they allow valid assignments that become invalid if more hosts are added to the networks. There are no easy rules, so most sites make conservative choices. Typically, a site selects contiguous bits from the local portion of an address to identify a network, and uses the same partition (i.e., the same mask) for all local physical networks at the site. For example, many sites simply use a single subnet octet when subnetting a class *B* address.

10.15 Broadcasting To Subnets

Broadcasting is more difficult in a subnet architecture. Recall that in the original IP addressing scheme, an address with a host portion of all *1*s denotes broadcast to all hosts on the specified network. From the viewpoint of an observer outside a subnetted site, broadcasting to the network address still makes sense†. That is, the address:

{ network, -1, -1 }

means "deliver a copy to all machines that have *network* as their network addresses, even if they lie on separate physical networks." Operationally, broadcasting to such an address makes sense only if the routers that interconnect the subnets agree to propagate the datagram to all physical networks. Of course, care must be taken to avoid routing loops. In particular, a router cannot merely propagate a broadcast packet that arrives on one interface to all interfaces that share the subnet prefix. To prevent such loops, routers use *reverse path forwarding*. The router extracts the source of the broadcast datagram, and looks up the source in its routing table. The router then discards the da-

†Classless addressing, covered later in this chapter, has made broadcasting to all subnets obsolete.

tagram unless it arrived on the interface used to route to the source (i.e., arrived from the shortest path).

Within a set of subnetted networks, it becomes possible to broadcast to a specific subnet (i.e., to broadcast to all hosts on a physical network that has been assigned one of the subnet addresses). The subnet address standard uses a host field of all ones to denote subnet broadcast. That is, a subnet broadcast address becomes:

{ network, subnet, -1 }

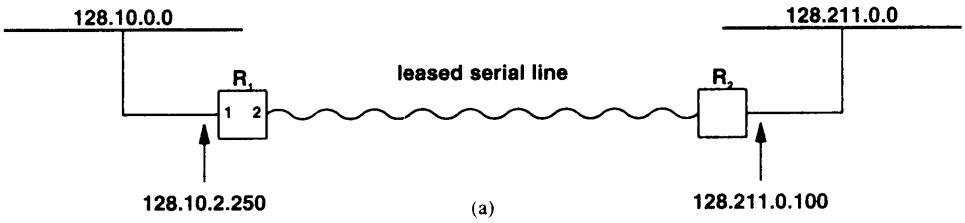
Considering subnet broadcast addresses and subnet broadcasting clarifies the recommendation for using a consistent subnet mask across all networks that share a subnetted IP address. As long as the subnet and host fields are identical, subnet broadcast addresses are unambiguous. More complex subnet address assignments may or may not allow broadcasting to selected subsets of the physical networks that comprise a subnet.

10.16 Anonymous Point-To-Point Networks

In the original IP addressing scheme, each network was assigned a unique prefix. In particular, because IP views each point-to-point connection between a pair of machines as a "network," the connection was assigned a network prefix and each computer was assigned a host suffix. When addresses became scarce, the use of a prefix for each point-to-point connection seemed absurd. The problem is especially severe for organizations that have many point-to-point connections. For example, an organization with multiple sites might use leased digital circuits (e.g., T1 lines) to form a backbone that interconnects a router at each site to routers at other sites.

To avoid assigning a prefix to each point-to-point connection, a simple technique was invented. Known as *anonymous networking*, the technique is often applied when a pair of routers is connected with a leased digital circuit. The technique simply avoids numbering the leased line, and does not assign a host address to the routers at each end. No hardware address is needed, so the interface software is configured to ignore the next hop address when sending datagrams. Consequently, an arbitrary value can be used as the next-hop address in the IP routing table.

When the anonymous networking technique is applied to a point-to-point connection, the connection is known as an *unnumbered network* or an *anonymous network*. The example in Figure 10.9 will help explain routing in unnumbered networks.



(a)

TO REACH HOSTS ON NETWORK	ROUTE TO THIS ADDRESS	USING THIS INTERFACE
128.10.0.0	DELIVER DIRECT	1
default	128.211.0.100	2

(b)

Figure 10.9 (a) An unnumbered point-to-point connection between two routers, and (b) the routing table in router R_1.

To understand why unnumbered networks are possible, one must remember that serial lines used for point-to-point connections do not operate like shared-media hardware. Because there is only one possible destination — the computer at the other end of the circuit — the underlying hardware does not use physical addresses when transmitting frames. Consequently, when IP hands a datagram to the network interface, any value can be specified as a next hop because the hardware will ignore it. Thus, the next-hop field of the IP routing table can contain an arbitrary value (e.g., zero).

The routing table in Figure 10.9b does not have a zero in the next hop field. Instead, the example demonstrates a technique often employed with unnumbered networks. Rather than leaving the next hop empty, it is filled with one of the IP addresses assigned to the next-hop router (i.e., an address assigned to another of the router's interfaces). In the example, the address of R_2's Ethernet connection has been used.

We said that the hardware ignores the next hop address, so it may seem odd that a value has been assigned. It may seem even more odd that the next-hop refers to a network not directly reachable from R_1. In fact, neither IP nor the network interface code uses the value in any way. The only reason for specifying a non-zero entry is to make it easier for humans to understand and remember the address of the router on the other end of the point-to-point connection. In the example, we chose the address assigned to R_2's Ethernet interface because R_2 does not have an address for the leased line interface.

10.17 Classless Addressing (Supernetting)

Subnet addressing was invented in the early 1980s to help conserve the IP address space; the unnumbered networking technique followed. By 1993, it became apparent that those techniques alone would not prevent Internet growth from eventually exhausting the address space. Work had begun on defining an entirely new version of IP with larger addresses. To accommodate growth until the new version of IP could be standardized and adopted, however, a temporary solution was found.

Called *classless addressing*, *supernet addressing*, or *supernetting*, the scheme takes an approach that is complementary to subnet addressing. Instead of using a single IP network prefix for multiple physical networks at a given organization, supernetting allows the addresses assigned to a single organization to span multiple classed prefixes.

To understand why classless addressing was adopted, one needs to know three facts. First, the classful scheme did not divide network addresses into classes equally. Although less than seventeen thousand class B numbers can be assigned, more than two million class C network numbers exist. Second, class C numbers were being requested slowly; only a small percentage of them had been assigned. Third, studies showed that at the rate class B numbers were being assigned, class B prefixes would be exhausted quickly. The situation became known as the *Running Out of ADdress Space* (*ROADS*) problem.

To understand how supernetting works, consider a medium-sized organization that joins the Internet. Such an organization would prefer to use a single class B address for two reasons: a class C address cannot accommodate more than 254 hosts and a class B address has sufficient bits to make subnetting convenient. To conserve class B numbers, the supernetting scheme assigns an organization a block of class C addresses instead of a single class B number. The block must be large enough to number all the networks the organization will eventually connect to the Internet. For example, suppose an organization requests a class B address and intends to subnet using the third octet as a subnet field. Instead of a single class B number, supernetting assigns the organization a block of 256 contiguous class C numbers that the organization can then assign to physical networks.

Although supernetting is easy to understand when viewed as a way to satisfy a single organization, the proposers intended it to be used in a broader context. They envisioned a hierarchical Internet in which commercial *Internet Service Providers* (*ISPs*) provide Internet connectivity. To connect its networks to the Internet, an organization contracts with an ISP; the service provider handles the details of assigning IP addresses to the organization as well as installing physical connections. The designers of supernetting propose that an Internet Service Provider be assigned a large part of the address space (i.e., a set of addresses that span many class C network numbers). The ISP can then allocate one or more addresses from the set to each of its subscribers.

10.18 The Effect Of Supernetting On Routing

Allocating many class C addresses in place of a single class B address conserves class B numbers and solves the immediate problem of address space exhaustion. However, it creates a new problem: the information that routers store and exchange increases dramatically. For example, assigning an organization 256 class C addresses instead of a class B address requires 256 routes instead of one.

A technique known as *Classless Inter-Domain Routing*† (*CIDR*) solves the problem. Conceptually, CIDR collapses a block of contiguous class C addresses into a single entry represented by a pair:

(network address, count)

where *network address* is the smallest address in the block, and *count* specifies the total number of network addresses in the block. For example, the pair:

(192.5.48.0, 3)

is used to specify the three network addresses 192.5.48.0, 192.5.49.0, and 192.5.50.0.

If a few Internet Service Providers form the core of the Internet and each ISP owns a large block of contiguous IP network numbers, the benefit of supernetting becomes clear: routing tables are much smaller. Consider routing table entries in routers owned by service provider *P*. The table must have a correct route to each of *P*'s subscribers, but the table does not need to contain a route for other providers' subscribers. Instead, the table stores one entry for each other provider, where the entry identifies the block of addresses owned by the provider.

10.19 CIDR Address Blocks And Bit Masks

In practice, CIDR does not restrict network numbers to class C addresses nor does it use an integer count to specify a block size. Instead, CIDR requires the size of each block of addresses to be a power of two, and uses a bit mask to identify the size of the block. For example, suppose an organization is assigned a block of 2048 contiguous addresses starting at address 128.211.168.0. The table in Figure 10.10 shows the binary values of addresses in the range.

CIDR requires two items to specify the block of addresses in Figure 10.10: the 32-bit value of the lowest address in the block and a 32-bit mask. The mask operates like a standard subnet mask by delineating the end of the prefix‡. For the range shown, a CIDR mask has 21 bits set, which means that the division between prefix and suffix occurs after the 21st bit:

11111111 11111111 11111000 00000000

†The name is a slight misnomer because the scheme specifies addressing as well as routing.

‡Unlike a subnet mask, a CIDR mask must use contiguous bits.

	Dotted Decimal	32-bit Binary Equivalent
lowest	128.211.168.0	10000000 11010011 10101000 00000000
highest	128.211.175.255	10000000 11010011 10101111 11111111

Figure 10.10 An example CIDR block of 2048 addresses. The table shows the lowest and highest addresses in the range expressed as dotted decimal and binary values.

10.20 Address Blocks And CIDR Notation

Because identifying a CIDR block requires both an address and a mask, a shorthand notation was devised to express the two items. Called *CIDR notation* but known informally as *slash notation*, the shorthand represents the mask length in decimal and uses a slash to separate it from the address. Thus, in CIDR notation, the block of addresses in Figure 10.10 would be expressed as:

128.211.168.0/21

where */21* denotes 21 bits in a mask. The table in Figure 10.11 lists dotted decimal values for all possible CIDR masks. The /8, /16, and /24 prefixes correspond to traditional class *A*, *B*, and *C* divisions.

CIDR Notation	Dotted Decimal	CIDR Notation	Dotted Decimal
/1	128.0.0.0	/17	255.255.128.0
/2	192.0.0.0	/18	255.255.192.0
/3	224.0.0.0	/19	255.255.224.0
/4	240.0.0.0	/20	255.255.240.0
/5	248.0.0.0	/21	255.255.248.0
/6	252.0.0.0	/22	255.255.252.0
/7	254.0.0.0	/23	255.255.254.0
/8	255.0.0.0	/24	255.255.255.0
/9	255.128.0.0	/25	255.255.255.128
/10	255.192.0.0	/26	255.255.255.192
/11	255.224.0.0	/27	255.255.255.224
/12	255.240.0.0	/28	255.255.255.240
/13	255.248.0.0	/29	255.255.255.248
/14	255.252.0.0	/30	255.255.255.252
/15	255.254.0.0	/31	255.255.255.254
/16	255.255.0.0	/32	255.255.255.255

Figure 10.11 Dotted decimal mask values for all possible CIDR prefixes.

10.21 A Classless Addressing Example

The table in Figure 10.11 illustrates one of the chief advantages of classless addressing: complete flexibility in allocating blocks of various sizes. With CIDR, the ISP can choose to assign each customer a block of an appropriate size. If it owns a CIDR block of N bits, an ISP can choose to hand customers any piece of more than N bits. For example, if the ISP is assigned 128.211.0.0/16, the ISP may choose to give one of its customers the 2048 address in the /21 range that Figure 10.10 specifies. If the same ISP also has a small customer with only two computers, the ISP might choose to assign another block 128.211.176.212/29, which covers the address range that Figure 10.12 specifies.

	Dotted Decimal	32-bit Binary Equivalent
lowest	128.211.176.212	10000000 11010011 10110000 11010100
highest	128.211.176.215	10000000 11010011 10110000 11010111

Figure 10.12 An example of CIDR block 128.211.176.212/29. The use of an arbitrary bit mask allows more flexibility in assigning a block size than the classful addressing scheme.

One way to think about classless addresses is as if each customer of an ISP obtains a (variable-length) subnet of the ISP's CIDR block. Thus, a given block of addresses can be subdivided on an arbitrary bit boundary, and a separate route can be entered for each subdivision. As a result, although the group of computers on a given network will be assigned addresses in a contiguous range, the range does not need to correspond to a predefined class. Instead, the scheme makes subdivision flexible by allowing one to specify the exact number of bits that correspond to a prefix. To summarize:

> *Classless addressing, which is now used by ISPs, treats IP addresses as arbitrary integers, and allows a network administrator to assign addresses in contiguous blocks, where the number of addresses in a block is a power of two.*

10.22 Data Structures And Algorithms For Classless Lookup

The fundamental criterion used to judge the algorithms and data structures used with routing tables is speed. There are two aspects: the primary consideration is the speed of finding a next hop for a given destination, while a secondary consideration is the speed of making changes to values in the table.

The introduction of classless addressing had a profound effect on routing because it changed a fundamental assumption: unlike a classful address, a CIDR address is not *self-identifying*. That is, a router cannot determine the division between prefix and suf-

fix merely by looking at the address. The difference is important because it means that data structures and search algorithms used with classful addresses do not work when routing tables contain classless addresses. After a brief review of classful lookup, we will consider one of the data structures used for classless lookup.

10.22.1 Hashing And Classful Addresses

All route lookup algorithms are optimized for speed. When IP permitted only classful addresses, a single technique provided the necessary optimization: hashing. When a classful address is entered in a routing table, the router extracts the network portion, *N*, and uses it as a hash key. Similarly, given a destination address, the router also extracts the network portion, *N*, computes a hash function *h(N)*, and uses the result as an index into a bucket.

Hashing works well in a classful situation because addresses are self-identifying. Even if some entries in a table correspond to subnet routes, hashing is still efficient because the network portion of the address can be extracted and used as a key. If multiple routes hash to the same bucket in the table, entries within the bucket are arranged in decreasing order of specificity – subnet routes precede network routes. Thus, if a given destination matches both a network route and a subnet route, the algorithm will correctly find and use the subnet route.

In a classless world, however, where addresses are not self-identifying, hashing does not work well. Because it cannot compute the division between prefix and suffix, a router cannot find a hash key for an arbitrary address. Thus, an alternate scheme must be found.

10.22.2 Searching By Mask Length

The simplest lookup algorithm that accommodates classless addressing merely iterates over all possible divisions between prefix and suffix. That is, given a destination address, *D*, the algorithm first tries using 32 bits of *D*, then 31 bits, and so on down to 0 bits. For each possible size, *M*, the router extracts *M* bits from *D*, assumes the extracted bits comprise a network prefix, and looks up the prefix in the table. The algorithm chooses the longest prefix that corresponds to a route in the table (i.e., the search stops as soon as a match has been found).

The disadvantage of trying all possible lengths should be obvious: doing so is many times slower than a standard classful lookup because the algorithm must search the table for each possible prefix size until a match is found. The worst case occurs when no route exists; in which case, the algorithm searches the table 32 times. Even when it finds a route, a router using the iterative approach searches the table many times unnecessarily. For example, 16 lookups are required before a router can find a traditional class *B* network (i.e., /16) route. More important, the algorithm performs 31 unnecessary lookups before it succeeds in matching the default route (in many routing tables, the default route is heavily used).

10.22.3 Binary Trie Structures

To avoid inefficient searches, production software for classless routing lookup must avoid the iterative approach. Instead, classless routing tables are usually stored in a hierarchical data structure, and searching proceeds down the hierarchy. The most popular data structures are variants of a *binary trie* in which the value of successive bits in the address determine a path from the root downward.

A binary trie is a tree with paths determined by the data stored. To visualize a binary trie, imagine that a set of 32-bit addresses is written as binary strings and redundant suffixes are removed. What remains is a set of prefixes that uniquely identify each item. For example, Figure 10.13 shows a set of seven addresses written in binary and the corresponding unique prefixes.

As Figure 10.13 illustrates, the number of bits required to identify an address depends on the values in the set. For example, the first address in the figure can be uniquely identified by three bits because no other addresses begin with *001*. However, five bits are required to identify the last item in the table because the 4-bit prefix *1011* is shared by more than one item.

32-Bit Address	Unique Prefix
00110101 00000000 00000000 00000000	001
01000110 00000000 00000000 00000000	0100
01010110 00000000 00000000 00000000	0101
01100001 00000000 00000000 00000000	011
10101010 11110000 00000000 00000000	1010
10110000 00000010 00000000 00000000	10110
10111011 00001010 00000000 00000000	10111

Figure 10.13 A set of 32-bit binary addresses and the corresponding set of prefixes that uniquely identify each.

Once a set of unique prefixes has been computed, they can be used to define a binary trie. Figure 10.14 illustrates a trie for the seven prefixes in Figure 10.13.

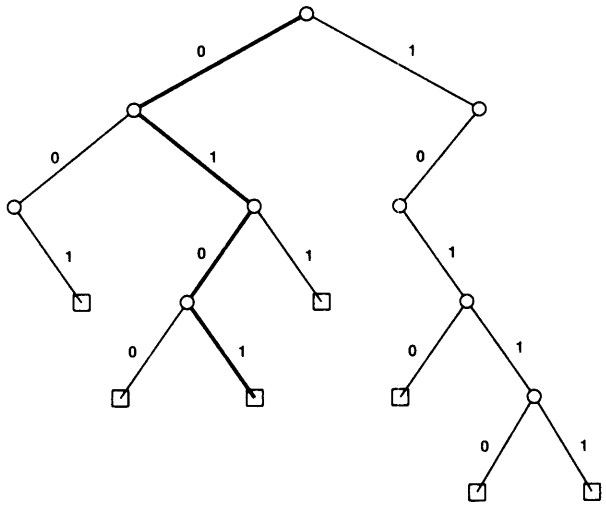


Figure 10.14 A binary trie for the seven binary prefixes listed in Figure 10.13. The path through the trie for prefix *0101* is shown darkened.

Each interior node in the trie (shown as a circle) corresponds to two or more prefixes, and each exterior node (shown as a square) corresponds to one unique prefix. The search algorithm stops when it reaches an exterior node or when no path exists for the specified prefix. For example, a search for address

10010010 11110000 00000000 00000001

fails because there is no branch with label *0* at the node corresponding to *10*.

To make routing lookup efficient, routing software that handles classless routes must use data structures and algorithms that differ from those used for classful lookup. Many systems use a scheme based on a binary trie to accommodate classless lookup.

10.23 Longest-Match Routing And Mixtures Of Route Types

Our brief description of binary tries only gives a sketch of the data structure used in practice. For example, we said that a trie only needs to store a unique prefix for each route in the table, without stating that the prefix must cover the entire network portion of the route. To guarantee that a router does not forward datagrams unless the entire network prefix in the destination matches the route, each exterior node in the trie must

contain a 32-bit address, A, and a 32-bit mask, M, that covers the entire network portion of A. When the search reaches an exterior node, the algorithm computes the logical *and* of M with the destination address, and compares the result to A in the same way that conventional lookup algorithms do. If the comparison fails, the datagram is rejected (also like conventional lookup algorithms). In other words, we can view the trie as a mechanism that quickly identifies items in the routing table that are potential candidates rather than a mechanism that finds an exact match.

Even if we consider the trie to be a mechanism that identifies potential matches, another important detail is missing from our description. We have assumed that each entry in a routing table has a unique binary prefix. In practice, however, the entries in most routing tables do not have unique prefixes because routing tables contain a mixture of general and specific routes for the same destination. For example, consider any routing table that contains a network-specific route and a different route for one particular subnet of the same network. Or consider a routing table that contains both a network-specific route and a special route for one host on that network. The binary prefix of the network route is also a prefix of the subnet or host-specific route. Figure 10.15 provides an example.

Prefix	Next Hop
128.10.0.0/16	10.0.0.2
128.10.2.0/24	10.0.0.4
128.10.3.0/24	10.1.0.5
128.10.4.0/24	10.0.0.6
128.10.4.3/32	10.0.0.3
128.10.5.0/24	10.0.0.6
128.10.5.1/32	10.0.0.3

Figure 10.15 An example set of routes without unique prefixes. The situation occurs frequently because many routing tables contain a mixture of general and specific routes for the same network.

To permit overlapping prefixes, the trie data structure described above must be modified to follow the *longest-match* paradigm when selecting a route. To do so, one must allow interior nodes to contain an address/mask pair, and modify the search algorithm to check for a match at each node. A match that occurs later in the search (i.e., a match that corresponds to a more specific route) must override any match that occurs earlier because a later match corresponds to a longer prefix.

10.23.1 PATRICIA And Level Compressed Tries

Our description of binary tries also omits details related to optimization of lookup. The most important involves "skipping" levels in the trie that do not distinguish among routes. For example, consider a binary trie for the set of routes in Figure 10.15. Because each route in the list begins with the same sixteen bits (i.e., the value

10000000 00001010), a binary trie for the routes will only have one node at each of the first sixteen levels below the root.

In this instance, it would be faster to examine all sixteen bits of a destination address at once rather than extracting bits one at a time and using them to move through the trie. Two modified versions of tries use the basic optimization. The first, a *PATRICIA tree*, allows each node to specify a value to test along with a number of bits to skip. The second, a *level compressed trie*, provides additional optimization by eliminating one or more levels in the trie that can be skipped along any path.

Of course, data structure optimizations represent a tradeoff. Although the optimizations improve search speed, they require more computation when creating or modifying a routing table. In most cases, however, such optimizations are justified because one expects a routing table to be modified much less frequently than it is searched.

10.24 CIDR Blocks Reserved For Private Networks

Chapter 4 stated that the IETF had designated a set of prefixes to be reserved for use with private networks. As a safeguard, reserved prefixes will never be assigned to networks in the global Internet. Collectively, the reserved prefixes are known as *private addresses* or *nonroutable addresses*. The latter term arises because routers in the global Internet understand that the addresses are reserved; if a datagram destined to one of the private addresses is accidentally routed onto the global Internet, a router in the Internet will be able to detect the problem.

In addition to blocks that correspond to classful addresses, the set of reserved IPv4 prefixes contains a CIDR block that spans multiple classes. Figure 10.16 lists the values in CIDR notation along with the dotted decimal value of the lowest and highest addresses in the block. The last address block listed, *169.254/16*, is unusual because it is used by systems that *autoconfigure* IP addresses.

Prefix	Lowest Address	Highest Address
10/8	10.0.0.0	10.255.255.255
172.16/12	172.16.0.0	172.31.255.255
192.168/16	192.168.0.0	192.168.255.255
169.254/16	169.254.0.0	169.254.255.255

Figure 10.16 The prefixes reserved for use with private internets not connected to the global Internet. If a datagram sent to one of these addresses accidentally reaches the Internet, an error will result.

10.25 Summary

The original IP address scheme assigns a unique prefix to each physical network. This chapter examined five techniques that have been invented to conserve IP addresses. The first technique uses transparent routers to extend the address space of a single network, usually a WAN, to include hosts on an attached local network. The second technique, called proxy ARP, arranges for a router to impersonate computers on another physical network by answering ARP requests on their behalf. Proxy ARP is useful only on networks that use ARP for address resolution, and only for ARP implementations that do not complain when multiple internet addresses map to the same hardware address. The third technique, a TCP/IP standard called subnet addressing, allows a site to share a single IP network address among multiple physical networks. All hosts and routers connected to networks using subnetting must use a modified routing scheme in which each routing table entry contains a subnet mask. The modified scheme can be viewed as a generalization of the original routing algorithm because it handles special cases like default routes or host-specific routes. The fourth technique allows a point-to-point link to remain unnumbered (i.e., have no IP prefix).

The fifth technique, known as classless addressing (CIDR), represents a major shift in IP technology. Instead of adhering to the original network classes, classless addressing allows the division between prefix and suffix to occur on an arbitrary bit boundary. CIDR allows the address space to be divided into blocks, where the size of each block is a power of two. One of the main motivations for CIDR arises from the desire to combine multiple class C prefixes into a single supernet block. Because classless addresses are not self-identifying like the original classful addresses, CIDR requires significant changes to the algorithms and data structures used by IP software on hosts and routers to store and look up routes. Many implementations use a scheme based on the binary trie data structure.

FOR FURTHER STUDY

The standard for subnet addressing comes from Mogul [RFC 950] with updates in Braden [RFC 1122]. Clark [RFC 932], Karels [RFC 936], Gads [RFC 940], and Mogul [RFC 917] all contain early proposals for subnet addressing schemes. Mogul [RFC 922] discusses broadcasting in the presence of subnets. Postel [RFC 925] considers the use of proxy ARP for subnets. Atallah and Comer [1998] presents a provably optimal algorithm for variable-length subnet assignment. Carl-Mitchell and Quarterman [RFC 1027] discusses using proxy ARP to implement transparent subnet routers. Rekhter and Li [RFC 1518] specifies classless IP address allocation. Fuller, Li, Yu, and Varadhan [RFC 1519] specifies CIDR routing and supernetting. Rekhter et. al. [RFC 1918] specifies address prefixes reserved for private networks. Knuth [1973] describes the PATRICIA data structure.

EXERCISES

10.1 If routers using proxy ARP use a table of host addresses to decide whether to answer ARP requests, the routing table must be changed whenever a new host is added to one of the networks. Explain how to assign IP addresses so hosts can be added without changing tables. Hint: think of subnets.

10.2 Although the standard allows all-0's to be assigned as a subnet number, some vendors' software does not operate correctly. Try to assign a zero subnet at your site and see if the route is propagated correctly.

10.3 Can transparent routers be used with local area networks like the Ethernet? Why or why not?

10.4 Show that proxy ARP can be used with three physical networks that are interconnected by two routers.

10.5 Consider a fixed subnet partition of a class *B* network number that will accommodate at least 76 networks. How many hosts can be on each network?

10.6 Does it ever make sense to subnet a class *C* network address? Why or why not?

10.7 A site that chose to subnet their class *B* address by using the third octet for the physical net was disappointed that they could not accommodate 255 or 256 networks. Explain.

10.8 Design a subnet address scheme for your organization assuming that you have one class *B* address to use.

10.9 Is it reasonable for a single router to use both proxy ARP and subnet addressing? If so, explain how. If not, explain why.

10.10 Argue that any network using proxy ARP is vulnerable to "spoofing" (i.e., an arbitrary machine can impersonate any other machine).

10.11 Can you devise a (nonstandard) implementation of ARP that supports normal use, but prohibits proxy ARP?

10.12 One vendor decided to add subnet addressing to its IP software by allocating a single subnet mask used for all IP network addresses. The vendor modified its standard IP routing software to make the subnet check a special case. Find a simple example in which this implementation cannot work correctly. (Hint: think of a multi-homed host.)

10.13 Characterize the (restricted) situations in which the subnet implementation discussed in the previous exercise will work correctly.

10.14 Read the standard to find out more about broadcasting in the presence of subnets. Can you characterize subnet address assignments that allow one to specify a broadcast address for all possible subnets?

10.15 The standard allows an arbitrary assignment of subnet masks for networks that comprise a subnetted IP address. Should the standard restrict subnet masks to cover contiguous bits in the address? Why or why not?

10.16 Find an example of variable length subnet assignments and host addresses that produces address ambiguity.

10.17 Carefully consider default routing in the presence of subnets. What can happen if a packet arrives destined for a nonexistent subnet?

10.18 Compare architectures that use subnet addressing and routers to interconnect multiple Ethernets to an architecture that uses bridges as described in Chapter 2. Under what circumstances is one architecture preferable to the other?

10.19 Consider a site that chooses to subnet a class *B* network address, but decides that some physical nets will use *6* bits of the local portion to identify the physical net while others will use *8*. Find an assignment of host addresses that makes destination addresses ambiguous.

10.20 The subnet routing algorithm in Figure 10.8 uses a sequential scan of entries in the routing table, allowing a manager to place host-specific routes before network-specific or subnet-specific routes. Invent a data structure that achieves the same flexibility but uses hashing to make the lookup efficient. [This exercise was suggested by Dave Mills.]

10.21 Although much effort has been expended on making routers operate quickly, software for classless route lookup still runs slower than the hashing schemes used with classful lookup. Investigate data structures and lookup algorithms that operate faster than a binary trie.

10.22 A binary trie uses one bit to select among two descendants at each node. Consider a trie that uses two bits to select among four descendants at each node. Under what conditions does such a trie make lookup faster? Slower?

10.23 If all Internet service providers use classless addressing and assign subscribers numbers from their block of addresses, what problem occurs when a subscriber changes from one provider to another?

11

Protocol Layering

11.1 Introduction

Previous chapters review the architectural foundations of internetworking, describe how hosts and routers forward Internet datagrams, and present mechanisms used to map IP addresses to physical network addresses. This chapter considers the structure of the software found in hosts and routers that carries out network communication. It presents the general principle of layering, shows how layering makes Internet Protocol software easier to understand and build, and traces the path of datagrams through the protocol software they encounter when traversing a TCP/IP internet.

11.2 The Need For Multiple Protocols

We have said that protocols allow one to specify or understand communication without knowing the details of a particular vendor's network hardware. They are to computer communication what programming languages are to computation. It should be apparent by now how closely the analogy fits. Like assembly language, some protocols describe communication across a physical network. For example, the details of the Ethernet frame format, network access policy, and frame error handling comprise a protocol that describes communication on an Ethernet. Similarly, like a high-level language, the Internet Protocol specifies higher-level abstractions (e.g., IP addressing, datagram format, and the concept of unreliable, connectionless delivery).

Complex data communication systems do not use a single protocol to handle all transmission tasks. Instead, they require a set of cooperative protocols, sometimes called a *protocol family* or *protocol suite*. To understand why, think of the problems that arise when machines communicate over a data network:

• *Hardware failure.* A host or router may fail either because the hardware fails or because the operating system crashes. A network transmission link may fail or accidentally be disconnected. The protocol software needs to detect such failures and recover from them if possible.

• *Network congestion.* Even when all hardware and software operates correctly, networks have finite capacity that can be exceeded. The protocol software needs to arrange ways that a congested machine can suppress further traffic.

• *Packet delay or loss.* Sometimes, packets experience extremely long delays or are lost. The protocol software needs to learn about failures or adapt to long delays.

• *Data corruption.* Electrical or magnetic interference or hardware failures can cause transmission errors that corrupt the contents of transmitted data. Protocol software needs to detect and recover from such errors.

• *Data duplication or inverted arrivals.* Networks that offer multiple routes may deliver data out of sequence or may deliver duplicates of packets. The protocol software needs to reorder packets and remove any duplicates.

Taken together, all the problems seem overwhelming. It is difficult to understand how to write a single protocol that will handle them all. From the analogy with programming languages, we can see how to conquer the complexity. Program translation has been partitioned into four conceptual subproblems identified with the software that handles each subproblem: compiler, assembler, link editor, and loader. The division makes it possible for the designer to concentrate on one subproblem at a time, and for the implementor to build and test each piece of software independently. We will see that protocol software is partitioned similarly.

Two final observations from our programming language analogy will help clarify the organization of protocols. First, it should be clear that pieces of translation software must agree on the exact format of data passed between them. For example, the data passed from a compiler to an assembler consists of a program defined by the assembly programming language. The translation process involves multiple representations. The analogy holds for communication software because multiple protocols define the representations of data passed among communication software modules. Second, the four parts of the translator form a linear sequence in which output from the compiler becomes input to the assembler, and so on. Protocol software also uses a linear sequence.

11.3 The Conceptual Layers Of Protocol Software

Think of the modules of protocol software on each machine as being stacked vertically into *layers*, as in Figure 11.1. Each layer takes responsibility for handling one part of the problem.

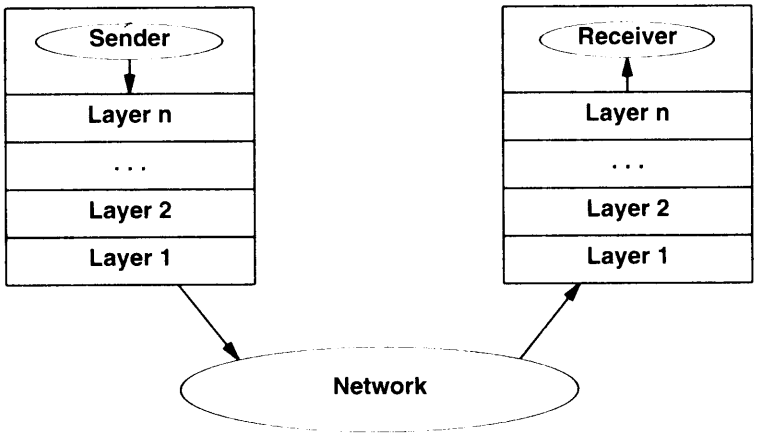


Figure 11.1 The conceptual organization of protocol software in layers.

Conceptually, sending a message from an application program on one machine to an application program on another means transferring the message down through successive layers of protocol software on the sender's machine, forwarding the message across the network, and transferring the message up through successive layers of protocol software on the receiver's machine.

In practice, the protocol software is much more complex than the simple model of Figure 11.1 indicates. Each layer makes decisions about the correctness of the message and chooses an appropriate action based on the message type or destination address. For example, one layer on the receiving machine must decide whether to keep the message or forward it to another machine. Another layer must decide which application program should receive the message.

To understand the difference between the conceptual organization of protocol software and the implementation details, consider the comparison shown in Figure 11.2. The conceptual diagram in Figure 11.2a shows an Internet layer between a high level protocol layer and a network interface layer. The realistic diagram in Figure 11.2b shows that the IP software may communicate with multiple high-level protocol modules and with multiple network interfaces.

Although a diagram of conceptual protocol layering does not show all details, it does help explain the general concept. For example, Figure 11.3 shows the layers of protocol software used by a message that traverses three networks. The diagram shows only the network interface and Internet Protocol layers in the routers because only those layers are needed to receive, route, and send datagrams. We understand that any machine attached to two networks must have two network interface modules, even though the conceptual layering diagram shows only a single network interface layer in each machine.

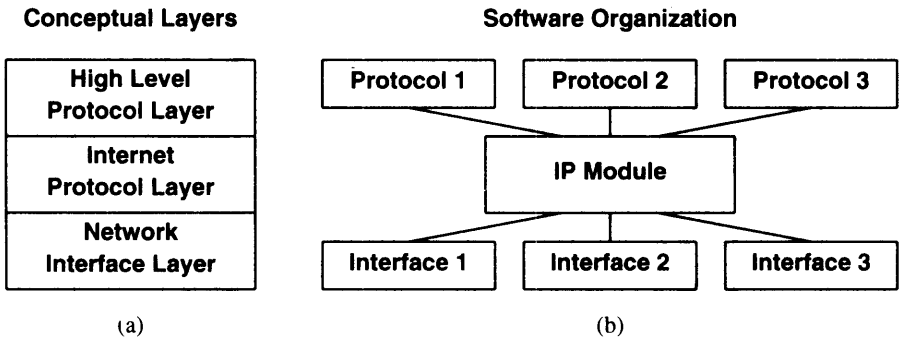


Figure 11.2 A comparison of (a) conceptual protocol layering and (b) a realistic view of software organization showing multiple network interfaces below IP and multiple protocols above it.

As Figure 11.3 shows, a sender on the original machine transmits a message which the IP layer places in a datagram and sends across network *1*. On intermediate routers, the datagram passes up to the IP layer which sends it back out again (on a different network). Only when it reaches the final destination machine, does IP extract the message and pass it up to higher layers of protocol software.

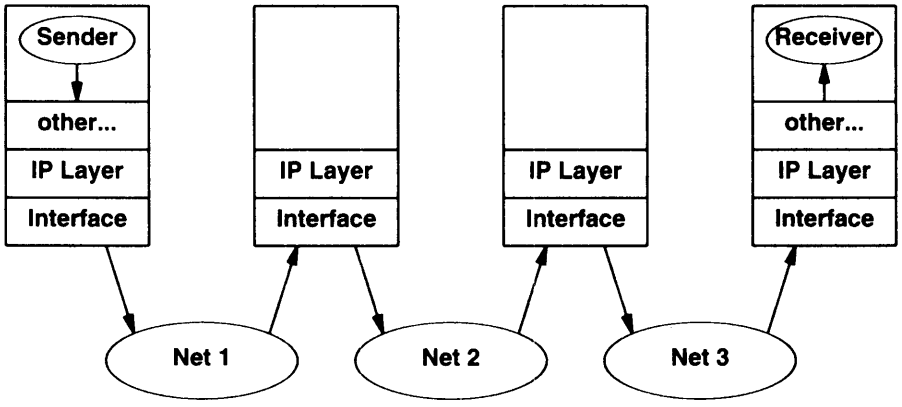


Figure 11.3 The path of a message traversing the Internet from the sender through two intermediate routers to the receiver. Intermediate routers only send the datagram to the IP software layer.

11.4 Functionality Of The Layers

Once the decision has been made to partition the communication problem and organize the protocol software into modules that each handle one subproblem, the question arises: "what functionality should reside in each module?" The question is not easy to answer for several reasons. First, given a set of goals and constraints governing a particular communication problem, it is possible to choose an organization that will optimize protocol software for that problem. Second, even when considering general network-level services such as reliable transport, it is possible to choose from among fundamentally distinct approaches to solving the problem. Third, the design of network (or internet) architecture and the organization of the protocol software are interrelated; one cannot be designed without the other.

11.4.1 ISO 7-Layer Reference Model

Two ideas about protocol layering dominate the field. The first, based on early work done by the International Organization for Standardization (ISO), is known as ISO's *Reference Model of Open System Interconnection*, often referred to as the *ISO model*. The ISO model contains 7 conceptual layers organized as Figure 11.4 shows.

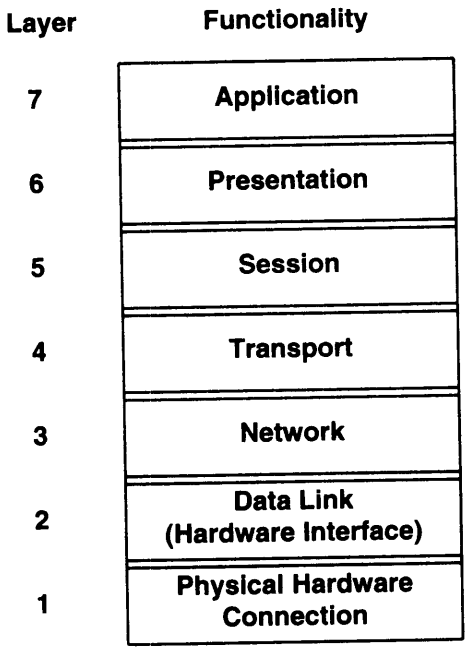


Figure 11.4 The ISO 7-layer reference model for protocol software.

The ISO model, built to describe protocols for a single network, does not contain a specific layer for internetwork routing in the same way TCP/IP protocols do.

11.5 X.25 And Its Relation To The ISO Model

Although it was designed to provide a conceptual model and not an implementation guide, the ISO layering scheme has been the basis for several protocol implementations. Among the protocols commonly associated with the ISO model, the suite of protocols known as X.25 is probably the best known and most widely used. X.25 was established as a recommendation of the *International Telecommunications Union (ITU)*, formerly the *CCITT*, an organization that recommends standards for international telephone services. X.25 has been adopted by public data networks, and became especially popular in Europe. Considering X.25 will help explain ISO layering.

In the X.25 view, a network operates much like a telephone system. An X.25 network is assumed to consist of complex packet switches that contain the intelligence needed to route packets. Hosts do not attach directly to communication wires of the network. Instead each host attaches to one of the packet switches using a serial communication line. In one sense, the connection between a host and an X.25 packet switch is a miniature network consisting of one serial link. The host must follow a complicated procedure to transfer packets onto the network.

• *Physical Layer.* X.25 specifies a standard for the physical interconnection between host computers and network packet switches, as well as the procedures used to transfer packets from one machine to another. In the reference model, layer *1* specifies the physical interconnection including electrical characteristics of voltage and current. A corresponding protocol, X.21, gives the details used by public data networks.

• *Data Link Layer.* The layer *2* portion of the X.25 protocol specifies how data travels between a host and the packet switch to which it connects. X.25 uses the term *frame* to refer to a unit of data as it passes between a host and a packet switch (it is important to understand that the X.25 definition of *frame* differs slightly from the way we have defined it). Because raw hardware delivers only a stream of bits, the layer *2* protocol must define the format of frames and specify how the two machines recognize frame boundaries. Because transmission errors can destroy data, the layer *2* protocol includes error detection (e.g., a frame checksum). Finally, because transmission is unreliable, the layer *2* protocol specifies an exchange of acknowledgements that allows the two machines to know when a frame has been transferred successfully.

One commonly used layer *2* protocol, named the *High Level Data Link Communication*, is best known by its acronym, *HDLC*. Several versions of HDLC exist, with the most recent known as *HDLC/LAPB*. It is important to remember that successful transfer at layer *2* means a frame has been passed to the network packet switch for delivery; it does not guarantee that the packet switch accepted the packet or was able to route it.

• *Network Layer.* The ISO reference model specifies that the third layer contains functionality that completes the definition of the interaction between host and network.

Called the *network* or *communication subnet* layer, this layer defines the basic unit of transfer across the network and includes the concepts of destination addressing and routing. Remember that in the X.25 world, communication between host and packet switch is conceptually isolated from the traffic that is being passed. Thus, the network might allow packets defined by layer 3 protocols to be larger than the size of frames that can be transferred at layer 2. The layer 3 software assembles a packet in the form the network expects and uses layer 2 to transfer it (possibly in pieces) to the packet switch. Layer 3 must also respond to network congestion problems.

• *Transport Layer.* Layer 4 provides end-to-end reliability by having the destination host communicate with the source host. The idea here is that even though lower layers of protocols provide reliable checks at each transfer, the end-to-end layer double checks to make sure that no machine in the middle failed.

• *Session Layer.* Higher layers of the ISO model describe how protocol software can be organized to handle all the functionality needed by application programs. The ISO committee considered the problem of remote terminal access so fundamental that they assigned layer 5 to handle it. In fact, the central service offered by early public data networks consisted of terminal to host interconnection. The carrier provides a special purpose host computer called a *Packet Assembler And Disassembler* (*PAD*) on the network with dialup access. Subscribers, often travelers who carry their own computer and modem, dial up the local PAD, make a network connection to the host with which they wish to communicate, and log in. Many carriers choose to make using the network for long distance communication less expensive than direct dialup.

• *Presentation Layer.* ISO layer 6 is intended to include functions that many application programs need when using the network. Typical examples include standard routines that compress text or convert graphics images into bit streams for transmission across a network. For example an ISO standard known as *Abstract Syntax Notation 1* (*ASN.1*), provides a representation of data that application programs use. One of the TCP/IP protocols, SNMP, also uses ASN.1 to represent data.

• *Application Layer.* Finally, ISO layer 7 includes application programs that use the network. Examples include electronic mail or file transfer programs. In particular, the ITU has devised a protocol for electronic mail known as the *X.400* standard. In fact, the ITU and ISO worked jointly on message handling systems; the ISO version is called *MOTIS*.

11.5.1 The TCP/IP 5-Layer Reference Model

The second major layering model did not arise from a standards committee, but came instead from research that led to the TCP/IP protocol suite. With a little work, the ISO model can be stretched to describe the TCP/IP layering scheme, but the underlying assumptions are different enough to warrant distinguishing the two.

Broadly speaking, TCP/IP software is organized into five conceptual layers — four software layers that build on a fifth layer of hardware. Figure 11.5 shows the conceptual layers as well as the form of data as it passes between them.

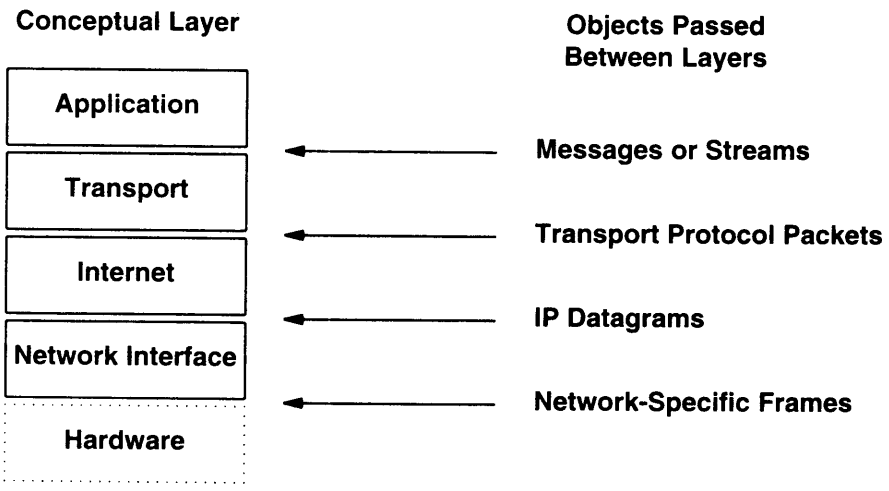


Figure 11.5 The 4 conceptual layers of TCP/IP software above the hardware layer, and the form of objects passed between layers. The layer labeled *network interface* is sometimes called the *data link* layer.

• *Application Layer.* At the highest layer, users invoke application programs that access services available across a TCP/IP internet. An application interacts with one of the transport layer protocols to send or receive data. Each application program chooses the style of transport needed, which can be either a sequence of individual messages or a continuous stream of bytes. The application program passes data in the required form to the transport layer for delivery.

• *Transport Layer.* The primary duty of the *transport layer* is to provide communication from one application program to another. Such communication is often called *end-to-end.* The transport layer may regulate flow of information. It may also provide reliable transport, ensuring that data arrives without error and in sequence. To do so, transport protocol software arranges to have the receiving side send back acknowledgements and the sending side retransmit lost packets. The transport software divides the stream of data being transmitted into small pieces (sometimes called *packets*) and passes each packet along with a destination address to the next layer for transmission.

Although Figure 11.5 uses a single block to represent the application layer, a general purpose computer can have multiple application programs accessing an internet at one time. The transport layer must accept data from several user programs and send it to the next lower layer. To do so, it adds additional information to each packet, includ-

ing codes that identify which application program sent it and which application program should receive it, as well as a checksum. The receiving machine uses the checksum to verify that the packet arrived intact, and uses the destination code to identify the application program to which it should be delivered.

• *Internet Layer.* As we have already seen, the Internet layer handles communication from one machine to another. It accepts a request to send a packet from the transport layer along with an identification of the machine to which the packet should be sent. It encapsulates the packet in an IP datagram, fills in the datagram header, uses the routing algorithm to determine whether to deliver the datagram directly or send it to a router, and passes the datagram to the appropriate network interface for transmission. The Internet layer also handles incoming datagrams, checking their validity, and uses the routing algorithm to decide whether the datagram should be processed locally or forwarded. For datagrams addressed to the local machine, software in the internet layer deletes the datagram header, and chooses from among several transport protocols the one that will handle the packet. Finally, the Internet layer sends and receives ICMP error and control messages as needed.

• *Network Interface Layer.* The lowest layer TCP/IP software comprises a network interface layer, responsible for accepting IP datagrams and transmitting them over a specific network. A network interface may consist of a device driver (e.g., when the network is a local area network to which the machine attaches directly) or a complex subsystem that uses its own data link protocol (e.g., when the network consists of packet switches that communicate with hosts using HDLC).

11.6 Differences Between ISO And Internet Layering

There are two subtle and important differences between the TCP/IP layering scheme and the ISO/X.25 scheme. The first difference revolves around the focus of attention on reliability, while the second involves the location of intelligence in the overall system.

11.6.1 Link-Level vs. End-To-End Reliability

One major difference between the TCP/IP protocols and the X.25 protocols lies in their approaches to providing reliable data transfer services. In the X.25 model, protocol software detects and handles errors at all layers. At the link level, complex protocols guarantee that the transfer between a host and the packet switch to which it connects will be correct. Checksums accompany each piece of data transferred, and the receiver acknowledges each piece of data received. The link layer protocol includes timeout and retransmission algorithms that prevent data loss and provide automatic recovery after hardware fails and restarts.

Successive layers of X.25 provide reliability of their own. At layer *3*, X.25 also provides error detection and recovery for packets transferred onto the network, using checksums as well as timeout and retransmission techniques. Finally, layer *4* must pro-

vide end-to-end reliability, having the source correspond with the ultimate destination to verify delivery.

In contrast to such a scheme, TCP/IP bases its protocol layering on the idea that reliability is an end-to-end problem. The architectural philosophy is simple: construct the internet so it can handle the expected load, but allow individual links or machines to lose data or corrupt it without trying to repeatedly recover. In fact, there is little or no reliability in most TCP/IP network interface layer software. Instead, the transport layer handles most error detection and recovery problems.

The resulting freedom from interface layer verification makes TCP/IP software much easier to understand and implement correctly. Intermediate routers can discard datagrams that become corrupted because of transmission errors or that cannot be delivered. They can discard datagrams when the arrival rate exceeds machine capacity, and can reroute datagrams through paths with shorter or longer delay without informing the source or destination.

Having unreliable links means that some datagrams do not arrive. Detection and recovery of datagram loss is carried out between the source host and the ultimate destination and is, therefore, called *end-to-end* verification. The end-to-end software located in the TCP/IP transport layer uses checksums, acknowledgements, and timeouts to control transmission. Thus, unlike the connection-oriented X.25 protocol layering, the TCP/IP software focuses most of its reliability control in one layer.

11.6.2 Locus of Intelligence and Decision Making

Another difference between the X.25 model and the TCP/IP model emerges when one considers the locus of authority and control. As a general rule, networks using X.25 adhere to the idea that a network is a utility that provides a transport service. The vendor that offers the service controls network access and monitors traffic to keep records for accounting and billing. The network vendor also handles problems like routing, flow control, and acknowledgements internally, making transfers reliable. This view leaves little that the hosts can (or need to) do. In short, the network is a complex, independent system to which one can attach relatively simple host computers; the hosts themselves participate minimally in the network operation.

In contrast, TCP/IP requires hosts to participate in almost all of the network protocols. We have already mentioned that hosts actively implement end-to-end error detection and recovery. They also participate in routing because they must choose a router when sending datagrams, and they participate in network control because they must handle ICMP control messages. Thus, when compared to an X.25 network, a TCP/IP internet can be viewed as a relatively simple packet delivery system to which intelligent hosts attach.

11.7 The Protocol Layering Principle

Independent of the particular layering scheme used or the functions of the layers, the operation of layered protocols is based on a fundamental idea. The idea, called the *layering principle*, can be summarized succinctly:

> *Layered protocols are designed so that layer* n *at the destination receives exactly the same object sent by layer* n *at the source.*

The layering principle explains why layering is such a powerful idea. It allows the protocol designer to focus attention on one layer at a time, without worrying about how other layers perform. For example, when building a file transfer application, the designer considers only two copies of the application program executing on two computers, and concentrates on the messages they need to exchange for file transfer. The designer assumes that the application on one host receives exactly the data that the application on the other host sends.

Figure 11.6 illustrates how the layering principle works:

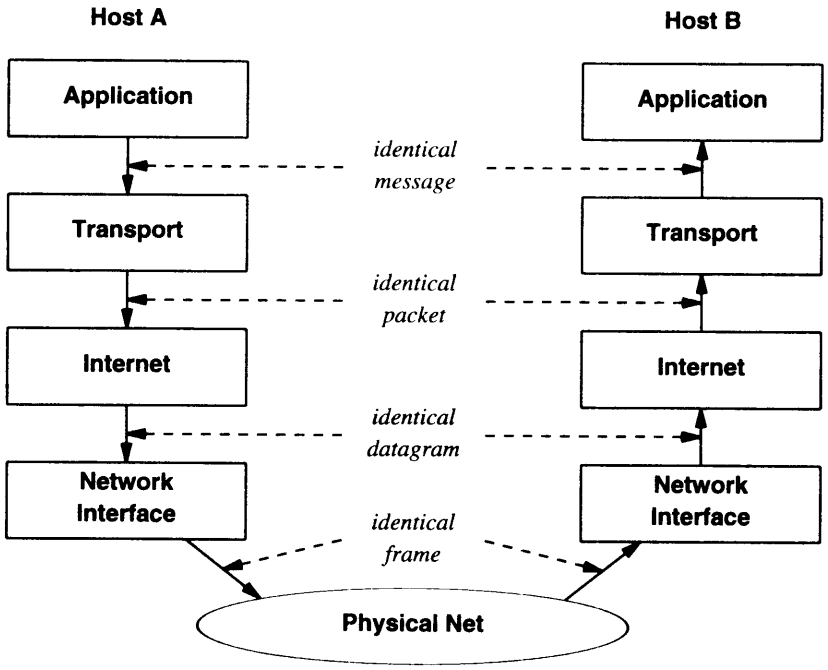


Figure 11.6 The path of a message as it passes from an application on one host to an application on another. Layer *n* on host *B* receives exactly the same object that layer *n* on host *A* sent.

11.7.1 Layering in a TCP/IP Internet Environment

Our statement of the layering principle is somewhat vague, and the illustration in Figure 11.6 skims over an important issue because it fails to distinguish between transfers from source to ultimate destination and transfers across multiple networks. Figure 11.7 illustrates the distinction, showing the path of a message sent from an application program on one host to an application on another through a router.

As the figure shows, message delivery uses two separate network frames, one for the transmission from host *A* to router *R*, and another from router *R* to host *B*. The network layering principle states that the frame delivered to *R* is identical to the frame sent by host *A*. By contrast, the application and transport layers deal with end-to-end issues and are designed so the software at the source communicates with its peer at the ultimate destination. Thus, the layering principle states that the packet received by the transport layer at the ultimate destination is identical to the packet sent by the transport layer at the original source.

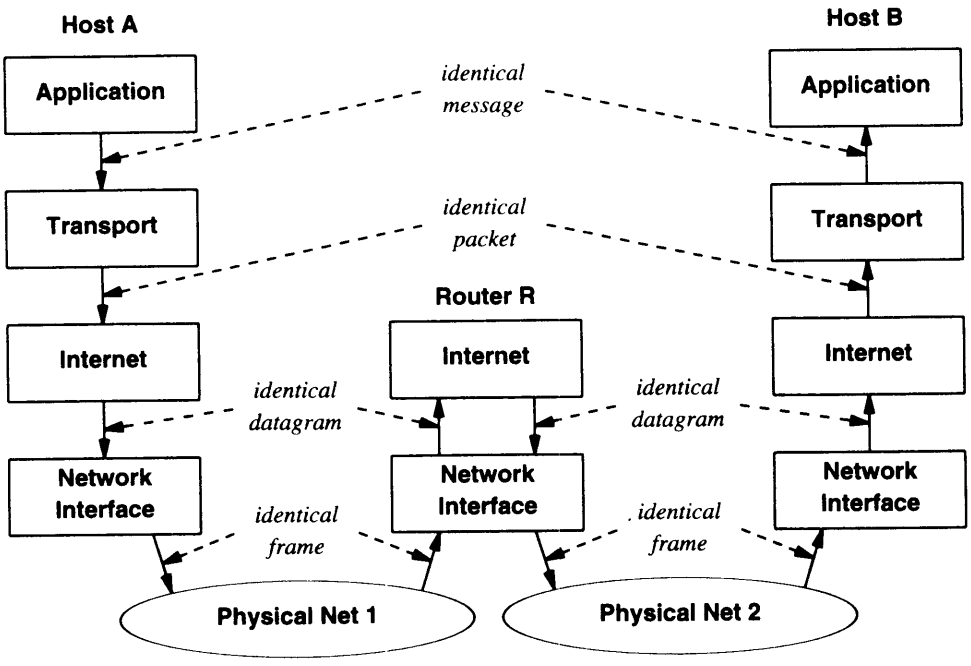


Figure 11.7 The layering principle when a router is used. The frame delivered to router *R* is exactly the frame sent from host *A*, but differs from the frame sent between *R* and *B*.

It is easy to understand that in higher layers, the layering principle applies across end-to-end transfers, and that at the lowest layer it applies to a single machine transfer. It is not as easy to see how the layering principle applies to the Internet layer. On one hand, we have said that hosts attached to an internet should view it as a large, virtual network, with the IP datagram taking the place of a network frame. In this view, datagrams travel from original source to ultimate destination, and the layering principle guarantees that the ultimate destination receives exactly the datagram that the original source sent. On the other hand, we know that the datagram header contains fields, like a *time to live* counter, that change each time the datagram passes through a router. Thus, the ultimate destination will not receive exactly the same datagram as the source sent. We conclude that although most of the datagram stays intact as it passes across an internet, the layering principle only applies to datagrams across single machine transfers. To be accurate, we should not view the Internet layer as providing end-to-end service.

11.8 Layering In The Presence Of Network Substructure

Recall from Chapter 2 that some wide area networks contain multiple packet switches. For example, a WAN can consist of routers that connect to a local network at each site as well as to other routers using leased serial lines. When a router receives a datagram, it either delivers the datagram to its destination on the local network, or transfers the datagram across a serial line to another router. The question arises: "How do the protocols used on serial lines fit into the TCP/IP layering scheme?" The answer depends on how the designer views the serial line interconnections.

From the perspective of IP, the set of point-to-point connections among routers can either function like a set of independent physical networks, or they can function collectively like a single physical network. In the first case, each physical link is treated exactly like any other network in the internet. The link is assigned a unique network number, and the two hosts that share the link each have a unique IP address assigned for their connection†. Routes are added to the IP routing table as they would be for any other network. A new software module is added at the network interface layer to control the new link hardware, but no substantial changes are made to the layering scheme. The main disadvantage of the independent network approach is that it proliferates network numbers (one for each connection between two machines) and causes routing tables to be larger than necessary. Both *Serial Line IP* (*SLIP*) and the *Point to Point Protocol* (*PPP*) treat each serial link as a separate network.

The second approach to accommodating point-to-point connections avoids assigning multiple IP addresses to the physical wires. Instead, it treats all the connections collectively as a single, independent IP network with its own frame format, hardware addressing scheme, and data link protocols. Routers that use the second approach need only one IP network number for all point-to-point connections.

Using the single network approach means extending the protocol layering scheme to add a new intranetwork routing layer between the network interface layer and the

†The only exception arises when using the anonymous network scheme described in Chapter 10; leaving the link unnumbered does not change the layering.

hardware devices. For machines with only one point-to-point connection, an additional layer seems unnecessary. To see why it is needed, consider a machine with several physical point-to-point connections, and recall from Figure 11.2 how the network interface layer is divided into multiple software modules that each control one network. We need to add one network interface for the new point-to-point network, but the new interface must control multiple hardware devices. Furthermore, given a datagram to send, the new interface must choose the correct link over which the datagram should be sent. Figure 11.8 shows the organization.

The Internet layer software passes to the network interface all datagrams that should be sent on any of the point-to-point connections. The network interface passes them to the intranet routing module that must further distinguish among multiple physical connections and route the datagram across the correct one.

The programmer who designs the intranet routing software determines exactly how the software chooses a physical link. Usually, the algorithm relies on an intranet routing table. The intranet routing table is analogous to the internet routing table in that it specifies a mapping of destination address to route. The table contains pairs of entries, (D, L), where D is a destination host address and L specifies the physical line used to reach that destination.

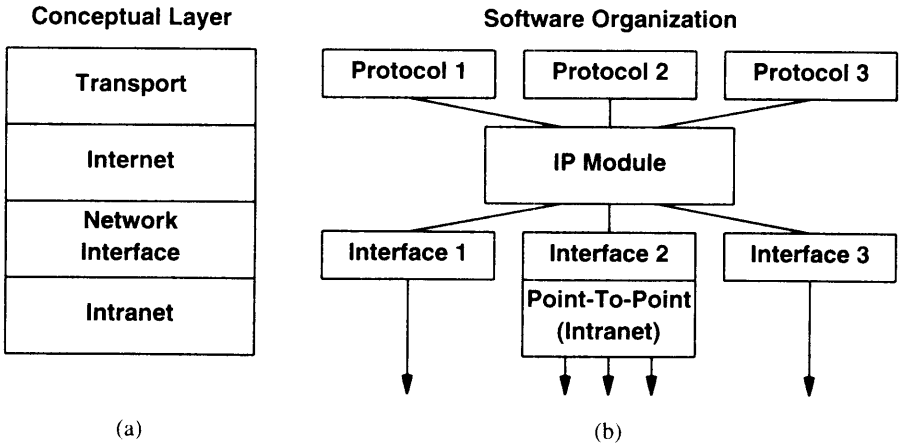


Figure 11.8 (a) Conceptual position of an intranet protocol for point-to-point connections when IP treats them as a single IP network, and (b) detailed diagram of corresponding software modules. Each arrow corresponds to one physical device.

The difference between an internet routing table and an intranet routing table is that intranet routing tables are quite small. They only contain routing information for hosts directly attached to the point-to-point network. The reason is simple: the Internet layer maps an arbitrary destination address to a specific router address before passing

the datagram to a network interface. The intranet layer is asked only to distinguish among machines on a single point-to-point network.

11.9 Two Important Boundaries In The TCP/IP Model

The conceptual protocol layering includes two boundaries that may not be obvious: a protocol address boundary that separates high-level and low-level addressing, and an operating system boundary that separates the system from application programs.

11.9.1 High-Level Protocol Address Boundary

Now that we have seen the layering of TCP/IP software, we can be precise about an idea introduced in Chapter 8: a conceptual boundary partitions software that uses low-level (physical) addresses from software that uses high-level (IP) addresses. As Figure 11.9 shows, the boundary occurs between the network interface layer and the Internet layer. That is,

> *Application programs as well as all protocol software from the Internet layer upward use only IP addresses; the network interface layer handles physical addresses.*

Thus, protocols like ARP belong in the network interface layer. They are not part of IP.

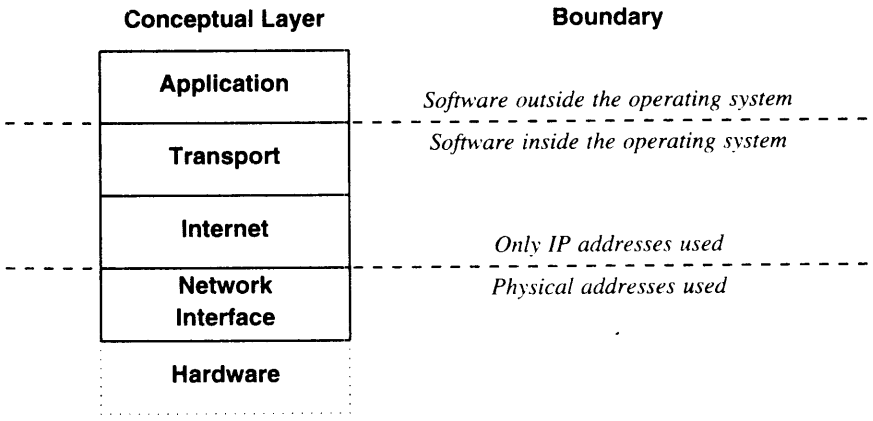


Figure 11.9 The relationship between conceptual layering and the boundaries for operating system and high-level protocol addresses.

11.9.2 Operating System Boundary

Figure 11.9 shows another important boundary as well, the division between software that is generally considered part of the operating system and software that is not. While each implementation of TCP/IP chooses how to make the distinction, many follow the scheme shown. Because they lie inside the operating system, passing data between lower layers of protocol software is much less expensive than passing it between an application program and a transport layer. Chapter 20 discusses the problem in more detail and describes an example of the interface an operating system might provide.

11.10 The Disadvantage Of Layering

We have said that layering is a fundamental idea that provides the basis for protocol design. It allows the designer to divide a complicated problem into subproblems and solve each one independently. Unfortunately, the software that results from strict layering can be extremely inefficient. As an example, consider the job of the transport layer. It must accept a stream of bytes from an application program, divide the stream into packets, and send each packet across the internet. To optimize transfer, the transport layer should choose the largest possible packet size that will allow one packet to travel in one network frame. In particular, if the destination machine attaches directly to one of the same networks as the source, only one physical net will be involved in the transfer, so the sender can optimize packet size for that network. If the software preserves strict layering, however, the transport layer cannot know how the Internet module will route traffic or which networks attach directly. Furthermore, the transport layer will not understand the datagram or frame formats nor will it be able to determine how many octets of header will be added to a packet. Thus, strict layering will prevent the transport layer from optimizing transfers.

Usually, implementors relax the strict layering scheme when building protocol software. They allow information like route selection and network MTU to propagate upward. When allocating buffers, they often leave space for headers that will be added by lower layer protocols and may retain headers on incoming frames when passing them to higher layer protocols. Such optimizations can make dramatic improvements in efficiency while retaining the basic layered structure.

11.11 The Basic Idea Behind Multiplexing And Demultiplexing

Communication protocols use techniques of *multiplexing* and *demultiplexing* throughout the layered hierarchy. When sending a message, the source computer includes extra bits that encode the message type, originating program, and protocols used.

Eventually, all messages are placed into network frames for transfer and combined into a stream of packets. At the receiving end, the destination machine uses the extra information to guide processing.

Consider an example of demultiplexing shown in Figure 11.10.

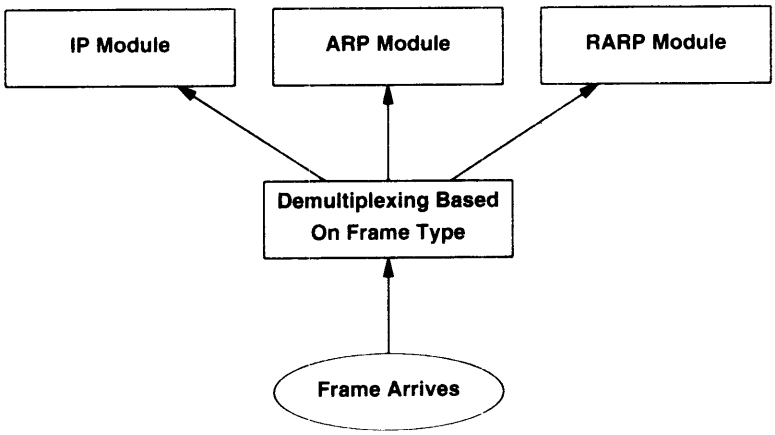


Figure 11.10 Demultiplexing of incoming frames based on the type field found in the frame header.

The figure illustrates how software in the network interface layer uses the frame type to choose a procedure to handle the incoming frame. We say that the network interface *demultiplexes* the frame based on its type. To make such a choice possible, software in the source machine must set the frame type field before transmission. Thus, each software module that sends frames uses the type field to specify frame contents.

Multiplexing and demultiplexing occur at almost every protocol layer. For example, after the network interface demultiplexes frames and passes those frames that contain IP datagrams to the IP module, the IP software extracts the datagram and demultiplexes further based on the transport protocol. Figure 11.11 demonstrates demultiplexing at the Internet layer.

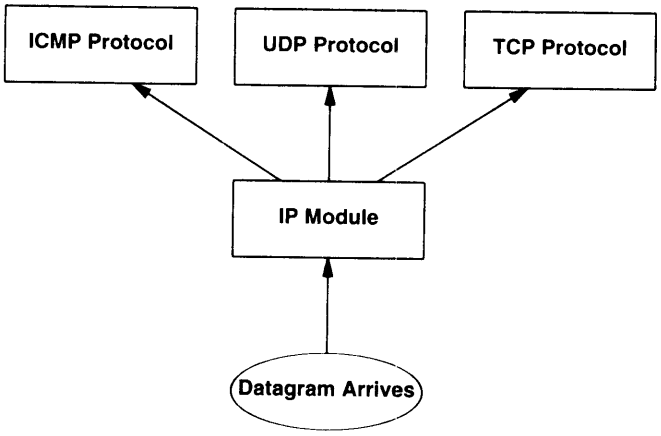


Figure 11.11 Demultiplexing at the Internet layer. IP software chooses an appropriate procedure to handle a datagram based on the protocol type field in the datagram header.

To decide how to handle a datagram, internet software examines the header of a datagram and selects a protocol handler based on the datagram type. In the example, the possible datagram types are: *ICMP*, which we have already examined, and *UDP*, and *TCP*, which we will examine in later chapters.

11.12 Summary

Protocols are the standards that specify how data is represented when being transferred from one machine to another. Protocols specify how the transfer occurs, how errors are detected, and how acknowledgements are passed. To simplify protocol design and implementation, communication problems are segregated into subproblems that can be solved independently. Each subproblem is assigned a separate protocol.

The idea of layering is fundamental because it provides a conceptual framework for protocol design. In a layered model, each layer handles one part of the communication problem and usually corresponds to one protocol. Protocols follow the layering principle, which states that the software implementing layer n on the destination machine receives exactly what the software implementing layer n on the source machine sends.

We examined the 5-layer Internet reference model as well as the older ISO 7-layer reference model. In both cases, the layering model provides only a conceptual framework for protocol software. The ITU X.25 protocols follow the ISO reference model and provide an example of reliable communication service offered by a commercial utility, while the TCP/IP protocols provide an example of a different layering scheme.

In practice, protocol software uses multiplexing and demultiplexing to distinguish among multiple protocols within a given layer, making protocol software more complex than the layering model suggests.

FOR FURTHER STUDY

Postel [RFC 791] provides a sketch of the Internet Protocol layering scheme, and Clark [RFC 817] discusses the effect of layering on implementations. Saltzer, Reed, and Clark [1984] argues that end-to-end verification is important. Chesson [1987] makes the controversial argument that layering produces intolerably bad network throughput. Volume 2 of this text examines layering in detail, and shows an example implementation that achieves efficiency by compromising strict layering and passing pointers between layers.

The ISO protocol documents [1987a] and [1987b] describe ASN.1 in detail. Sun [RFC 1014] describes XDR, an example of what might be called a TCP/IP presentation protocol. Clark [1985] discusses passing information upward through layers.

EXERCISES

11.1 Study the ISO layering model in more detail. How well does the model describe communication on a local area network like an Ethernet?

11.2 Build a case that TCP/IP is moving toward a six-layer protocol architecture that includes a presentation layer. (Hint: various programs use the XDR protocol, Courier-Dandi, ASN.1.)

11.3 Do you think any single presentation protocol will eventually emerge that replaces all others? Why or why not?

11.4 Compare and contrast the tagged data format used by the ASN.1 presentation scheme with the untagged format used by XDR. Characterize situations in which one is better than the other.

11.5 Find out how a UNIX system uses the *mbuf* structure to make layered protocol software efficient.

11.6 Read about the System V UNIX *streams* mechanism. How does it help make protocol implementation easier? What is its chief disadvantage?

12

User Datagram Protocol (UDP)

12.1 Introduction

Previous chapters describe a TCP/IP internet capable of transferring IP datagrams among host computers, where each datagram is routed through the internet based on the destination's IP address. At the Internet Protocol layer, a destination address identifies a host computer; no further distinction is made regarding which user or which application program will receive the datagram. This chapter extends the TCP/IP protocol suite by adding a mechanism that distinguishes among destinations within a given host, allowing multiple application programs executing on a given computer to send and receive datagrams independently.

12.2 Identifying The Ultimate Destination

The operating systems in most computers support multiprogramming, which means they permit multiple application programs to execute simultaneously. Using operating system jargon, we refer to each executing program as a *process*, *task*, *application program*, or a *user level process*; the systems are called multitasking systems. It may seem natural to say that a process is the ultimate destination for a message. However, specifying that a particular process on a particular machine is the ultimate destination for a datagram is somewhat misleading. First, because processes are created and destroyed dynamically, senders seldom know enough to identify a process on another machine. Second, we would like to be able to replace processes that receive datagrams without

informing all senders (e.g., rebooting a machine can change all the processes, but senders should not be required to know about the new processes). Third, we need to identify destinations from the functions they implement without knowing the process that implements the function (e.g., to allow a sender to contact a file server without knowing which process on the destination machine implements the file server function). More important, in systems that allow a single process to handle two or more functions, it is essential that we arrange a way for a process to decide exactly which function the sender desires.

Instead of thinking of a process as the ultimate destination, we will imagine that each machine contains a set of abstract destination points called *protocol ports*. Each protocol port is identified by a positive integer. The local operating system provides an interface mechanism that processes use to specify a port or access it.

Most operating systems provide synchronous access to ports. From a particular process's point of view, synchronous access means the computation stops during a port access operation. For example, if a process attempts to extract data from a port before any data arrives, the operating system temporarily stops (blocks) the process until data arrives. Once the data arrives, the operating system passes the data to the process and restarts it. In general, ports are *buffered*, so data that arrives before a process is ready to accept it will not be lost. To achieve buffering, the protocol software located inside the operating system places packets that arrive for a particular protocol port in a (finite) queue until a process extracts them.

To communicate with a foreign port, a sender needs to know both the IP address of the destination machine and the protocol port number of the destination within that machine. Each message must carry the number of the *destination port* on the machine to which the message is sent, as well as the *source port* number on the source machine to which replies should be addressed. Thus, it is possible for any process that receives a message to reply to the sender.

12.3 The User Datagram Protocol

In the TCP/IP protocol suite, the *User Datagram Protocol* or *UDP* provides the primary mechanism that application programs use to send datagrams to other application programs. UDP provides protocol ports used to distinguish among multiple programs executing on a single machine. That is, in addition to the data sent, each UDP message contains both a destination port number and a source port number, making it possible for the UDP software at the destination to deliver the message to the correct recipient and for the recipient to send a reply.

UDP uses the underlying Internet Protocol to transport a message from one machine to another, and provides the same unreliable, connectionless datagram delivery semantics as IP. It does not use acknowledgements to make sure messages arrive, it does not order incoming messages, and it does not provide feedback to control the rate at which information flows between the machines. Thus, UDP messages can be lost, duplicated, or arrive out of order. Furthermore, packets can arrive faster than the recipient can process them. We can summarize:

The User Datagram Protocol (UDP) provides an unreliable connectionless delivery service using IP to transport messages between machines. It uses IP to carry messages, but adds the ability to distinguish among multiple destinations within a given host computer.

An application program that uses UDP accepts full responsibility for handling the problem of reliability, including message loss, duplication, delay, out-of-order delivery, and loss of connectivity. Unfortunately, application programmers often ignore these problems when designing software. Furthermore, because programmers often test network software using highly reliable, low-delay local area networks, testing may not expose potential failures. Thus, many application programs that rely on UDP work well in a local environment but fail in dramatic ways when used in a larger TCP/IP internet.

12.4 Format Of UDP Messages

Each UDP message is called a *user datagram*. Conceptually, a user datagram consists of two parts: a UDP header and a UDP data area. As Figure 12.1 shows, the header is divided into four 16-bit fields that specify the port from which the message was sent, the port to which the message is destined, the message length, and a UDP checksum.

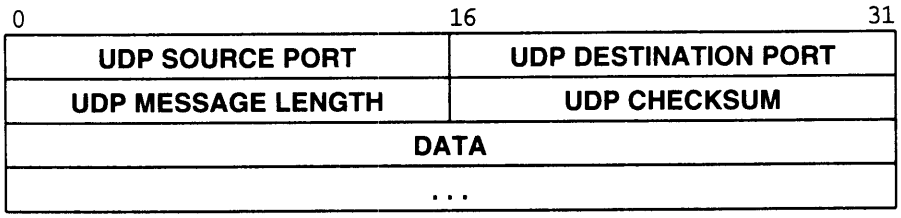


Figure 12.1 The format of fields in a UDP datagram.

The *SOURCE PORT* and *DESTINATION PORT* fields contain the 16-bit UDP protocol port numbers used to demultiplex datagrams among the processes waiting to receive them. The *SOURCE PORT* is optional. When used, it specifies the port to which replies should be sent; if not used, it should be zero.

The *LENGTH* field contains a count of octets in the UDP datagram, including the UDP header and the user data. Thus, the minimum value for *LENGTH* is eight, the length of the header alone.

The UDP checksum is optional and need not be used at all; a value of zero in the *CHECKSUM* field means that the checksum has not been computed. The designers chose to make the checksum optional to allow implementations to operate with little

computational overhead when using UDP across a highly reliable local area network. Recall, however, that IP does not compute a checksum on the data portion of an IP datagram. Thus, the UDP checksum provides the only way to guarantee that data has arrived intact and should be used.

Beginners often wonder what happens to UDP messages for which the computed checksum is zero. A computed value of zero is possible because UDP uses the same checksum algorithm as IP: it divides the data into 16-bit quantities and computes the one's complement of their one's complement sum. Surprisingly, zero is not a problem because one's complement arithmetic has two representations for zero: all bits set to zero or all bits set to one. When the computed checksum is zero, UDP uses the representation with all bits set to one.

12.5 UDP Pseudo-Header

The UDP checksum covers more information than is present in the UDP datagram alone. To compute the checksum, UDP prepends a *pseudo-header* to the UDP datagram, appends an octet of zeros to pad the datagram to an exact multiple of 16 bits, and computes the checksum over the entire object. The octet used for padding and the pseudo-header are *not* transmitted with the UDP datagram, nor are they included in the length. To compute a checksum, the software first stores zero in the *CHECKSUM* field, then accumulates a 16-bit one's complement sum of the entire object, including the pseudo-header, UDP header, and user data.

The purpose of using a pseudo-header is to verify that the UDP datagram has reached its correct destination. The key to understanding the pseudo-header lies in realizing that the correct destination consists of a specific machine and a specific protocol port within that machine. The UDP header itself specifies only the protocol port number. Thus, to verify the destination, UDP on the sending machine computes a checksum that covers the destination IP address as well as the UDP datagram. At the ultimate destination, UDP software verifies the checksum using the destination IP address obtained from the header of the IP datagram that carried the UDP message. If the checksums agree, then it must be true that the datagram has reached the intended destination host as well as the correct protocol port within that host.

The pseudo-header used in the UDP checksum computation consists of 12 octets of data arranged as Figure 12.2 shows. The fields of the pseudo-header labeled *SOURCE IP ADDRESS* and *DESTINATION IP ADDRESS* contain the source and destination IP addresses that will be used when sending the UDP message. Field *PROTO* contains the IP protocol type code (*17* for UDP), and the field labeled *UDP LENGTH* contains the length of the UDP datagram (not including the pseudo-header). To verify the checksum, the receiver must extract these fields from the IP header, assemble them into the pseudo-header format, and recompute the checksum.

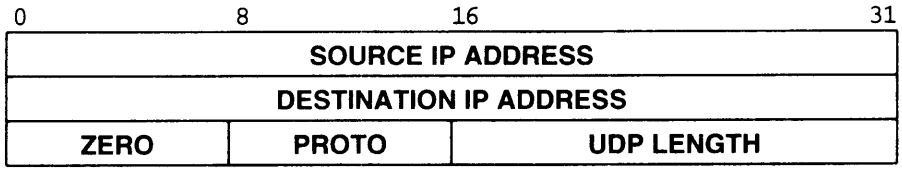


Figure 12.2 The 12 octets of the pseudo-header used during UDP checksum computation.

12.6 UDP Encapsulation And Protocol Layering

UDP provides our first example of a transport protocol. In the layering model of Chapter 11, UDP lies in the layer above the Internet Protocol layer. Conceptually, application programs access UDP, which uses IP to send and receive datagrams as Figure 12.3 shows.

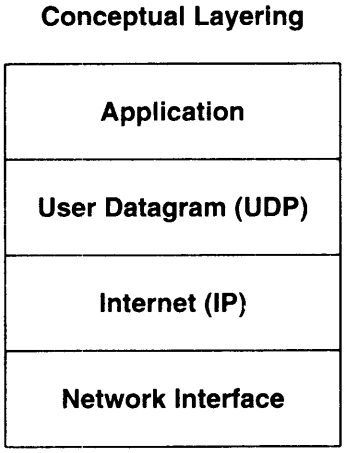


Figure 12.3 The conceptual layering of UDP between application programs and IP.

Layering UDP above IP means that a complete UDP message, including the UDP header and data, is encapsulated in an IP datagram as it travels across an internet as Figure 12.4 shows.

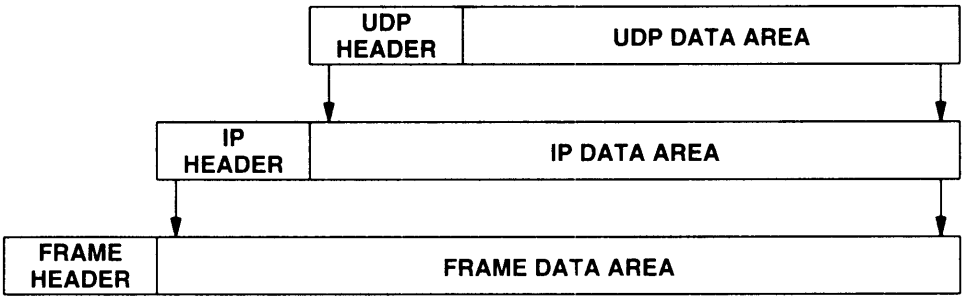


Figure 12.4 A UDP datagram encapsulated in an IP datagram for transmission across an internet. The datagram is further encapsulated in a frame each time it travels across a single network.

For the protocols we have examined, encapsulation means that UDP prepends a header to the data that a user sends and passes it to IP. The IP layer prepends a header to what it receives from UDP. Finally, the network interface layer embeds the datagram in a frame before sending it from one machine to another. The format of the frame depends on the underlying network technology. Usually, network frames include an additional header.

On input, a packet arrives at the lowest layer of network software and begins its ascent through successively higher layers. Each layer removes one header before passing the message on, so that by the time the highest level passes data to the receiving process, all headers have been removed. Thus, the outermost header corresponds to the lowest layer of protocol, while the innermost header corresponds to the highest protocol layer. When considering how headers are inserted and removed, it is important to keep in mind the layering principle. In particular, observe that the layering principle applies to UDP, so the UDP datagram received from IP on the destination machine is identical to the datagram that UDP passed to IP on the source machine. Also, the data that UDP delivers to a user process on the receiving machine will be exactly the data that a user process passed to UDP on the sending machine.

The division of duties among various protocol layers is rigid and clear:

> *The IP layer is responsible only for transferring data between a pair of hosts on an internet, while the UDP layer is responsible only for differentiating among multiple sources or destinations within one host.*

Thus, only the IP header identifies the source and destination hosts; only the UDP layer identifies the source or destination ports within a host.

12.7 Layering And The UDP Checksum Computation

Observant readers will have noticed a seeming contradiction between the layering rules and the UDP checksum computation. Recall that the UDP checksum includes a pseudo-header that has fields for the source and destination IP addresses. It can be argued that the destination IP address must be known to the user when sending a UDP datagram, and the user must pass it to the UDP layer. Thus, the UDP layer can obtain the destination IP address without interacting with the IP layer. However, the source IP address depends on the route IP chooses for the datagram, because the IP source address identifies the network interface over which the datagram is transmitted. Thus, UDP cannot know a source IP address unless it interacts with the IP layer.

We assume that UDP software asks the IP layer to compute the source and (possibly) destination IP addresses, uses them to construct a pseudo-header, computes the checksum, discards the pseudo-header, and then passes the UDP datagram to IP for transmission. An alternative approach that produces greater efficiency arranges to have the UDP layer encapsulate the UDP datagram in an IP datagram, obtain the source address from IP, store the source and destination addresses in the appropriate fields of the datagram header, compute the UDP checksum, and then pass the IP datagram to the IP layer, which only needs to fill in the remaining IP header fields.

Does the strong interaction between UDP and IP violate our basic premise that layering reflects separation of functionality? Yes. UDP has been tightly integrated with the IP protocol. It is clearly a compromise of the pure separation, made for entirely practical reasons. We are willing to overlook the layering violation because it is impossible to fully identify a destination application program without specifying the destination machine, and we want to make the mapping between addresses used by UDP and those used by IP efficient. One of the exercises examines this issue from a different point of view, asking the reader to consider whether UDP should be separated from IP.

12.8 UDP Multiplexing, Demultiplexing, And Ports

We have seen in Chapter 11 that software throughout the layers of a protocol hierarchy must multiplex or demultiplex among multiple objects at the next layer. UDP software provides another example of multiplexing and demultiplexing. It accepts UDP datagrams from many application programs and passes them to IP for transmission, and it accepts arriving UDP datagrams from IP and passes each to the appropriate application program.

Conceptually, all multiplexing and demultiplexing between UDP software and application programs occur through the port mechanism. In practice, each application program must negotiate with the operating system to obtain a protocol port and an associated port number before it can send a UDP datagram†. Once the port has been assigned, any datagram the application program sends through the port will have that port number in its UDP *SOURCE PORT* field.

---

†For now, we will describe ports abstractly; Chapter 22 provides an example of the operating system primitives used to create and use ports.

While processing input, UDP accepts incoming datagrams from the IP software and demultiplexes based on the UDP destination port, as Figure 12.5 shows.

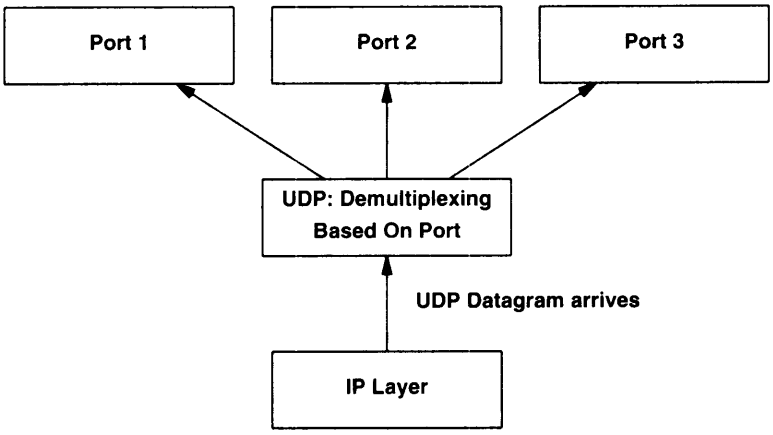


Figure 12.5 Example of demultiplexing one layer above IP. UDP uses the UDP destination port number to select an appropriate destination port for incoming datagrams.

The easiest way to think of a UDP port is as a queue. In most implementations, when an application program negotiates with the operating system to use a given port, the operating system creates an internal queue that can hold arriving messages. Often, the application can specify or change the queue size. When UDP receives a datagram, it checks to see that the destination port number matches one of the ports currently in use. If not, it sends an ICMP *port unreachable* error message and discards the datagram. If a match is found, UDP enqueues the new datagram at the port where an application program can access it. Of course, an error occurs if the port is full, and UDP discards the incoming datagram.

12.9 Reserved And Available UDP Port Numbers

How should protocol port numbers be assigned? The problem is important because two computers need to agree on port numbers before they can interoperate. For example, when computer *A* wants to obtain a file from computer *B*, it needs to know what port the file transfer program on computer *B* uses. There are two fundamental approaches to port assignment. The first approach uses a central authority. Everyone agrees to allow a central authority to assign port numbers as needed and to publish the list of all assignments. Then all software is built according to the list. This approach is sometimes called *universal assignment*, and the port assignments specified by the authority are called *well-known port assignments*.

The second approach to port assignment uses dynamic binding. In the dynamic binding approach, ports are not globally known. Instead, whenever a program needs a port, the network software assigns one. To learn about the current port assignment on another computer, it is necessary to send a request that asks about the current port assignment (e.g., What port is the file transfer service using?). The target machine replies by giving the correct port number to use.

The TCP/IP designers adopted a hybrid approach that assigns some port numbers a priori, but leaves many available for local sites or application programs. The assigned port numbers begin at low values and extend upward, leaving large integer values available for dynamic assignment. The table in Figure 12.6 lists some of the currently assigned UDP port numbers. The second column contains Internet standard assigned keywords, while the third contains keywords used on most UNIX systems.

Decimal	Keyword	UNIX Keyword	Description
0	-	-	Reserved
7	ECHO	echo	Echo
9	DISCARD	discard	Discard
11	USERS	systat	Active Users
13	DAYTIME	daytime	Daytime
15	-	netstat	Network status program
17	QUOTE	qotd	Quote of the Day
19	CHARGEN	chargen	Character Generator
37	TIME	time	Time
42	NAMESERVER	name	Host Name Server
43	NICNAME	whois	Who Is
53	DOMAIN	nameserver	Domain Name Server
67	BOOTPS	bootps	BOOTP or DHCP Server
68	BOOTPC	bootpc	BOOTP or DHCP Client
69	TFTP	tftp	Trivial File Transfer
88	KERBEROS	kerberos	Kerberos Security Service
111	SUNRPC	sunrpc	Sun Remote Procedure Call
123	NTP	ntp	Network Time Protocol
161	-	snmp	Simple Network Management Protocol
162	-	snmp-trap	SNMP traps
512	-	biff	UNIX comsat
513	-	who	UNIX rwho daemon
514	-	syslog	System log
525	-	timed	Time daemon

Figure 12.6 An illustrative sample of currently assigned UDP ports showing the standard keyword and the UNIX equivalent; the list is not exhaustive. To the extent possible, other transport protocols that offer identical services use the same port numbers as UDP.

12.10 Summary

Most computer systems permit multiple application programs to execute simultaneously. Using operating system jargon, we refer to each executing program as a *process*. The User Datagram Protocol, UDP, distinguishes among multiple processes within a given machine by allowing senders and receivers to add two 16-bit integers called protocol port numbers to each UDP message. The port numbers identify the source and destination. Some UDP port numbers, called *well known*, are permanently assigned and honored throughout the Internet (e.g., port *69* is reserved for use by the trivial file transfer protocol *TFTP* described in Chapter 26). Other port numbers are available for arbitrary application programs to use.

UDP is a thin protocol in the sense that it does not add significantly to the semantics of IP. It merely provides application programs with the ability to communicate using IP's unreliable connectionless packet delivery service. Thus, UDP messages can be lost, duplicated, delayed, or delivered out of order; the application program using UDP must handle these problems. Many programs that use UDP do not work correctly across an internet because they fail to accommodate these conditions.

In the protocol layering scheme, UDP lies in the transport layer, above the Internet Protocol layer and below the application layer. Conceptually, the transport layer is independent of the Internet layer, but in practice they interact strongly. The UDP checksum includes IP source and destination addresses, meaning that UDP software must interact with IP software to find addresses before sending datagrams.

FOR FURTHER STUDY

Tanenbaum [1981] contains a tutorial comparison of the datagram and virtual circuit models of communication. Ball *et. al.* [1979] describes message-based systems without discussing the message protocol. The UDP protocol described here is a standard for TCP/IP and is defined by Postel [RFC 768].

EXERCISES

12.1 Try UDP in your local environment. Measure the average transfer speed with messages of 256, 512, 1024, 2048, 4096, and 8192 bytes. Can you explain the results (hint: what is your network MTU)?

12.2 Why is the UDP checksum separate from the IP checksum? Would you object to a protocol that used a single checksum for the complete IP datagram including the UDP message?

12.3 Not using checksums can be dangerous. Explain how a single corrupted ARP packet broadcast by machine *P* can make it impossible to reach another machine, *Q*.

12.4 Should the notion of multiple destinations identified by protocol ports have been built into IP? Why, or why not?

12.5 *Name Registry.* Suppose you want to allow arbitrary pairs of application programs to establish communication with UDP, but you do not wish to assign them fixed UDP port numbers. Instead, you would like potential correspondents to be identified by a character string of 64 or fewer characters. Thus, a program on machine *A* might want to communicate with the "funny-special-long-id" program on machine *B* (you can assume that a process always knows the IP address of the host with which it wants to communicate). Meanwhile, a process on machine *C* wants to communicate with the "comer's-own-program-id" on machine *A*. Show that you only need to assign one UDP port to make such communication possible by designing software on each machine that allows (a) a local process to pick an unused UDP port ID over which it will communicate, (b) a local process to register the 64-character name to which it responds, and (c) a foreign process to use UDP to establish communication using only the 64-character name and destination internet address.

12.6 Implement name registry software from the previous exercise.

12.7 What is the chief advantage of using preassigned UDP port numbers? The chief disadvantage?

12.8 What is the chief advantage of using protocol ports instead of process identifiers to specify the destination within a machine?

12.9 UDP provides unreliable datagram communication because it does not guarantee delivery of the message. Devise a reliable datagram protocol that uses timeouts and acknowledgements to guarantee delivery. How much network overhead and delay does reliability introduce?

12.10 Send UDP datagrams across a wide area network and measure the percentage lost and the percentage reordered. Does the result depend on the time of day? The network load?

13

Reliable Stream Transport Service (TCP)

13.1 Introduction

Previous chapters explore the unreliable connectionless packet delivery service that forms the basis for all internet communication and the IP protocol that defines it. This chapter introduces the second most important and well-known network-level service, reliable stream delivery, and the *Transmission Control Protocol (TCP)* that defines it. We will see that TCP adds substantial functionality to the protocols already discussed, but that its implementation is also substantially more complex.

Although TCP is presented here as part of the TCP/IP Internet protocol suite, it is an independent, general purpose protocol that can be adapted for use with other delivery systems. For example, because TCP makes very few assumptions about the underlying network, it is possible to use it over a single network like an Ethernet, as well as over a complex internet. In fact, TCP has been so popular that one of the International Organization for Standardization's open systems protocols, TP-4, has been derived from it.

13.2 The Need For Stream Delivery

At the lowest level, computer communication networks provide unreliable packet delivery. Packets can be lost or destroyed when transmission errors interfere with data, when network hardware fails, or when networks become too heavily loaded to accommodate the load presented. Networks that route packets dynamically can deliver them out of order, deliver them after a substantial delay, or deliver duplicates. Furthermore,

underlying network technologies may dictate an optimal packet size or pose other constraints needed to achieve efficient transfer rates.

At the highest level, application programs often need to send large volumes of data from one computer to another. Using an unreliable connectionless delivery system for large volume transfers becomes tedious and annoying, and it requires programmers to build error detection and recovery into each application program. Because it is difficult to design, understand, or modify software that correctly provides reliability, few application programmers have the necessary technical background. As a consequence, one goal of network protocol research has been to find general purpose solutions to the problems of providing reliable stream delivery, making it possible for experts to build a single instance of stream protocol software that all application programs use. Having a single general purpose protocol helps isolate application programs from the details of networking, and makes it possible to define a uniform interface for the stream transfer service.

13.3 Properties Of The Reliable Delivery Service

The interface between application programs and the TCP/IP reliable delivery service can be characterized by 5 features:

• *Stream Orientation.* When two application programs (user processes) transfer large volumes of data, we think of the data as a *stream* of bits, divided into 8-bit *octets*, which are informally called *bytes*. The stream delivery service on the destination machine passes to the receiver exactly the same sequence of octets that the sender passes to it on the source machine.

• *Virtual Circuit Connection.* Making a stream transfer is analogous to placing a telephone call. Before transfer can start, both the sending and receiving application programs interact with their respective operating systems, informing them of the desire for a stream transfer. Conceptually, one application places a "call" which must be accepted by the other. Protocol software modules in the two operating systems communicate by sending messages across an internet, verifying that the transfer is authorized, and that both sides are ready. Once all details have been settled, the protocol modules inform the application programs that a *connection* has been established and that transfer can begin. During transfer, protocol software on the two machines continue to communicate to verify that data is received correctly. If the communication fails for any reason (e.g., because network hardware along the path between the machines fails), both machines detect the failure and report it to the appropriate application programs. We use the term *virtual circuit* to describe such connections because although application programs view the connection as a dedicated hardware circuit, the reliability is an illusion provided by the stream delivery service.

• *Buffered Transfer.* Application programs send a data stream across the virtual circuit by repeatedly passing data octets to the protocol software. When transferring data, each application uses whatever size pieces it finds convenient, which can be as small as a single octet. At the receiving end, the protocol software delivers octets from

the data stream in exactly the same order they were sent, making them available to the receiving application program as soon as they have been received and verified. The protocol software is free to divide the stream into packets independent of the pieces the application program transfers. To make transfer more efficient and to minimize network traffic, implementations usually collect enough data from a stream to fill a reasonably large datagram before transmitting it across an internet. Thus, even if the application program generates the stream one octet at a time, transfer across an internet may be quite efficient. Similarly, if the application program chooses to generate extremely large blocks of data, the protocol software can choose to divide each block into smaller pieces for transmission.

For those applications where data should be delivered even though it does not fill a buffer, the stream service provides a *push* mechanism that applications use to force a transfer. At the sending side, a push forces protocol software to transfer all data that has been generated without waiting to fill a buffer. When it reaches the receiving side, the push causes TCP to make the data available to the application without delay. The reader should note, however, that the push function only guarantees that all data will be transferred; it does not provide record boundaries. Thus, even when delivery is forced, the protocol software may choose to divide the stream in unexpected ways.

- *Unstructured Stream.* It is important to understand that the TCP/IP stream service does not honor structured data streams. For example, there is no way for a payroll application to have the stream service mark boundaries between employee records, or to identify the contents of the stream as being payroll data. Application programs using the stream service must understand stream content and agree on stream format before they initiate a connection.
- *Full Duplex Connection.* Connections provided by the TCP/IP stream service allow concurrent transfer in both directions. Such connections are called *full duplex*. From the point of view of an application process, a full duplex connection consists of two independent streams flowing in opposite directions, with no apparent interaction. The stream service allows an application process to terminate flow in one direction while data continues to flow in the other direction, making the connection *half duplex*. The advantage of a full duplex connection is that the underlying protocol software can send control information for one stream back to the source in datagrams carrying data in the opposite direction. Such *piggybacking* reduces network traffic.

13.4 Providing Reliability

We have said that the reliable stream delivery service guarantees to deliver a stream of data sent from one machine to another without duplication or data loss. The question arises: "How can protocol software provide reliable transfer if the underlying communication system offers only unreliable packet delivery?" The answer is complicated, but most reliable protocols use a single fundamental technique known as *positive acknowledgement with retransmission*. The technique requires a recipient to communicate with the source, sending back an *acknowledgement* (*ACK*) message as it receives

data. The sender keeps a record of each packet it sends and waits for an acknowledgement before sending the next packet. The sender also starts a timer when it sends a packet and *retransmits* a packet if the timer expires before an acknowledgement arrives.

Figure 13.1 shows how the simplest positive acknowledgement protocol transfers data.

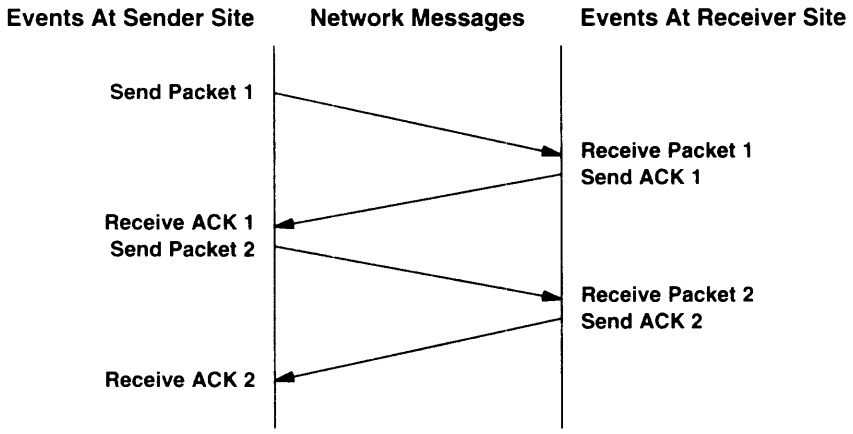


Figure 13.1 A protocol using positive acknowledgement with retransmission in which the sender awaits an acknowledgement for each packet sent. Vertical distance down the figure represents increasing time and diagonal lines across the middle represent network packet transmission.

In the figure, events at the sender and receiver are shown on the left and right. Each diagonal line crossing the middle shows the transfer of one message across the network.

Figure 13.2 uses the same format diagram as Figure 13.1 to show what happens when a packet is lost or corrupted. The sender starts a timer after transmitting a packet. When the timer expires, the sender assumes the packet was lost and retransmits it.

The final reliability problem arises when an underlying packet delivery system duplicates packets. Duplicates can also arise when networks experience high delays that cause premature retransmission. Solving duplication requires careful thought because both packets and acknowledgements can be duplicated. Usually, reliable protocols detect duplicate packets by assigning each packet a sequence number and requiring the receiver to remember which sequence numbers it has received. To avoid confusion caused by delayed or duplicated acknowledgements, positive acknowledgement protocols send sequence numbers back in acknowledgements, so the receiver can correctly associate acknowledgements with packets.

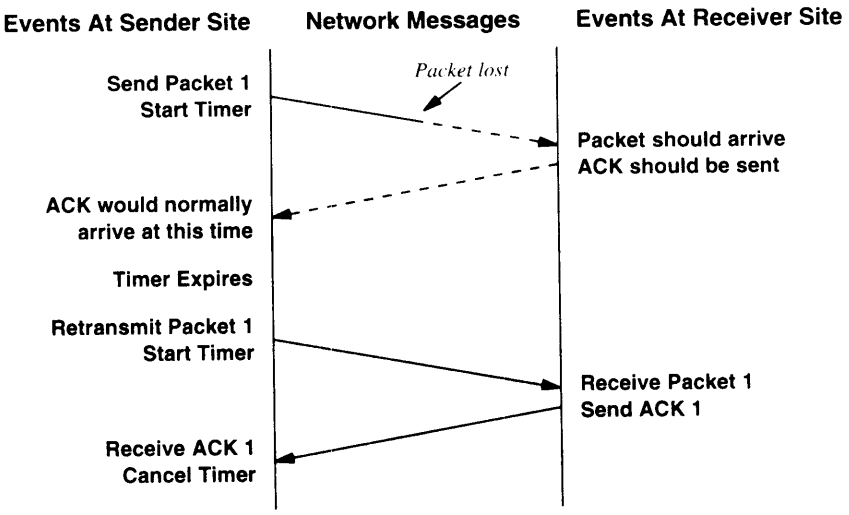


Figure 13.2 Timeout and retransmission that occurs when a packet is lost. The dotted lines show the time that would be taken by the transmission of a packet and its acknowledgement, if the packet was not lost.

13.5 The Idea Behind Sliding Windows

Before examining the TCP stream service, we need to explore an additional concept that underlies stream transmission. The concept, known as a *sliding window*, makes stream transmission efficient. To understand the motivation for sliding windows, recall the sequence of events that Figure 13.1 depicts. To achieve reliability, the sender transmits a packet and then waits for an acknowledgement before transmitting another. As Figure 13.1 shows, data only flows between the machines in one direction at any time, even if the network is capable of simultaneous communication in both directions. The network will be completely idle during times that machines delay responses (e.g., while machines compute routes or checksums). If we imagine a network with high transmission delays, the problem becomes clear:

> *A simple positive acknowledgement protocol wastes a substantial amount of network bandwidth because it must delay sending a new packet until it receives an acknowledgement for the previous packet.*

The sliding window technique is a more complex form of positive acknowledgement and retransmission than the simple method discussed above. Sliding window protocols use network bandwidth better because they allow the sender to transmit multiple packets before waiting for an acknowledgement. The easiest way to envision sliding

window operation is to think of a sequence of packets to be transmitted as Figure 13.3 shows. The protocol places a small, fixed-size *window* on the sequence and transmits all packets that lie inside the window.

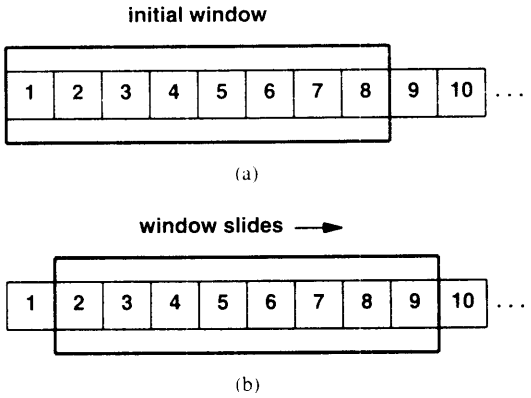


Figure 13.3 (a) A sliding window protocol with eight packets in the window, and (b) The window sliding so that packet 9 can be sent when an acknowledgement has been received for packet *1*. Only unacknowledged packets are retransmitted.

We say that a packet is *unacknowledged* if it has been transmitted but no acknowledgement has been received. Technically, the number of packets that can be unacknowledged at any given time is constrained by the *window size* and is limited to a small, fixed number. For example, in a sliding window protocol with window size *8*, the sender is permitted to transmit *8* packets before it receives an acknowledgement.

As Figure 13.3 shows, once the sender receives an acknowledgement for the first packet inside the window, it "slides" the window along and sends the next packet. The window continues to slide as long as acknowledgements are received.

The performance of sliding window protocols depends on the window size and the speed at which the network accepts packets. Figure 13.4 shows an example of the operation of a sliding window protocol when sending three packets. Note that the sender transmits all three packets before receiving any acknowledgements.

With a window size of *1*, a sliding window protocol is exactly the same as our simple positive acknowledgement protocol. By increasing the window size, it is possible to eliminate network idle time completely. That is, in the steady state, the sender can transmit packets as fast as the network can transfer them. The main point is:

> *Because a well tuned sliding window protocol keeps the network completely saturated with packets, it obtains substantially higher throughput than a simple positive acknowledgement protocol.*

Conceptually, a sliding window protocol always remembers which packets have been acknowledged and keeps a separate timer for each unacknowledged packet. If a packet is lost, the timer expires and the sender retransmits that packet. When the sender slides its window, it moves past all acknowledged packets. At the receiving end, the protocol software keeps an analogous window, accepting and acknowledging packets as they arrive. Thus, the window partitions the sequence of packets into three sets: those packets to the left of the window have been successfully transmitted, received, and acknowledged; those packets to the right have not yet been transmitted; and those packets that lie in the window are being transmitted. The lowest numbered packet in the window is the first packet in the sequence that has not been acknowledged.

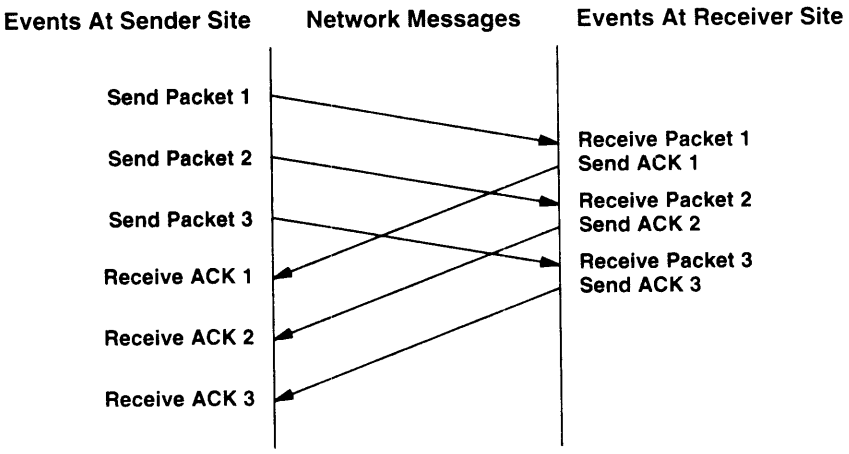


Figure 13.4 An example of three packets transmitted using a sliding window protocol. The key concept is that the sender can transmit all packets in the window without waiting for an acknowledgement.

13.6 The Transmission Control Protocol

Now that we understand the principle of sliding windows, we can examine the reliable stream service provided by the TCP/IP Internet protocol suite. The service is defined by the *Transmission Control Protocol*, or *TCP*. The reliable stream service is so important that the entire protocol suite is referred to as TCP/IP. It is important to understand that:

TCP is a communication protocol, not a piece of software.

The difference between a protocol and the software that implements it is analogous to the difference between the definition of a programming language and a compiler. As in the programming language world, the distinction between definition and implementa-

tion sometimes becomes blurred. People encounter TCP software much more frequently than they encounter the protocol specification, so it is natural to think of a particular implementation as the standard. Nevertheless, the reader should try to distinguish between the two.

Exactly what does TCP provide? TCP is complex, so there is no simple answer. The protocol specifies the format of the data and acknowledgements that two computers exchange to achieve a reliable transfer, as well as the procedures the computers use to ensure that the data arrives correctly. It specifies how TCP software distinguishes among multiple destinations on a given machine, and how communicating machines recover from errors like lost or duplicated packets. The protocol also specifies how two computers initiate a TCP stream transfer and how they agree when it is complete.

It is also important to understand what the protocol does not include. Although the TCP specification describes how application programs use TCP in general terms, it does not dictate the details of the interface between an application program and TCP. That is, the protocol documentation only discusses the operations TCP supplies; it does not specify the exact procedures application programs invoke to access these operations. The reason for leaving the application program interface unspecified is flexibility. In particular, because programmers usually implement TCP in the computer's operating system, they need to employ whatever interface the operating system supplies. Allowing the implementor flexibility makes it possible to have a single specification for TCP that can be used to build software for a variety of machines.

Because TCP assumes little about the underlying communication system, TCP can be used with a variety of packet delivery systems, including the IP datagram delivery service. For example, TCP can be implemented to use dialup telephone lines, a local area network, a high speed fiber optic network, or a lower speed long haul network. In fact, the large variety of delivery systems TCP can use is one of its strengths.

13.7 Ports, Connections, And Endpoints

Like the User Datagram Protocol (UDP) presented in Chapter 12, TCP resides above IP in the protocol layering scheme. Figure 13.5 shows the conceptual organization. TCP allows multiple application programs on a given machine to communicate concurrently, and it demultiplexes incoming TCP traffic among application programs. Like the User Datagram Protocol, TCP uses *protocol port* numbers to identify the ultimate destination within a machine. Each port is assigned a small integer used to identify it†.

†Although both TCP and UDP use integer port identifiers starting at *1* to identify ports, there is no confusion between them because an incoming IP datagram identifies the protocol being used as well as the port number.

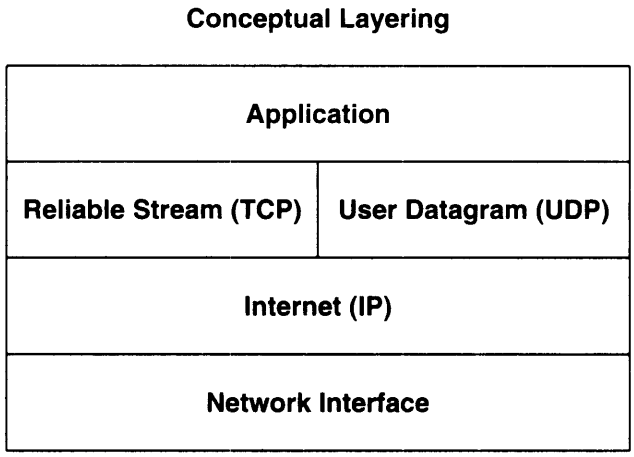


Figure 13.5 The conceptual layering of UDP and TCP above IP. TCP provides a reliable stream service, while UDP provides an unreliable datagram delivery service. Application programs use both.

When we discussed UDP ports, we said to think of each port as a queue into which protocol software places arriving datagrams. TCP ports are much more complex because a given port number does not correspond to a single object. Instead, TCP has been built on the *connection abstraction*, in which the objects to be identified are virtual circuit connections, not individual ports. Understanding that TCP uses the notion of connections is crucial because it helps explain the meaning and use of TCP port numbers:

> *TCP uses the connection, not the protocol port, as its fundamental abstraction; connections are identified by a pair of endpoints.*

Exactly what are the "endpoints" of a connection? We have said that a connection consists of a virtual circuit between two application programs, so it might be natural to assume that an application program serves as the connection "endpoint." It is not. Instead, TCP defines an *endpoint* to be a pair of integers (*host*, *port*), where *host* is the IP address for a host and *port* is a TCP port on that host. For example, the endpoint (*128.10.2.3*, *25*) specifies TCP port *25* on the machine with IP address *128.10.2.3*.

Now that we have defined endpoints, it will be easy to understand connections. Recall that a connection is defined by its two endpoints. Thus, if there is a connection from machine (*18.26.0.36*) at MIT to machine (*128.10.2.3*) at Purdue University, it might be defined by the endpoints:

(*18.26.0.36*, *1069*) and (*128.10.2.3*, *25*).

Meanwhile, another connection might be in progress from machine (*128.9.0.32*) at the Information Sciences Institute to the same machine at Purdue, identified by its endpoints:

(*128.9.0.32, 1184*) and (*128.10.2.3, 53*).

So far, our examples of connections have been straightforward because the ports used at all endpoints have been unique. However, the connection abstraction allows multiple connections to share an endpoint. For example, we could add another connection to the two listed above from machine (*128.2.254.139*) at CMU to the machine at Purdue:

(*128.2.254.139, 1184*) and (*128.10.2.3, 53*).

It might seem strange that two connections can use the TCP port *53* on machine 128.10.2.3 simultaneously, but there is no ambiguity. Because TCP associates incoming messages with a connection instead of a protocol port, it uses both endpoints to identify the appropriate connection. The important idea to remember is:

> *Because TCP identifies a connection by a pair of endpoints, a given TCP port number can be shared by multiple connections on the same machine.*

From a programmer's point of view, the connection abstraction is significant. It means a programmer can devise a program that provides concurrent service to multiple connections simultaneously without needing unique local port numbers for each connection. For example, most systems provide concurrent access to their electronic mail service, allowing multiple computers to send them electronic mail concurrently. Because the program that accepts incoming mail uses TCP to communicate, it only needs to use one local TCP port even though it allows multiple connections to proceed concurrently.

13.8 Passive And Active Opens

Unlike UDP, TCP is a connection-oriented protocol that requires both endpoints to agree to participate. That is, before TCP traffic can pass across an internet, application programs at both ends of the connection must agree that the connection is desired. To do so, the application program on one end performs a *passive open* function by contacting its operating system and indicating that it will accept an incoming connection. At that time, the operating system assigns a TCP port number for its end of the connection. The application program at the other end must then contact its operating system using an *active open* request to establish a connection. The two TCP software modules communicate to establish and verify a connection. Once a connection has been created, application programs can begin to pass data; the TCP software modules at each end exchange messages that guarantee reliable delivery. We will return to the details of establishing connections after examining the TCP message format.

13.9 Segments, Streams, And Sequence Numbers

TCP views the data stream as a sequence of octets or bytes that it divides into *segments* for transmission. Usually, each segment travels across an internet in a single IP datagram.

TCP uses a specialized sliding window mechanism to solve two important problems: efficient transmission and flow control. Like the sliding window protocol described earlier, the TCP window mechanism makes it possible to send multiple segments before an acknowledgement arrives. Doing so increases total throughput because it keeps the network busy. The TCP form of a sliding window protocol also solves the end-to-end *flow control* problem, by allowing the receiver to restrict transmission until it has sufficient buffer space to accommodate more data.

The TCP sliding window mechanism operates at the octet level, not at the segment or packet level. Octets of the data stream are numbered sequentially, and a sender keeps three pointers associated with every connection. The pointers define a sliding window as Figure 13.6 illustrates. The first pointer marks the left of the sliding window, separating octets that have been sent and acknowledged from octets yet to be acknowledged. A second pointer marks the right of the sliding window and defines the highest octet in the sequence that can be sent before more acknowledgements are received. The third pointer marks the boundary inside the window that separates those octets that have already been sent from those octets that have not been sent. The protocol software sends all octets in the window without delay, so the boundary inside the window usually moves from left to right quickly.

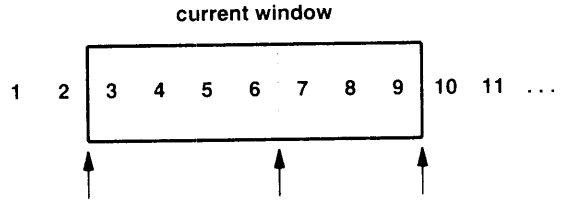


Figure 13.6 An example of the TCP sliding window. Octets through 2 have been sent and acknowledged, octets 3 through 6 have been sent but not acknowledged, octets 7 though 9 have not been sent but will be sent without delay, and octets 10 and higher cannot be sent until the window moves.

We have described how the sender's TCP window slides along and mentioned that the receiver must maintain a similar window to piece the stream together again. It is important to understand, however, that because TCP connections are full duplex, two transfers proceed simultaneously over each connection, one in each direction. We think of the transfers as completely independent because at any time data can flow across the connection in one direction, or in both directions. Thus, TCP software at each end

maintains two windows per connection (for a total of four), one slides along the data stream being sent, while the other slides along as data is received.

13.10 Variable Window Size And Flow Control

One difference between the TCP sliding window protocol and the simplified sliding window protocol presented earlier occurs because TCP allows the window size to vary over time. Each acknowledgement, which specifies how many octets have been received, contains a *window advertisement* that specifies how many additional octets of data the receiver is prepared to accept. We think of the window advertisement as specifying the receiver's current buffer size. In response to an increased window advertisement, the sender increases the size of its sliding window and proceeds to send octets that have not been acknowledged. In response to a decreased window advertisement, the sender decreases the size of its window and stops sending octets beyond the boundary. TCP software should not contradict previous advertisements by shrinking the window past previously acceptable positions in the octet stream. Instead, smaller advertisements accompany acknowledgements, so the window size changes at the time it slides forward.

The advantage of using a variable size window is that it provides flow control as well as reliable transfer. To avoid receiving more data than it can store, the receiver sends smaller window advertisements as its buffer fills. In the extreme case, the receiver advertises a window size of zero to stop all transmissions. Later, when buffer space becomes available, the receiver advertises a nonzero window size to trigger the flow of data again†.

Having a mechanism for flow control is essential in an internet environment, where machines of various speeds and sizes communicate through networks and routers of various speeds and capacities. There are two independent flow problems. First, internet protocols need end-to-end flow control between the source and ultimate destination. For example, when a minicomputer communicates with a large mainframe, the minicomputer needs to regulate the influx of data, or protocol software would be overrun quickly. Thus, TCP must implement end-to-end flow control to guarantee reliable delivery. Second, internet protocols need a flow control mechanism that allows intermediate systems (i.e., routers) to control a source that sends more traffic than the machine can tolerate.

When intermediate machines become overloaded, the condition is called *congestion*, and mechanisms to solve the problem are called *congestion control* mechanisms. TCP uses its sliding window scheme to solve the end-to-end flow control problem; it does not have an explicit mechanism for congestion control. We will see later, however, that a carefully programmed TCP implementation can detect and recover from congestion while a poor implementation can make it worse. In particular, although a carefully chosen retransmission scheme can help avoid congestion, a poorly chosen scheme can exacerbate it.

---

†There are two exceptions to transmission when the window size is zero. First, a sender is allowed to transmit a segment with the urgent bit set to inform the receiver that urgent data is available. Second, to avoid a potential deadlock that can arise if a nonzero advertisement is lost after the window size reaches zero, the sender probes a zero-sized window periodically.

13.11 TCP Segment Format

The unit of transfer between the TCP software on two machines is called a *segment*. Segments are exchanged to establish connections, transfer data, send acknowledgements, advertise window sizes, and close connections. Because TCP uses piggybacking, an acknowledgement traveling from machine *A* to machine *B* may travel in the same segment as data traveling from machine *A* to machine *B*, even though the acknowledgement refers to data sent from *B* to *A*†. Figure 13.7 shows the TCP segment format.

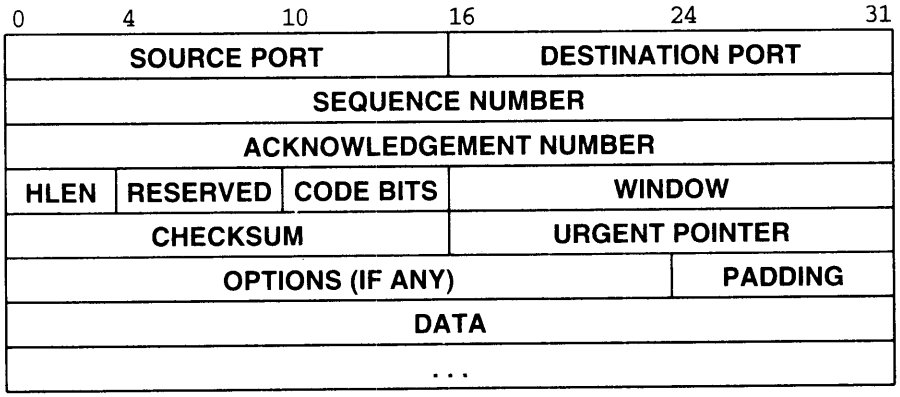


Figure 13.7 The format of a TCP segment with a TCP header followed by data. Segments are used to establish connections as well as to carry data and acknowledgements.

Each segment is divided into two parts, a header followed by data. The header, known as the *TCP header*, carries the expected identification and control information. Fields *SOURCE PORT* and *DESTINATION PORT* contain the TCP port numbers that identify the application programs at the ends of the connection. The *SEQUENCE NUMBER* field identifies the position in the sender's byte stream of the data in the segment. The *ACKNOWLEDGEMENT NUMBER* field identifies the number of the octet that the source expects to receive next. Note that the sequence number refers to the stream flowing in the same direction as the segment, while the acknowledgement number refers to the stream flowing in the opposite direction from the segment.

The *HLEN*‡ field contains an integer that specifies the length of the segment header measured in 32-bit multiples. It is needed because the *OPTIONS* field varies in length, depending on which options have been included. Thus, the size of the TCP header varies depending on the options selected. The 6-bit field marked *RESERVED* is reserved for future use.

---

†In practice, piggybacking does not usually occur because most applications do not send data in both directions simultaneously.

‡The specification says the *HLEN* field is the *offset* of the data area within the segment.

Some segments carry only an acknowledgement while some carry data. Others carry requests to establish or close a connection. TCP software uses the 6-bit field labeled *CODE BITS* to determine the purpose and contents of the segment. The six bits tell how to interpret other fields in the header according to the table in Figure 13.8.

Bit (left to right)	Meaning if bit set to 1
URG	Urgent pointer field is valid
ACK	Acknowledgement field is valid
PSH	This segment requests a push
RST	Reset the connection
SYN	Synchronize sequence numbers
FIN	Sender has reached end of its byte stream

Figure 13.8 Bits of the CODE field in the TCP header.

TCP software advertises how much data it is willing to accept every time it sends a segment by specifying its buffer size in the *WINDOW* field. The field contains a 16-bit unsigned integer in network-standard byte order. Window advertisements provide another example of piggybacking because they accompany all segments, including those carrying data as well as those carrying only an acknowledgement.

13.12 Out Of Band Data

Although TCP is a stream-oriented protocol, it is sometimes important for the program at one end of a connection to send data *out of band*, without waiting for the program at the other end of the connection to consume octets already in the stream. For example, when TCP is used for a remote login session, the user may decide to send a keyboard sequence that *interrupts* or *aborts* the program at the other end. Such signals are most often needed when a program on the remote machine fails to operate correctly. The signals must be sent without waiting for the program to read octets already in the TCP stream (or one would not be able to abort programs that stop reading input).

To accommodate out of band signaling, TCP allows the sender to specify data as *urgent*, meaning that the receiving program should be notified of its arrival as quickly as possible, regardless of its position in the stream. The protocol specifies that when urgent data is found, the receiving TCP should notify whatever application program is associated with the connection to go into "urgent mode." After all urgent data has been consumed, TCP tells the application program to return to normal operation.

The exact details of how TCP informs the application program about urgent data depend on the computer's operating system, of course. The mechanism used to mark urgent data when transmitting it in a segment consists of the URG code bit and the *URGENT POINTER* field. When the URG bit is set, the urgent pointer specifies the position in the segment where urgent data ends.

13.13 Maximum Segment Size Option

Not all segments sent across a connection will be of the same size. However, both ends need to agree on a maximum segment they will transfer. TCP software uses the *OPTIONS* field to negotiate with the TCP software at the other end of the connection; one of the options allows TCP software to specify the *maximum segment size* (*MSS*) that it is willing to receive. For example, when an embedded system that only has a few hundred bytes of buffer space connects to a large supercomputer, it can negotiate an MSS that restricts segments so they fit in the buffer. It is especially important for computers connected by high-speed local area networks to choose a maximum segment size that fills packets or they will not make good use of the bandwidth. Therefore, if the two endpoints lie on the same physical network, TCP usually computes a maximum segment size such that the resulting IP datagrams will match the network MTU. If the endpoints do not lie on the same physical network, they can attempt to discover the minimum MTU along the path between them, or choose a maximum segment size of *536* (the default size of an IP datagram, *576*, minus the standard size of IP and TCP headers).

In a general internet environment, choosing a good maximum segment size can be difficult because performance can be poor for either extremely large segment sizes or extremely small sizes. On one hand, when the segment size is small, network utilization remains low. To see why, recall that TCP segments travel encapsulated in IP datagrams which are encapsulated in physical network frames. Thus, each segment has at least 40 octets of TCP and IP headers in addition to the data. Therefore, datagrams carrying only one octet of data use at most 1/41 of the underlying network bandwidth for user data; in practice, minimum interpacket gaps and network hardware framing bits make the ratio even smaller.

On the other hand, extremely large segment sizes can also produce poor performance. Large segments result in large IP datagrams. When such datagrams travel across a network with small MTU, IP must fragment them. Unlike a TCP segment, a fragment cannot be acknowledged or retransmitted independently; all fragments must arrive or the entire datagram must be retransmitted. Because the probability of losing a given fragment is nonzero, increasing segment size above the fragmentation threshold decreases the probability the datagram will arrive, which decreases throughput.

In theory, the optimum segment size, *S*, occurs when the IP datagrams carrying the segments are as large as possible without requiring fragmentation anywhere along the path from the source to the destination. In practice, finding *S* is difficult for several reasons. First, most implementations of TCP do not include a mechanism for doing so†. Second, because routers in an internet can change routes dynamically, the path datagrams follow between a pair of communicating computers can change dynamically and so can the size at which datagrams must be fragmented. Third, the optimum size depends on lower-level protocol headers (e.g., the segment size must be reduced to accommodate IP options). Research on the problem of finding an optimal segment size continues.

---

†To discover the path MTU, a sender probes the path by sending datagrams with the IP *do not fragment* bit set. It then decreases the size if ICMP error messages report that fragmentation was required.

13.14 TCP Checksum Computation

The *CHECKSUM* field in the TCP header contains a 16-bit integer checksum used to verify the integrity of the data as well as the TCP header. To compute the checksum, TCP software on the sending machine follows a procedure like the one described in Chapter 12 for UDP. It prepends a *pseudo header* to the segment, appends enough zero bits to make the segment a multiple of 16 bits, and computes the 16-bit checksum over the entire result. TCP does not count the pseudo header or padding in the segment length, . or does it transmit them. Also, it assumes the checksum field itself is zero for purposes of the checksum computation. As with other checksums, TCP uses 16-bit arithmetic and takes the one's complement of the one's complement sum. At the receiving site, TCP software performs the same computation to verify that the segment arrived intact.

The purpose of using a pseudo header is exactly the same as in UDP. It allows the receiver to verify that the segment has reached its correct destination, which includes both a host IP address as well as a protocol port number. Both the source and destination IP addresses are important to TCP because it must use them to identify a connection to which the segment belongs. Therefore, whenever a datagram arrives carrying a TCP segment, IP must pass to TCP the source and destination IP addresses from the datagram as well as the segment itself. Figure 13.9 shows the format of the pseudo header used in the checksum computation.

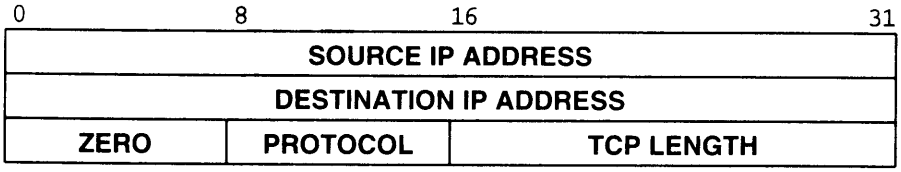


Figure 13.9 The format of the pseudo header used in TCP checksum computations. At the receiving site, this information is extracted from the IP datagram that carried the segment.

The sending TCP assigns field *PROTOCOL* the value that the underlying delivery system will use in its protocol type field. For IP datagrams carrying TCP, the value is 6. The *TCP LENGTH* field specifies the total length of the TCP segment including the TCP header. At the receiving end, information used in the pseudo header is extracted from the IP datagram that carried the segment and included in the checksum computation to verify that the segment arrived at the correct destination intact.