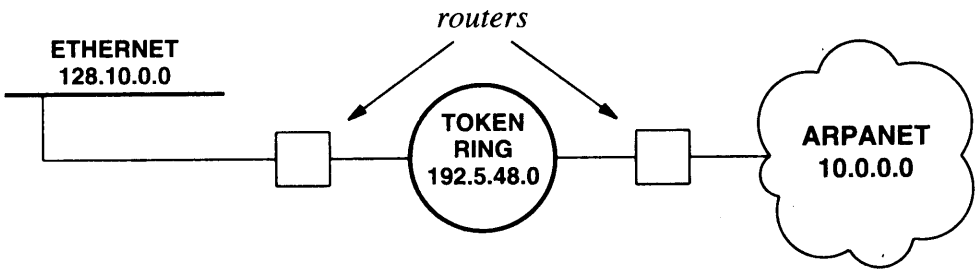


Figure 4.5 The logical connection of two networks to the Internet backbone. Each network has been assigned an IP address.

The example shows three networks and the network numbers they have been assigned: the ARPANET (10.0.0.0), an Ethernet (128.10.0.0), and a token ring network (192.5.48.0). According to the table in Figure 4.3, the addresses have classes *A*, *B*, and *C*, respectively.

Figure 4.6 shows the same networks with host computers attached and Internet addresses assigned to each network connection.

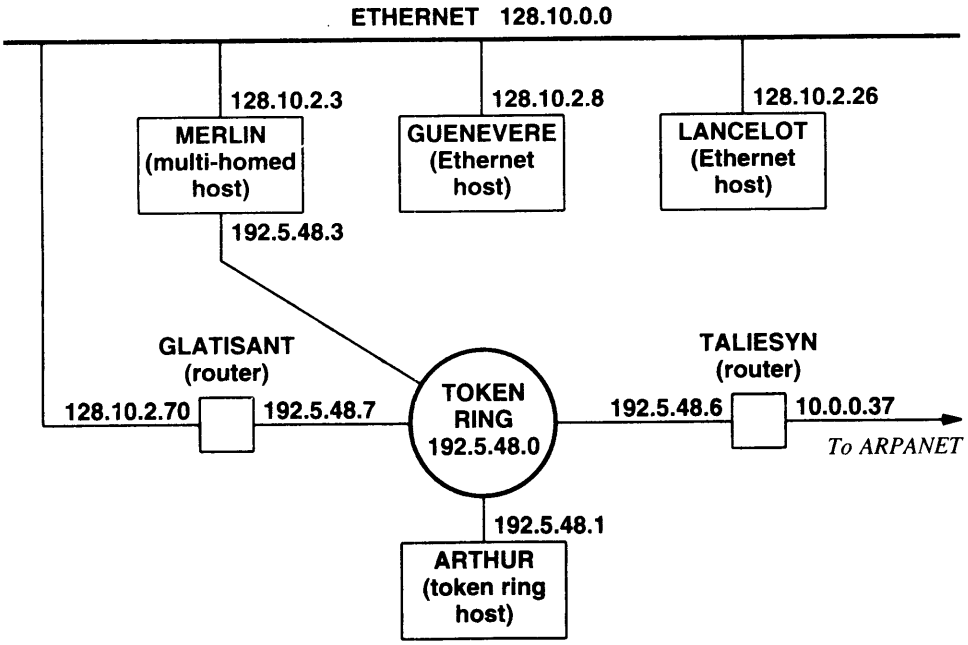


Figure 4.6 Example IP address assignment for routers and hosts attached to the three networks in the previous figure.

In the figure, four hosts labeled *Arthur*, *Merlin*, *Guenevere*, and *Lancelot*, attach to the networks, *Taliesyn* is a router that connects the ARPANET and the token ring network, and *Glatisant* is a router that connects the token ring network to the Ethernet. Host *Merlin* has connections to both the Ethernet and the token ring network, so it can reach destinations on either network directly. Although a multi-homed host like *Merlin* can be configured to route packets between the two nets, most sites use dedicated computers as routers to avoid overloading conventional computer systems with the processing required for routing. In the figure, a dedicated router, *Glatisant*, performs the task of routing traffic between the Ethernet and token ring networks. (Note: actual traffic between these two networks was higher than this configuration suggests because the figure only shows a few of the computers attached to the nets.)

As Figure 4.5 shows, an IP address must be assigned to each network connection. *Lancelot*, which connects only to the Ethernet, has been assigned 128.10.2.26 as its only IP address. *Merlin* has address 128.10.2.3 for its connection to the Ethernet and 192.5.48.3 for its connection to the token ring network. Whoever made the address assignment chose the same value for the low-order byte of each address. The addresses assigned to routers *Glatisant* and *Taliesyn* do not follow the convention. For example, *Taliesyn's* addresses, 10.0.0.37 and 192.5.48.6, are two completely unrelated strings of digits. IP does not care whether any of the bytes in the dotted decimal form of a computer's addresses are the same or different. However, network technicians, managers, and administrators may need to use addresses for maintenance, testing, and debugging. Choosing to make all of a computer's addresses end with the same octet makes it easier for humans to remember or guess the address of a particular interface.

4.17 Network Byte Order

To create an internet that is independent of any particular vendor's machine architecture or network hardware, the software must define a standard representation for data. Consider what happens, for example, when software on one computer sends a 32-bit binary integer to another computer. The physical transport hardware moves the sequence of bits from the first machine to the second without changing the order. However, not all architectures store 32-bit integers in the same way. On some (called *Little Endian*), the lowest memory address contains the low-order byte of the integer. On others (called *Big Endian*), the lowest memory address holds the high-order byte of the integer. Still others store integers in groups of 16-bit words, with the lowest addresses holding the low-order word, but with bytes swapped. Thus, direct copying of bytes from one machine to another may change the value of the number.

Standardizing byte-order for integers is especially important in an internet because internet packets carry binary numbers that specify information like destination addresses and packet lengths. Such quantities must be understood by both the senders and receivers. The TCP/IP protocols solve the byte-order problem by defining a *network standard byte order* that all machines must use for binary fields in internet packets. Each host or router converts binary items from the local representation to network standard byte order before sending a packet, and converts from network byte order to the host-specific order when a packet arrives. Naturally, the user data field in a packet is

exempt from this standard because the TCP/IP protocols do not know what data is being carried — application programmers are free to format their own data representation and translation. When sending integer values, many application programmers do choose to follow the TCP/IP byte-order standards. Of course, users who merely invoke application programs never need to deal with the byte order problem directly.

The internet standard for byte order specifies that integers are sent with the most significant byte first (i.e., *Big Endian* style). If one considers the successive bytes in a packet as it travels from one machine to another, a binary integer in that packet has its most significant byte nearest the beginning of the packet and its least significant byte nearest the end of the packet. Many arguments have been offered about which data representation should be used, and the internet standard still comes under attack from time to time. In particular, proponents of change argue that although most computers were big endian when the standard was defined, most are now little endian. However, everyone agrees that having a standard is crucial, and the exact form of the standard is far less important.

4.18 Summary

TCP/IP uses 32-bit binary addresses as universal machine identifiers. Called Internet Protocol addresses or IP addresses, the identifiers are partitioned into two parts: a prefix identifies the network to which the computer attaches and the suffix provides a unique identifier for the computer on that network. The original IP addressing scheme is known as classful, with each prefix assigned to one of three primary classes. Leading bits define the class of an address; the classes are of unequal size. The classful scheme provides for 127 networks with over a million hosts each, thousands of networks with thousands of hosts each, and over a million networks with up to 254 hosts each. To make such addresses easier for humans to understand, they are written in dotted decimal notation, with the values of the four octets written in decimal, separated by decimal points.

Because the IP address encodes network identification as well as the identification of a specific host on that network, routing is efficient. An important property of IP addresses is that they refer to network connections. Hosts with multiple connections have multiple addresses. One advantage of the internet addressing scheme is that the form includes an address for a specific host, a network, or all hosts on a network (broadcast). The biggest disadvantage of the IP addressing scheme is that if a machine has multiple addresses, knowing one address may not be sufficient to reach it when no path exists to the specified interface (e.g., because a particular network is unavailable).

To permit the exchange of binary data among machines, TCP/IP protocols enforce a standard byte ordering for integers within protocol fields. A host must convert all binary data from its internal form to network standard byte order before sending a packet, and it must convert from network byte order to internal order upon receipt.

FOR FURTHER STUDY

The internet addressing scheme presented here can be found in Reynolds and Postel [RFC 1700]; further information can be found in Stahl, Romano, and Recker [RFC 1117].

Several important additions have been made to the Internet addressing scheme over the years; later chapters cover them in more detail. Chapter 10 discusses an important extension called *classless addressing* that permits the division between prefix and suffix to occur at an arbitrary bit position. In addition, Chapter 10 examines an essential part of the Internet address standard called *subnet addressing*. Subnet addressing allows a single network address to be used with multiple physical networks. Chapter 17 continues the exploration of IP addresses by describing how class D addresses are assigned for internet *multicast*.

Cohen [1981] explains bit and byte ordering, and introduces the terms "Big Endian" and "Little Endian."

EXERCISES

4.1 Exactly how many class *A*, *B*, and *C* networks can exist? Exactly how many hosts can a network in each class have? Be careful to allow for broadcast as well as class *D* and *E* addresses.

4.2 A machine readable list of assigned addresses is sometimes called an internet *host table*. If your site has a host table, find out how many class *A*, *B*, and *C* network numbers have been assigned.

4.3 How many hosts are attached to each of the local area networks at your site? Does your site have any local area networks for which a class *C* address is insufficient?

4.4 What is the chief difference between the IP addressing scheme and the U.S. telephone numbering scheme?

4.5 A single central authority cannot manage to assign Internet addresses fast enough to accommodate the demand. Can you invent a scheme that allows the central authority to divide its task among several groups but still ensure that each assigned address is unique?

4.6 Does network standard byte order differ from your local machine's byte order?

4.7 How many IP addresses would be needed to assign a unique IP address to every house in your country? the world? Is the IP address space sufficient?

5

Mapping Internet Addresses To Physical Addresses (ARP)

5.1 Introduction

We described the TCP/IP address scheme in which each host is assigned a 32-bit address, and said that an internet behaves like a virtual network, using only the assigned addresses when sending and receiving packets. We also reviewed several network hardware technologies, and noted that two machines on a given physical network can communicate *only if they know each other's physical network address*. What we have not mentioned is how a host or a router maps an IP address to the correct physical address when it needs to send a packet across a physical net. This chapter considers that mapping, showing how it is implemented for the two most common physical network address schemes.

5.2 The Address Resolution Problem

Consider two machines A and B that connect to the same physical network. Each has an assigned IP address I_A and I_B and a physical address P_A and P_B. The goal is to devise low-level software that hides physical addresses and allows higher-level programs to work only with internet addresses. Ultimately, however, communication must be carried out by physical networks using whatever physical address scheme the underlying network hardware supplies. Suppose machine A wants to send a packet to

machine *B* across a physical network to which they both attach, but *A* has only *B*'s internet address I_B. The question arises: how does *A* map that address to *B's* physical address, P_B?

Address mapping must be performed at each step along a path from the original source to the ultimate destination. In particular, two cases arise. First, at the last step of delivering a packet, the packet must be sent across one physical network to its final destination. The computer sending the packet must map the final destination's Internet address to the destination's physical address. Second, at any point along the path from the source to the destination other than the final step, the packet must be sent to an intermediate router. Thus, the sender must map the intermediate router's Internet address to a physical address.

The problem of mapping high-level addresses to physical addresses is known as the *address resolution problem* and has been solved in several ways. Some protocol suites keep tables in each machine that contain pairs of high-level and physical addresses. Others solve the problem by encoding hardware addresses in high-level addresses. Using either approach exclusively makes high-level addressing awkward at best. This chapter discusses two techniques for address resolution used by TCP/IP protocols and shows when each is appropriate.

5.3 Two Types Of Physical Addresses

There are two basic types of physical addresses, exemplified by the Ethernet, which has large, fixed physical addresses, and proNET, which has small, easily configured physical addresses. Address resolution is difficult for Ethernet-like networks, but easy for networks like proNET. We will consider the easy case first.

5.4 Resolution Through Direct Mapping

Consider a proNET token ring network. Recall from Chapter 2 that proNET uses small integers for physical addresses and allows the user to choose a hardware address when installing an interface board in a computer. The key to making address resolution easy with such network hardware lies in observing that as long as one has the freedom to choose both IP and physical addresses, they can be selected such that parts of them are the same. Typically, one assigns IP addresses with the hostid portion equal to 1, 2, 3, and so on, and then, when installing network interface hardware, selects a physical address that corresponds to the IP address. For example, the system administrator would select physical address 3 for a computer with the IP address 192.5.48.3 because 192.5.48.3 is a class *C* address with the host portion equal to 3.

For networks like proNET, computing a physical address from an IP address is trivial. The computation consists of extracting the host portion of the IP address. Extraction is computationally efficient on most architectures because it requires only a few machine instructions. The mapping is easy to maintain because it can be performed

without reference to external data. Finally, new computers can be added to the network without changing existing assignments or recompiling code.

Conceptually, choosing a numbering scheme that makes address resolution efficient means selecting a function f that maps IP addresses to physical addresses. The designer may be able to select a physical address numbering scheme as well, depending on the hardware. Resolving IP address I_A means computing

$$P_A = f(I_A)$$

We want the computation of f to be efficient. If the set of physical addresses is constrained, it may be possible to arrange efficient mappings other than the one given in the example above. For instance, when using IP over a connection-oriented network such as ATM, one cannot choose physical addresses. On such networks, one or more computers (servers) store pairs of addresses, where each pair contains an Internet address and the corresponding physical address. Typically, such servers store the pairs in a table in memory to speed searching. To guarantee efficient address resolution in such cases, software can use a conventional hash function to search the table. Exercise 5.1 suggests a related alternative.

5.5 Resolution Through Dynamic Binding

To understand why address resolution is difficult for some networks, consider Ethernet technology. Recall from Chapter 2 that each Ethernet interface is assigned a 48-bit physical address when the device is manufactured. As a consequence, when hardware fails and requires that an Ethernet interface be replaced, the machine's physical address changes. Furthermore, because the Ethernet address is 48 bits long, there is no hope it can be encoded in a 32-bit IP address†.

Designers of TCP/IP protocols found a creative solution to the address resolution problem for networks like the Ethernet that have broadcast capability. The solution allows new hosts or routers to be added to the network without recompiling code, and does not require maintenance of a centralized database. To avoid maintaining a table of mappings, the designers chose to use a low-level protocol to bind addresses dynamically. Termed the *Address Resolution Protocol* (*ARP*), the protocol provides a mechanism that is both reasonably efficient and easy to maintain.

As Figure 5.1 shows, the idea behind dynamic resolution with ARP is simple: when host A wants to resolve IP address I_B, it broadcasts a special packet that asks the host with IP address I_B to respond with its physical address, P_B. All hosts, including B, receive the request, but only host B recognizes its IP address and sends a reply that contains its physical address. When A receives the reply, it uses the physical address to send the internet packet directly to B. We can summarize:

†Because direct mapping is more convenient and efficient than dynamic binding, the next generation of IP is being designed to allow 48-bit hardware addresses to be encoded in IP addresses.

The Address Resolution Protocol, ARP, allows a host to find the physical address of a target host on the same physical network, given only the target's IP address.

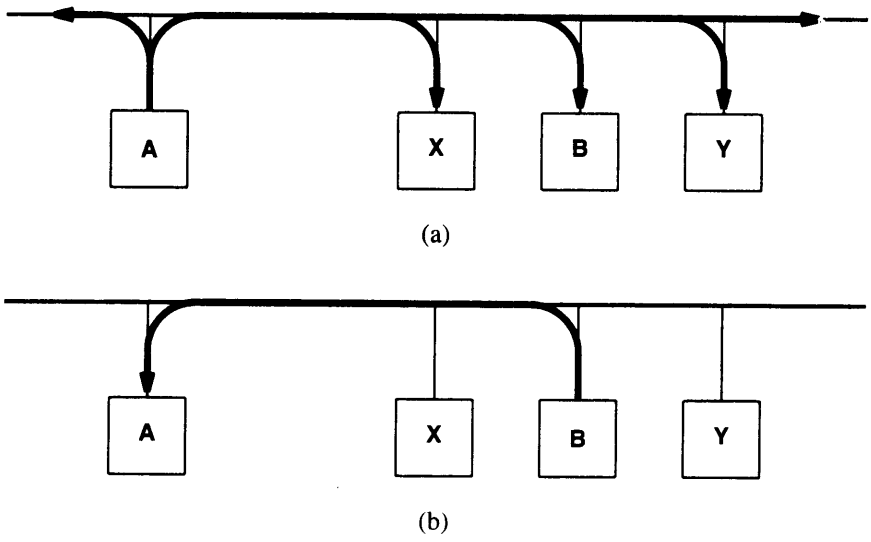


Figure 5.1 The ARP protocol. To determine P_B, B's physical address, from I_B, its IP address, (a) host A broadcasts an ARP request containing I_B to all machines on the net, and (b) host B responds with an ARP reply that contains the pair (I_B, P_B).

5.6 The Address Resolution Cache

It may seem silly that for A to send a packet to B it first sends a broadcast that reaches B. Or it may seem even sillier that A broadcasts the question, "how can I reach you?" instead of just broadcasting the packet it wants to deliver. But there is an important reason for the exchange. Broadcasting is far too expensive to be used every time one machine needs to transmit a packet to another because every machine on the network must receive and process the broadcast packet.

5.7 ARP Cache Timeout

To reduce communication costs, computers that use ARP maintain a cache of recently acquired IP-to-physical address bindings. That is, whenever a computer sends an ARP request and receives an ARP reply, it saves the IP address and corresponding hardware address information in its cache for successive lookups. When transmitting a packet, a computer always looks in its cache for a binding before sending an ARP request. If it finds the desired binding in its ARP cache, the computer need not broadcast on the network. Thus, when two computers on a network communicate, they begin with an ARP request and response, and then repeatedly transfer packets without using ARP for each one. Experience shows that because most network communication involves more than one packet transfer, even a small cache is worthwhile.

The ARP cache provides an example of *soft state*, a technique commonly used in network protocols. The name describes a situation in which information can become "stale" without warning. In the case of ARP, consider two computers, *A* and *B*, both connected to an Ethernet. Assume *A* has sent an ARP request, and *B* has replied. Further assume that after the exchange *B* crashes. Computer *A* will not receive any notification of the crash. Moreover, because it already has address binding information for *B* in its ARP cache, computer *A* will continue to send packets to *B*. The Ethernet hardware provides no indication that *B* is not on-line because Ethernet does not have guaranteed delivery. Thus, *A* has no way of knowing when information in its ARP cache has become incorrect.

To accommodate soft state, responsibility for correctness lies with the owner of the information. Typically, protocols that implement soft state use timers, with the state information being deleted when the timer expires. For example, whenever address binding information is placed in an ARP cache, the protocol requires a timer to be set, with a typical timeout being 20 minutes. When the timer expires, the information must be removed. After removal there are two possibilities. If no further packets are sent to the destination, nothing occurs. If a packet must be sent to the destination and there is no binding present in the cache, the computer follows the normal procedure of broadcasting an ARP request and obtaining the binding. If the destination is still reachable, the binding will again be placed in the ARP cache. If not, the sender will discover that the destination is off-line.

The use of soft state in ARP has advantages and disadvantages. The chief advantage arises from autonomy. First, a computer can determine when information in its ARP cache should be revalidated independent of other computers. Second, a sender does not need successful communication with the receiver or a third party to determine that a binding has become invalid; if a target does not respond to an ARP request, the sender will declare the target to be down. Third, the scheme does not rely on network hardware to provide reliable transfer. The chief disadvantage of soft state arises from delay — if the timer interval is *N* seconds, a sender may not detect that a receiver has crashed until *N* seconds elapse.

5.8 ARP Refinements

Several refinements of ARP have been included in the protocol. First, observe that if host *A* is about to use ARP because it needs to send to *B*, there is a high probability that host *B* will need to send to *A* in the near future. To anticipate *B*'s need and avoid extra network traffic, *A* includes its IP-to-physical address binding when sending *B* a request. *B* extracts *A*'s binding from the request, saves the binding in its ARP cache, and then sends a reply to *A*. Second, notice that because *A* broadcasts its initial request, all machines on the network receive it and can extract and update *A*'s IP-to-physical address binding in their cache. Third, when a computer has its host interface replaced, (e.g., because the hardware has failed) its physical address changes. Other computers on the net that have stored a binding in their ARP cache need to be informed so they can change the entry. The computer can notify others of a new address by sending an ARP broadcast when it boots.

The following rule summarizes refinements:

> *The sender's IP-to-physical address binding is included in every ARP broadcast; receivers update the IP-to-physical address binding information in their cache before processing an ARP packet.*

5.9 Relationship Of ARP To Other Protocols

ARP provides one possible mechanism to map from IP addresses to physical addresses; we have already seen that some network technologies do not need it. The point is that ARP would be completely unnecessary if we could make all network hardware recognize IP addresses. Thus, ARP merely imposes a new address scheme on top of whatever low-level address mechanism the hardware uses. The idea can be summarized:

> *ARP is a low-level protocol that hides the underlying network physical addressing, permitting one to assign an arbitrary IP address to every machine. We think of ARP as part of the physical network system, and not as part of the internet protocols.*

5.10 ARP Implementation

Functionally, ARP is divided into two parts. The first part maps an IP address to a physical address when sending a packet, and the second part answers requests from other machines. Address resolution for outgoing packets seems straightforward, but small details complicate an implementation. Given a destination IP address the software consults its ARP cache to see if it knows the mapping from IP address to physical address.

If it does, the software extracts the physical address, places the data in a frame using that address, and sends the frame. If it does not know the mapping, the software must broadcast an ARP request and wait for a reply.

Broadcasting an ARP request to find an address mapping can become complex. The target machine can be down or just too busy to accept the request. If so, the sender may not receive a reply or the reply may be delayed. Because the Ethernet is a best-effort delivery system, the initial ARP broadcast request can also be lost (in which case the sender should retransmit, at least once). Meanwhile, the host must store the original outgoing packet so it can be sent once the address has been resolved†. In fact, the host must decide whether to allow other application programs to proceed while it processes an ARP request (most do). If so, the software must handle the case where an application generates additional ARP requests for the same address without broadcasting multiple requests for a given target.

Finally, consider the case where machine *A* has obtained a binding for machine *B*, but then *B*'s hardware fails and is replaced. Although *B*'s address has changed, *A*'s cached binding has not, so *A* uses a nonexistent hardware address, making successful reception impossible. This case shows why it is important to have ARP software treat its table of bindings as a cache and remove entries after a fixed period. Of course, the timer for an entry in the cache must be reset whenever an ARP broadcast arrives containing the binding (but it is not reset when the entry is used to send a packet).

The second part of the ARP code handles ARP packets that arrive from the network. When an ARP packet arrives, the software first extracts the sender's IP address and hardware address pair, and examines the local cache to see if it already has an entry for the sender. If a cache entry exists for the given IP address, the handler updates that entry by overwriting the physical address with the physical address obtained from the packet. The receiver then processes the rest of the ARP packet.

A receiver must handle two types of incoming ARP packets. If an ARP request arrives, the receiving machine must see if it is the target of the request (i.e., some other machine has broadcast a request for the receiver's physical address). If so, the ARP software forms a reply by supplying its physical hardware address, and sends the reply directly back to the requester. The receiver also adds the sender's address pair to its cache if the pair is not already present. If the IP address mentioned in the ARP request does not match the local IP address, the packet is requesting a mapping for some other machine on the network and can be ignored.

The other interesting case occurs when an ARP reply arrives. Depending on the implementation, the handler may need to create a cache entry, or the entry may have been created when the request was generated. In any case, once the cache has been updated, the receiver tries to match the reply with a previously issued request. Usually, replies arrive in response to a request, which was generated because the machine has a packet to deliver. Between the time a machine broadcasts its ARP request and receives the reply, application programs or higher-level protocols may generate additional requests for the same address; the software must remember that it has already sent a request and not send more. Usually, ARP software places the additional packets on a queue. Once the reply arrives and the address binding is known, the ARP software re-

†If the delay is significant, the host may choose to discard the outgoing packet(s).

moves packets from the queue, places each packet in a frame, and uses the address binding to fill in the physical destination address. If it did not previously issue a request for the IP address in the reply, the machine updates the sender's entry in its cache, and then simply stops processing the packet.

5.11 ARP Encapsulation And Identification

When ARP messages travel from one machine to another, they must be carried in physical frames. Figure 5.2 shows that the ARP message is carried in the data portion of a frame.

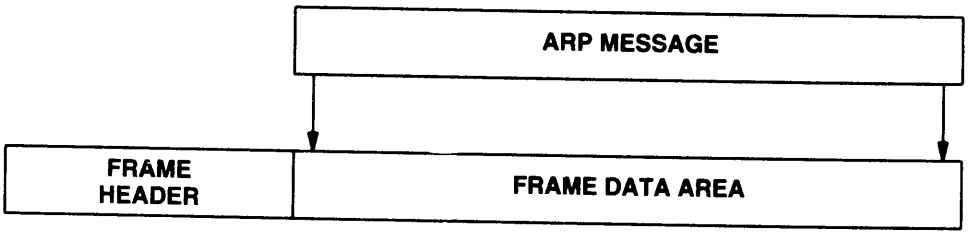


Figure 5.2 An ARP message encapsulated in a physical network frame.

To identify the frame as carrying an ARP message, the sender assigns a special value to the type field in the frame header, and places the ARP message in the frame's data field. When a frame arrives at a computer, the network software uses the frame type to determine its contents. In most technologies, a single type value is used for all frames that carry an ARP message — network software in the receiver must further examine the ARP message to distinguish between ARP requests and ARP replies. For example, on an Ethernet, frames carrying ARP messages have a type field of 0806_{16}. This is a standard value assigned by the authority for Ethernet; other network hardware technologies use other values.

5.12 ARP Protocol Format

Unlike most protocols, the data in ARP packets does not have a fixed-format header. Instead, to make ARP useful for a variety of network technologies, the length of fields that contain addresses depend on the type of network. However, to make it possible to interpret an arbitrary ARP message, the header includes fixed fields near the beginning that specify the lengths of the addresses found in succeeding fields. In fact, the ARP message format is general enough to allow it to be used with arbitrary physical addresses and arbitrary protocol addresses. The example in Figure 5.3 shows the 28-octet ARP message format used on Ethernet hardware (where physical addresses are

48-bits or 6 octets long), when resolving IP protocol addresses (which are 4 octets long).

Figure 5.3 shows an ARP message with 4 octets per line, a format that is standard throughout this text. Unfortunately, unlike most of the remaining protocols, the variable-length fields in ARP packets do not align neatly on 32-bit boundaries, making the diagram difficult to read. For example, the sender's hardware address, labeled *SENDER HA*, occupies 6 contiguous octets, so it spans two lines in the diagram.

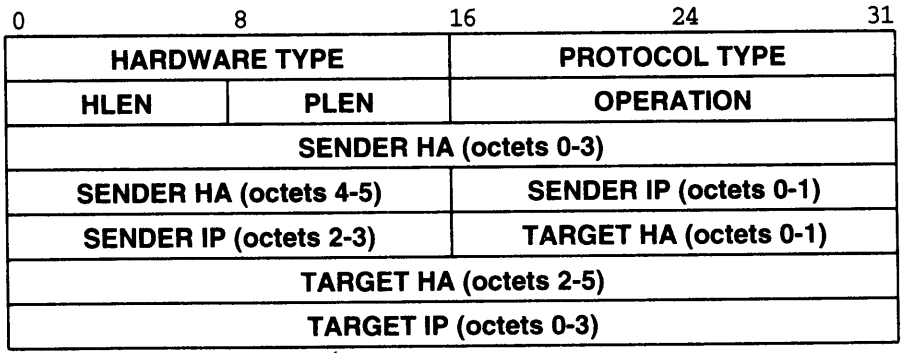


Figure 5.3 An example of the ARP/RARP message format when used for IP-to-Ethernet address resolution. The length of fields depends on the hardware and protocol address lengths, which are 6 octets for an Ethernet address and 4 octets for an IP address.

Field *HARDWARE TYPE* specifies a hardware interface type for which the sender seeks an answer; it contains the value *1* for Ethernet. Similarly, field *PROTOCOL TYPE* specifies the type of high-level protocol address the sender has supplied; it contains 0800_{16} for IP addresses. Field *OPERATION* specifies an ARP request (*1*), ARP response (*2*), RARP† request (*3*), or RARP response (*4*). Fields *HLEN* and *PLEN* allow ARP to be used with arbitrary networks because they specify the length of the hardware address and the length of the high-level protocol address. The sender supplies its hardware address and IP address, if known, in fields *SENDER HA* and *SENDER IP*.

When making a request, the sender also supplies the target hardware address (RARP) or target IP address (ARP), using fields *TARGET HA* or *TARGET IP*. Before the target machine responds, it fills in the missing addresses, swaps the target and sender pairs, and changes the operation to a reply. Thus, a reply carries the IP and hardware addresses of the original requester, as well as the IP and hardware addresses of the machine for which a binding was sought.

†The next chapter describes RARP, another protocol that uses the same message format.

5.13 Summary

IP addresses are assigned independent of a machine's physical hardware address. To send an internet packet across a physical net from one computer to another, the network software must map the IP address into a physical hardware address and use the hardware address to transmit the frame. If hardware addresses are smaller than IP addresses, a direct mapping can be established by having the machine's physical address encoded in its IP address. Otherwise, the mapping must be performed dynamically. The Address Resolution Protocol (ARP) performs dynamic address resolution, using only the low-level network communication system. ARP permits machines to resolve addresses without keeping a permanent record of bindings.

A machine uses ARP to find the hardware address of another machine by broadcasting an ARP request. The request contains the IP address of the machine for which a hardware address is needed. All machines on a network receive an ARP request. If the request matches a machine's IP address, the machine responds by sending a reply that contains the needed hardware address. Replies are directed to one machine; they are not broadcast.

To make ARP efficient, each machine caches IP-to-physical address bindings. Because internet traffic tends to consist of a sequence of interactions between pairs of machines, the cache eliminates most ARP broadcast requests.

FOR FURTHER STUDY

The address resolution protocol used here is given by Plummer [RFC 826] and has become a TCP/IP internet protocol standard. Dalal and Printis [1981] describes the relationship between Ethernet and IP addresses, and Clark [RFC 814] discusses addresses and bindings in general. Parr [RFC 1029] discusses fault tolerant address resolution. Kirkpatrick and Recker [RFC 1166] specifies values used to identify network frames in the Internet Numbers document. Volume 2 of this text presents an example ARP implementation, and discusses the caching policy.

EXERCISES

5.1 Given a small set of physical addresses (positive integers), can you find a function f and an assignment of IP addresses such that f maps the IP addresses 1-to-1 onto the physical addresses and computing f is efficient? (Hint: look at the literature on perfect hashing).

5.2 In what special cases does a host connected to an Ethernet not need to use ARP or an ARP cache before transmitting an IP datagram?

5.3 One common algorithm for managing the ARP cache replaces the least recently used entry when adding a new one. Under what circumstances can this algorithm produce unnecessary network traffic?

5.4 Read the standard carefully. Should ARP update the cache if an old entry already exists for a given IP address? Why or why not?

5.5 Should ARP software modify the cache even when it receives information without specifically requesting it? Why or why not?

5.6 Any implementation of ARP that uses a fixed-size cache can fail when used on a network that has many hosts and much ARP traffic. Explain how.

5.7 ARP is often cited as a security weakness. Explain why.

5.8 Suppose an (incorrect) ARP implementation does not remove cache entries if they are frequently used. Explain what can happen if the hardware address field in an ARP response becomes corrupted during transmission.

5.9 Suppose machine C receives an ARP request sent from A looking for target B, and suppose C has the binding from I_B to P_B in its cache. Should C answer the request? Explain.

5.10 How can a workstation use ARP when it boots to find out if any other machine on the network is impersonating it? What are the disadvantages of the scheme?

5.11 Explain how sending IP packets to nonexistent addresses on a remote Ethernet can generate excess broadcast traffic on that network.

6

Determining An Internet Address At Startup (RARP)

6.1 Introduction

We now know that physical network addresses are both low-level and hardware dependent, and we understand that each machine using TCP/IP is assigned one or more 32-bit IP addresses that are independent of the machine's hardware addresses. Application programs always use the IP address when specifying a destination. Because hosts and routers must use a physical address to transmit a datagram across an underlying hardware network; they rely on address resolution schemes like ARP to map between an IP address and an equivalent hardware address.

Usually, a computer's IP address is kept on its secondary storage, where the operating system finds it at startup. The question arises, "How does a machine without a permanently attached disk determine its IP address?" The problem is critical for workstations that store files on a remote server or for small embedded systems because such machines need an IP address before they can use standard TCP/IP file transfer protocols to obtain their initial boot image. This chapter explores the question of how to obtain an IP address, and describes a low-level protocol that such machines can use before they boot from a remote file server. Chapter 23 extends the discussion of bootstrapping, and considers popular alternatives to the protocol presented here.

Because an operating system image that has a specific IP address bound into the code cannot be used on multiple computers, designers usually try to avoid compiling a machine's IP address in the operating system code or support software. In particular, the bootstrap code often found in Read Only Memory (ROM) is usually built so the same image can run on many machines. When such code starts execution, it uses the network to contact a server and obtain the computer's IP address.

The bootstrap procedure sounds paradoxical: a machine communicates with a remote server to obtain an address needed for communication. The paradox is only imagined, however, because the machine *does* know how to communicate. It can use its physical address to communicate over a single network. Thus, the machine must resort to physical network addressing temporarily in the same way that operating systems use physical memory addressing to set up page tables for virtual addressing. Once a machine knows its IP address, it can communicate across an internet.

The idea behind finding an IP address is simple: a machine that needs to know its address sends a request to a *server*† on another machine, and waits until the server sends a response. We assume the server has access to a disk where it keeps a database of internet addresses. In the request, the machine that needs to know its internet address must uniquely identify itself, so the server can look up the correct internet address and send a reply. Both the machine that issues the request and the server that responds use physical network addresses during their brief communication. How does the requester know the physical address of a server? Usually, it does not — it simply broadcasts the request to all machines on the local network. One or more servers respond.

Whenever a machine broadcasts a request for an address, it must uniquely identify itself. What information can be included in its request that will uniquely identify the machine? Any unique hardware identification suffices (e.g., the CPU serial number). However, the identification should be something that an executing program can obtain easily. Unfortunately, the length or format of CPU-specific information may vary among processor models, and we would like to devise a server that accepts requests from all machines on the physical network using a single format. Furthermore, engineers who design bootstrap code attempt to create a single software image that can execute on an arbitrary processor, and each processor model may have a slightly different set of instructions for obtaining a serial number.

6.2 Reverse Address Resolution Protocol (RARP)

The designers of TCP/IP protocols realized that there is another piece of uniquely identifying information readily available, namely, the machine's physical network address. Using the physical address as a unique identification has two advantages. Because a host obtains its physical addresses from the network interface hardware, such addresses are always available and do not have to be bound into the bootstrap code. Because the identifying information depends on the network and not on the CPU vendor or model, all machines on a given network will supply uniform, unique identifiers. Thus, the problem becomes the reverse of address resolution: given a physical network address, devise a scheme that will allow a server to map it into an internet address.

The TCP/IP protocol that allows a computer to obtain its IP address from a server is known as the *Reverse Address Resolution Protocol* (*RARP*). RARP is adapted from the ARP protocol of the previous chapter and uses the same message format shown in Figure 5.3. In practice, the RARP message sent to request an internet address is a little more general than what we have outlined above: it allows a machine to request the IP

†Chapter 21 discusses servers in detail.

address of a third party as easily as its own. It also allows for multiple physical network types.

Like an ARP message, a RARP message is sent from one machine to another encapsulated in the data portion of a network frame. For example, an Ethernet frame carrying a RARP request has the usual preamble, Ethernet source and destination addresses, and packet type fields in front of the frame. The frame type contains the value 8035_{16} to identify the contents of the frame as a RARP message. The data portion of the frame contains the 28-octet RARP message.

Figure 6.1 illustrates how a host uses RARP. The sender broadcasts a RARP request that specifies itself as both the sender and target machine, and supplies its physical network address in the target hardware address field. All computers on the network receive the request, but only those authorized to supply the RARP service process the request and send a reply; such computers are known informally as *RARP servers*. For RARP to succeed, the network must contain at least one RARP server.

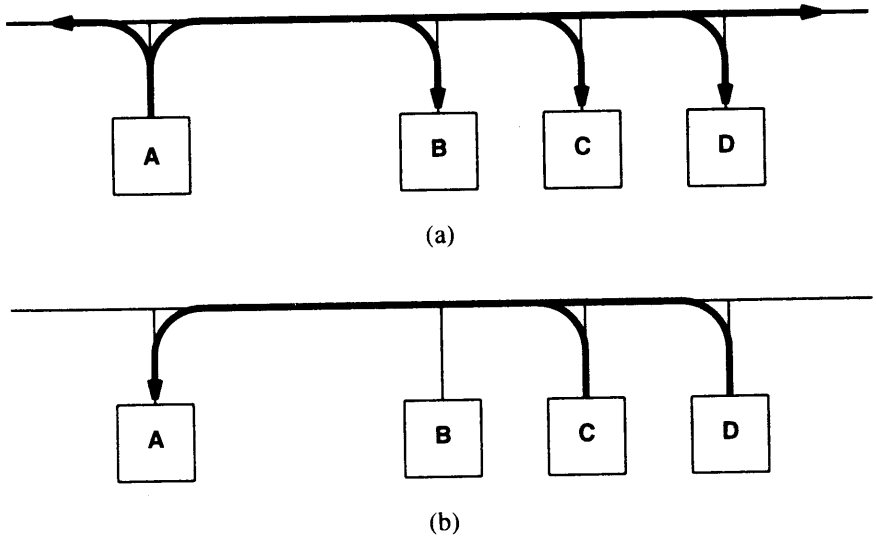


Figure 6.1 Example exchange using the RARP protocol. (a) Machine *A* broadcasts a RARP request specifying itself as a target, and (b) those machines authorized to supply the RARP service (*C* and *D*) reply directly to *A*.

Servers answer requests by filling in the target protocol address field, changing the message type from *request* to *reply*, and sending the reply back directly to the machine making the request. The original machine receives replies from all RARP servers, even though only the first is needed.

Keep in mind that all communication between the computer seeking its IP address and the server supplying it must be carried out using only the physical network. Furthermore, the protocol allows a host to ask about an arbitrary target. Thus, the sender supplies its hardware address separate from the target hardware address, and the server is careful to send the reply to the sender's hardware address. On an Ethernet, having a field for the sender's hardware address may seem redundant because the information is also contained in the Ethernet frame header. However, not all Ethernet hardware provides the operating system with access to the physical frame header.

6.3 Timing RARP Transactions

Like any communication on a best-effort delivery network, RARP requests and responses are susceptible to loss (including discard by the network interface if the CRC indicates that the frame was corrupted). Because RARP uses the physical network directly, no other protocol software will time the response or retransmit the request; RARP software must handle these tasks. In general, RARP is used only on local area networks like the Ethernet, where the probability of failure is low. If a network has only one RARP server, however, that machine may not be able to handle the load, so packets may be dropped.

Some computers that rely on RARP to boot, choose to retry indefinitely until they receive a response. Other implementations announce failure after only a few tries to avoid flooding the network with unnecessary broadcast traffic (e.g., in case the server is unavailable). On an Ethernet, network failure is less likely than server overload. Making RARP software retransmit quickly may have the unwanted effect of flooding a congested server with more traffic. Using a large delay ensures that servers have ample time to satisfy the request and return an answer.

6.4 Primary And Backup RARP Servers

The chief advantage of having several computers function as RARP servers is that it makes the system more reliable. If one server is down or too heavily loaded to respond, another answers the request. Thus, it is highly likely that the service will be available. The chief disadvantage of using many servers is that when a machine broadcasts a RARP request, the network becomes overloaded because all servers attempt to respond. On an Ethernet, for example, using multiple RARP servers makes the probability of collision high.

How can the RARP service be arranged to keep it available and reliable without incurring the cost of multiple, simultaneous replies? There are at least two possibilities, and they both involve delaying responses. In the first solution, each machine that makes RARP requests is assigned a *primary server*. Under normal circumstances, only the machine's primary server responds to its RARP request. All nonprimary servers receive the request but merely record its arrival time. If the primary server is unavailable,

the original machine will timeout waiting for a response and then rebroadcast the request. Whenever a nonprimary server receives a second copy of a RARP request within a short time of the first, it responds.

The second solution uses a similar scheme but attempts to avoid having all nonprimary servers transmit responses simultaneously. Each nonprimary machine that receives a request computes a random delay and then sends a response. Under normal circumstances, the primary server responds immediately and successive responses are delayed, so there is low probability that several responses arrive at the same time. When the primary server is unavailable, the requesting machine experiences a small delay before receiving a reply. By choosing delays carefully, the designer can ensure that requesting machines do not rebroadcast before they receive an answer.

6.5 Summary

At system startup, a computer that does not have permanent storage must contact a server to find its IP address before it can communicate using TCP/IP. This chapter examined the RARP protocol that uses physical network addressing to obtain the machine's internet address. The RARP mechanism supplies the target machine's physical hardware address to uniquely identify the processor and broadcasts the RARP request. Servers on the network receive the message, look up the mapping in a table (presumably from secondary storage), and reply to the sender. Once a machine obtains its IP address, it stores the address in memory and does not use RARP again until it reboots.

FOR FURTHER STUDY

The details of RARP are given in Finlayson, *et. al.* [RFC 903]. Finlayson [RFC 906] describes workstation bootstrapping using the TFTP protocol. Bradley and Brown [RFC 1293] specifies a related protocol, *Inverse ARP*. Inverse ARP permits a computer to query the machine at the opposite end of a hardware connection to determine its IP address, and was intended for computers on a connection-oriented network such as Frame Relay or ATM. Volume 2 of this text describes an example implementation of RARP.

Chapter 23 considers alternatives to RARP known as BOOTP and DHCP. Unlike the low-level address determination scheme RARP supplies, BOOTP and DHCP build on higher level protocols like IP and UDP. Chapter 23 compares the two approaches, discussing the strengths and weaknesses of each.

EXERCISES

6.1 A RARP server can broadcast RARP replies to all machines or transmit each reply directly to the machine that makes the request. Characterize a network technology in which broadcasting replies to all machines is beneficial.

6.2 RARP is a narrowly focused protocol in the sense that replies only contain one piece of information (i.e., the requested IP address). When a computer boots, it usually needs to know its name in addition to its Internet address. Extend RARP to supply the additional information.

6.3 How much larger will Ethernet frames become when information is added to RARP as described in the previous exercise?

6.4 Adding a second RARP server to a network increases reliability. Does it ever make sense to add a third? How about a fourth? Why or Why not?

6.5 The diskless workstations from one vendor use RARP to obtain their IP addresses, but always assume the response comes from the workstation's file server. The diskless machine then tries to obtain a boot image from that server. If it does not receive a response, the workstation enters an infinite loop broadcasting boot requests. Explain how adding a backup RARP server to such a configuration can cause the network to become congested with broadcasts. Hint: think of power failures.

6.6 Monitor a local network while you reboot various computers. Which use RARP?

6.7 The backup RARP servers discussed in the text use the arrival of a second request in a short period of time to trigger a reply. Consider the RARP server scheme that has all servers answer the first request, but avoids congestion by having each server delay a random time before answering. Under what circumstances could such a design yield better results than the design described in the text?

7

Internet Protocol: Connectionless Datagram Delivery

7.1 Introduction

Previous chapters review pieces of network hardware and software that make internet communication possible, explaining the underlying network technologies and address resolution. This chapter explains the fundamental principle of connectionless delivery and discusses how it is provided by the *Internet Protocol* (*IP*), which is one of the two major protocols used in internetworking (TCP being the other). We will study the format of IP datagrams and see how they form the basis for all internet communication. The next two chapters continue our examination of the Internet Protocol by discussing datagram routing and error handling.

7.2 A Virtual Network

Chapter 3 discusses an internet architecture in which routers connect multiple physical networks. Looking at the architecture may be misleading, because the focus should be on the interface that an internet provides to users, not on the interconnection technology.

A user thinks of an internet as a single virtual network that interconnects all hosts, and through which communication is possible; its underlying architecture is both hidden and irrelevant.

In a sense, an internet is an abstraction of physical networks because, at the lowest level, it provides the same functionality: accepting packets and delivering them. Higher levels of internet software add most of the rich functionality users perceive.

7.3 Internet Architecture And Philosophy

Conceptually, a TCP/IP internet provides three sets of services as shown in Figure 7.1; their arrangement in the figure suggests dependencies among them. At the lowest level, a connectionless delivery service provides a foundation on which everything rests. At the next level, a reliable transport service provides a higher level platform on which applications depend. We will soon explore each of these services, understand what they provide, and see the protocols associated with them.

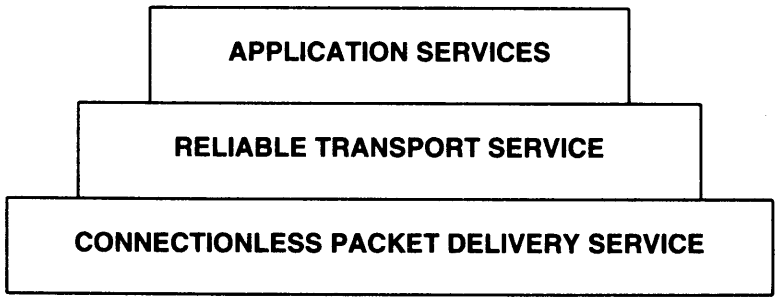


Figure 7.1 The three conceptual layers of internet services.

7.4 The Conceptual Service Organization

Although we can associate protocol software with each of the services in Figure 7.1, the reason for identifying them as conceptual parts of the internet is that they clearly point out the philosophical underpinnings of the design. The point is:

Internet software is designed around three conceptual networking services arranged in a hierarchy; much of its success has resulted because this architecture is surprisingly robust and adaptable.

One of the most significant advantages of this conceptual separation is that it becomes possible to replace one service without disturbing others. Thus, research and development can proceed concurrently on all three.

7.5 Connectionless Delivery System

The most fundamental internet service consists of a packet delivery system. Technically, the service is defined as an unreliable, best-effort, connectionless packet delivery system, analogous to the service provided by network hardware that operates on a best-effort delivery paradigm. The service is called *unreliable* because delivery is not guaranteed. The packet may be lost, duplicated, delayed, or delivered out of order, but the service will not detect such conditions, nor will it inform the sender or receiver. The service is called *connectionless* because each packet is treated independently from all others. A sequence of packets sent from one computer to another may travel over different paths, or some may be lost while others are delivered. Finally, the service is said to use *best-effort delivery* because the internet software makes an earnest attempt to deliver packets. That is, the internet does not discard packets capriciously; unreliability arises only when resources are exhausted or underlying networks fail.

7.6 Purpose Of The Internet Protocol

The protocol that defines the unreliable, connectionless delivery mechanism is called the *Internet Protocol* and is usually referred to by its initials, *IP*†. IP provides three important definitions. First, the IP protocol defines the basic unit of data transfer used throughout a TCP/IP internet. Thus, it specifies the exact format of all data as it passes across the internet. Second, IP software performs the *routing* function, choosing a path over which data will be sent. Third, in addition to the precise, formal specification of data formats and routing, IP includes a set of rules that embody the idea of unreliable packet delivery. The rules characterize how hosts and routers should process packets, how and when error messages should be generated, and the conditions under which packets can be discarded. IP is such a fundamental part of the design that a TCP/IP internet is sometimes called an *IP-based technology*.

We begin our consideration of IP in this chapter by looking at the packet format it specifies. We leave until later chapters the topics of routing and error handling.

7.7 The Internet Datagram

The analogy between a physical network and a TCP/IP internet is strong. On a physical network, the unit of transfer is a frame that contains a header and data, where the header gives information such as the (physical) source and destination addresses. The internet calls its basic transfer unit an *Internet datagram*, sometimes referred to as

†The abbreviation IP gives rise to the term "IP address."

an *IP datagram* or merely a *datagram*. Like a typical physical network frame, a datagram is divided into header and data areas. Also like a frame, the datagram header contains the source and destination addresses and a type field that identifies the contents of the datagram. The difference, of course, is that the datagram header contains IP addresses whereas the frame header contains physical addresses. Figure 7.2 shows the general form of a datagram:

Figure 7.2 General form of an IP datagram, the TCP/IP analogy to a network frame. IP specifies the header format including the source and destination IP addresses. IP does not specify the format of the data area; it can be used to transport arbitrary data.

7.7.1 Datagram Format

Now that we have described the general layout of an IP datagram, we can look at the contents in more detail. Figure 7.3 shows the arrangement of fields in a datagram:

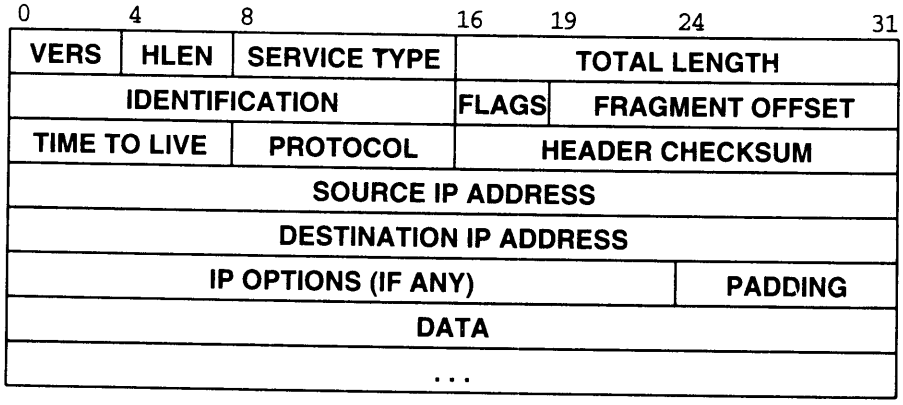


Figure 7.3 Format of an Internet datagram, the basic unit of transfer in a TCP/IP internet.

Because datagram processing occurs in software, the contents and format are not constrained by any hardware. For example, the first 4-bit field in a datagram (*VERS*) contains the version of the IP protocol that was used to create the datagram. It is used to verify that the sender, receiver, and any routers in between them agree on the format

of the datagram. All IP software is required to check the version field before processing a datagram to ensure it matches the format the software expects. If standards change, machines will reject datagrams with protocol versions that differ from theirs, preventing them from misinterpreting datagram contents according to an outdated format. The current IP protocol version is *4*. Consequently, the term *IPv4* is often used to denote the current protocol.

The header length field (*HLEN*), also 4 bits, gives the datagram header length measured in 32-bit words. As we will see, all fields in the header have fixed length except for the *IP OPTIONS* and corresponding *PADDING* fields. The most common header, which contains no options and no padding, measures 20 octets and has a header length field equal to *5*.

The *TOTAL LENGTH* field gives the length of the IP datagram measured in octets, including octets in the header and data. The size of the data area can be computed by subtracting the length of the header (*HLEN*) from the *TOTAL LENGTH*. Because the *TOTAL LENGTH* field is 16 bits long, the maximum possible size of an IP datagram is 2^{16} or 65,535 octets. In most applications this is not a severe limitation. It may become more important in the future if higher speed networks can carry data packets larger than 65,535 octets.

7.7.2 Datagram Type Of Service And Differentiated Services

Informally called *Type Of Service* (*TOS*), the 8-bit *SERVICE TYPE* field specifies how the datagram should be handled. The field was originally divided into five subfields as shown in Figure 7.4:

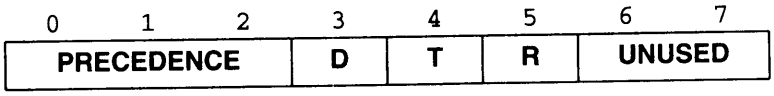


Figure 7.4 The original five subfields that comprise the 8-bit *SERVICE TYPE* field.

Three *PRECEDENCE* bits specify datagram precedence, with values ranging from 0 (normal precedence) through 7 (network control), allowing senders to indicate the importance of each datagram. Although some routers ignore type of service, it is an important concept because it provides a mechanism that can allow control information to have precedence over data. For example, many routers use a precedence value of 6 or 7 for routing traffic to make it possible for the routers to exchange routing information even when networks are congested.

Bits *D*, *T*, and *R* specify the type of transport desired for the datagram. When set, the *D* bit requests low delay, the *T* bit requests high throughput, and the *R* bit requests high reliability. Of course, it may not be possible for an internet to guarantee the type

of transport requested (i.e., it could be that no path to the destination has the requested property). Thus, we think of the transport request as a hint to the routing algorithms, not as a demand. If a router does know more than one possible route to a given destination, it can use the type of transport field to select one with characteristics closest to those desired. For example, suppose a router can select between a low capacity leased line or a high bandwidth (but high delay) satellite connection. Datagrams carrying keystrokes from a user to a remote computer could have the *D* bit set requesting that they be delivered as quickly as possible, while datagrams carrying a bulk file transfer could have the *T* bit set requesting that they travel across the high capacity satellite path.

In the late 1990s, the IETF redefined the meaning of the 8-bit *SERVICE TYPE* field to accommodate a set of *differentiated services* (*DS*). Figure 7.5 illustrates the resulting definition.

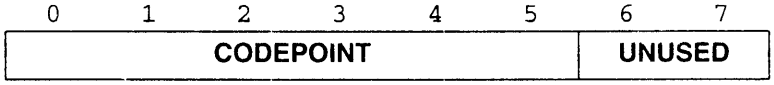


Figure 7.5 The differentiated services (DS) interpretation of the *SERVICE TYPE* field in an IP datagram.

Under the differentiated services interpretation, the first six bits comprise a *codepoint*, which is sometimes abbreviated *DSCP*, and the last two bits are left unused. A codepoint value maps to an underlying service definition, typically through an array of pointers. Although it is possible to define 64 separate services, the designers suggest that a given router will only have a few services, and multiple codepoints will map to each service. Moreover, to maintain backward compatibility with the original definition, the standard distinguishes between the first three bits of the codepoint (the bits that were formerly used for precedence) and the last three bits. When the last three bits contain zero, the precedence bits define eight broad classes of service that adhere to the same guidelines as the original definition: datagrams with a higher number in their precedence field are given preferential treatment over datagrams with a lower number. That is, the eight ordered classes are defined by codepoint values of the form:

xxx000

where x denotes either a zero or a one.

The differentiated services design also accommodates another existing practice — the widespread use of precedence 6 or 7 for routing traffic. The standard includes a special case to handle these precedence values. A router is required to implement at least two priority schemes: one for normal traffic and one for high-priority traffic. When the last three bits of the *CODEPOINT* field are zero, the router must map a

codepoint with precedence 6 or 7 into the higher priority class and other codepoint values into the lower priority class. Thus, if a datagram arrives that was sent using the original TOS scheme, a router using the differentiated services scheme will honor precedence 6 and 7 as the datagram sender expects.

The 64 codepoint values are divided into three administrative groups as Figure 7.6 illustrates.

Pool	Codepoint	Assigned By
1	xxxxx0	Standards organization
2	xxxx11	Local or experimental
3	xxxx01	Local or experimental for now

Figure 7.6 The three administrative pools of codepoint values.

As the figure indicates, half of the values (i.e., the 32 values in pool *1*) must be assigned interpretations by the IETF. Currently, all values in pools *2* and *3* are available for experimental or local use. However, if the standards bodies exhaust all values in pool *1*, they may also choose to assign values in pool *3*.

The division into pools may seem unusual because it relies on the low-order bits of the value to distinguish pools. Thus, rather than a contiguous set of values, pool *1* contains every other codepoint value (i.e., the even numbers between 2 and 64). The division was chosen to keep the eight codepoints corresponding to values xxx000 in the same pool.

Whether the original TOS interpretation or the revised differentiated services interpretation is used, it is important to realize that routing software must choose from among the underlying physical network technologies at hand and must adhere to local policies. Thus, specifying a level of service in a datagram does not guarantee that routers along the path will agree to honor the request. To summarize:

> *We regard the service type specification as a hint to the routing algorithm that helps it choose among various paths to a destination based on local policies and its knowledge of the hardware technologies available on those paths. An internet does not guarantee to provide any particular type of service.*

7.7.3 Datagram Encapsulation

Before we can understand the next fields in a datagram, it is important to consider how datagrams relate to physical network frames. We start with a question: "How large can a datagram be?" Unlike physical network frames that must be recognized by hardware, datagrams are handled by software. They can be of any length the protocol designers choose. We have seen that the IPv4 datagram format allots 16 bits to the total length field, limiting the datagram to at most 65,535 octets.

More fundamental limits on datagram size arise in practice. We know that as datagrams move from one machine to another, they must always be transported by the underlying physical network. To make internet transportation efficient, we would like to guarantee that each datagram travels in a distinct physical frame. That is, we want our abstraction of a physical network packet to map directly onto a real packet if possible.

The idea of carrying one datagram in one network frame is called *encapsulation*. To the underlying network, a datagram is like any other message sent from one machine to another. The hardware does not recognize the datagram format, nor does it understand the IP destination address. Thus, as Figure 7.7 shows, when one machine sends an IP datagram to another, the entire datagram travels in the data portion of the network frame†.

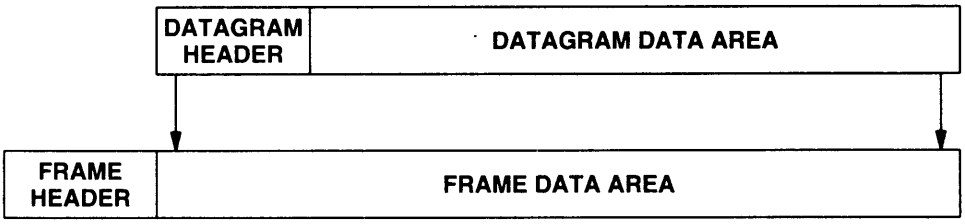


Figure 7.7 The encapsulation of an IP datagram in a frame. The physical network treats the entire datagram, including the header, as data.

7.7.4 Datagram Size, Network MTU, and Fragmentation

In the ideal case, the entire IP datagram fits into one physical frame, making transmission across the physical net efficient. To achieve such efficiency, the designers of IP might have selected a maximum datagram size such that a datagram would always fit into one frame. But which frame size should be chosen? After all, a datagram may travel across many types of physical networks as it moves across an internet to its final destination.

To understand the problem, we need a fact about network hardware: each packet-switching technology places a fixed upper bound on the amount of data that can be transferred in one physical frame. For example, Ethernet limits transfers to 1500‡ octets of data, while FDDI permits approximately 4470 octets of data per frame. We refer to these limits as the network's *maximum transfer unit* or *MTU*. MTU sizes can be quite small: some hardware technologies limit transfers to 128 octets or less. Limiting datagrams to fit the smallest possible MTU in the internet makes transfers inefficient when datagrams pass across a network that can carry larger size frames. However, allowing datagrams to be larger than the minimum network MTU in an internet means that a datagram may not always fit into a single network frame.

†A field in the frame header usually identifies the data being carried; Ethernet uses the type value 0800_{16} to specify that the data area contains an encapsulated IP datagram.

‡The limit of 1500 comes from the Ethernet specification; when used with a SNAP header the IEEE 802.3 standard limits data to 1492 octets. Some vendors' implementations allow slightly larger transfers.

The choice should be obvious: the point of the internet design is to hide underlying network technologies and make communication convenient for the user. Thus, instead of designing datagrams that adhere to the constraints of physical networks, TCP/IP software chooses a convenient initial datagram size and arranges a way to divide large datagrams into smaller pieces when the datagram needs to traverse a network that has a small MTU. The small pieces into which a datagram is divided are called *fragments*, and the process of dividing a datagram is known as *fragmentation*.

As Figure 7.8 illustrates, fragmentation usually occurs at a router somewhere along the path between the datagram source and its ultimate destination. The router receives a datagram from a network with a large MTU and must send it over a network for which the MTU is smaller than the datagram size.

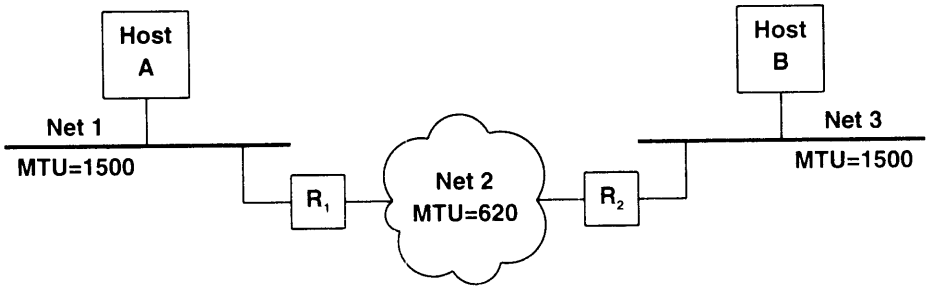


Figure 7.8 An illustration of where fragmentation occurs. Router R_1 fragments large datagrams sent from A to B; R_2 fragments large datagrams sent from B to A.

In the figure, both hosts attach directly to Ethernets which have an MTU of 1500 octets. Thus, both hosts can generate and send datagrams up to 1500 octets long. The path between them, however, includes a network with an MTU of 620. If host A sends host B a datagram larger than 620 octets, router R_1 will fragment the datagram. Similarly, if B sends a large datagram to A, router R_2 will fragment the datagram.

Fragment size is chosen so each fragment can be shipped across the underlying network in a single frame. In addition, because IP represents the offset of the data in multiples of eight octets, the fragment size must be chosen to be a multiple of eight. Of course, choosing the multiple of eight octets nearest to the network MTU does not usually divide the datagram into equal size pieces; the last piece is often shorter than the others. Fragments must be *reassembled* to produce a complete copy of the original datagram before it can be processed at the destination.

The IP protocol does not limit datagrams to a small size, nor does it guarantee that large datagrams will be delivered without fragmentation. The source can choose any datagram size it thinks appropriate; fragmentation and reassembly occur automatically, without the source taking special action. The IP specification states that routers must accept datagrams up to the maximum of the MTUs of networks to which they attach.

In addition, a router must always handle datagrams of up to 576 octets. (Hosts are also required to accept, and reassemble if necessary, datagrams of at least 576 octets.)

Fragmenting a datagram means dividing it into several pieces. It may surprise you to learn that each piece has the same format as the original datagram. Figure 7.9 illustrates the result of fragmentation.

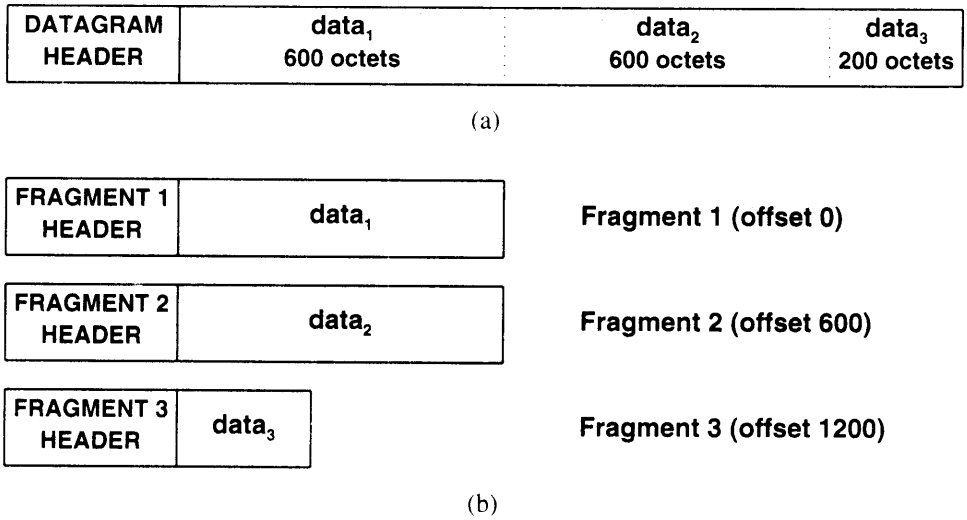


Figure 7.9 (a) An original datagram carrying 1400 octets of data and (b) the three fragments for network MTU of 620. Headers 1 and 2 have the *more fragments* bit set. Offsets shown are decimal octets; they must be divided by 8 to get the value stored in the fragment headers.

Each fragment contains a datagram header that duplicates most of the original datagram header (except for a bit in the *FLAGS* field that shows it is a fragment), followed by as much data as can be carried in the fragment while keeping the total length smaller than the MTU of the network over which it must travel.

7.7.5 Reassembly Of Fragments

Should a datagram be reassembled after passing across one network, or should the fragments be carried to the final host before reassembly? In a TCP/IP internet, once a datagram has been fragmented, the fragments travel as separate datagrams all the way to the ultimate destination where they must be reassembled. Preserving fragments all the way to the ultimate destination has two disadvantages. First, because datagrams are not reassembled immediately after passing across a network with small MTU, the small fragments must be carried from the point of fragmentation to the ultimate destination.

Reassembling datagrams at the ultimate destination can lead to inefficiency: even if some of the physical networks encountered after the point of fragmentation have large MTU capability, only small fragments traverse them. Second, if any fragments are lost, the datagram cannot be reassembled. The receiving machine starts a *reassembly timer* when it receives an initial fragment. If the timer expires before all fragments arrive, the receiving machine discards the surviving pieces without processing the datagram. Thus, the probability of datagram loss increases when fragmentation occurs because the loss of a single fragment results in loss of the entire datagram.

Despite the minor disadvantages, performing reassembly at the ultimate destination works well. It allows each fragment to be routed independently, and does not require intermediate routers to store or reassemble fragments.

7.7.6 Fragmentation Control

Three fields in the datagram header, *IDENTIFICATION*, *FLAGS*, and *FRAGMENT OFFSET*, control fragmentation and reassembly of datagrams. Field *IDENTIFICATION* contains a unique integer that identifies the datagram. Recall that when a router fragments a datagram, it copies most of the fields in the datagram header into each fragment. Thus, the *IDENTIFICATION* field must be copied. Its primary purpose is to allow the destination to know which arriving fragments belong to which datagrams. As a fragment arrives, the destination uses the *IDENTIFICATION* field along with the datagram source address to identify the datagram. Computers sending IP datagrams must generate a unique value for the *IDENTIFICATION* field for each datagram†. One technique used by IP software keeps a global counter in memory, increments it each time a new datagram is created, and assigns the result as the datagram's *IDENTIFICATION* field.

Recall that each fragment has exactly the same format as a complete datagram. For a fragment, field *FRAGMENT OFFSET* specifies the offset in the original datagram of the data being carried in the fragment, measured in units of 8 octets‡, starting at offset zero. To reassemble the datagram, the destination must obtain all fragments starting with the fragment that has offset *0* through the fragment with highest offset. Fragments do not necessarily arrive in order, and there is no communication between the router that fragmented the datagram and the destination trying to reassemble it.

The low-order two bits of the 3-bit *FLAGS* field control fragmentation. Usually, application software using TCP/IP does not care about fragmentation because both fragmentation and reassembly are automatic procedures that occur at a low level in the operating system, invisible to end users. However, to test internet software or debug operational problems, it may be important to test sizes of datagrams for which fragmentation occurs. The first control bit aids in such testing by specifying whether the datagram may be fragmented. It is called the *do not fragment* bit because setting it to *1* specifies that the datagram should not be fragmented. An application may choose to disallow fragmentation when only the entire datagram is useful. For example, consider a bootstrap sequence in which a small embedded system executes a program in ROM that sends a request over the internet to which another machine responds by sending

†In theory, retransmissions of a packet can carry the same *IDENTIFICATION* field as the original; in practice, higher-level protocols perform retransmission, resulting in a new datagram with its own *IDENTIFICATION*.

‡To save space in the header, offsets are specified in multiples of 8 octets.

back a memory image. If the embedded system has been designed so it needs the entire image or none of it, the datagram should have the *do not fragment* bit set. Whenever a router needs to fragment a datagram that has the *do not fragment* bit set, the router discards the datagram and sends an error message back to the source.

The low order bit in the *FLAGS* field specifies whether the fragment contains data from the middle of the original datagram or from the end. It is called the *more fragments* bit. To see why such a bit is needed, consider the IP software at the ultimate destination attempting to reassemble a datagram. It will receive fragments (possibly out of order) and needs to know when it has received all fragments for a datagram. When a fragment arrives, the *TOTAL LENGTH* field in the header refers to the size of the fragment and not to the size of the original datagram, so the destination cannot use the *TOTAL LENGTH* field to tell whether it has collected all fragments. The *more fragments* bit solves the problem easily: once the destination receives a fragment with the *more fragments* bit turned off, it knows this fragment carries data from the tail of the original datagram. From the *FRAGMENT OFFSET* and *TOTAL LENGTH* fields, it can compute the length of the original datagram. By examining the *FRAGMENT OFFSET* and *TOTAL LENGTH* of all fragments that have arrived, a receiver can tell whether the fragments on hand contain all pieces needed to reassemble the original datagram.

7.7.7 Time to Live (TTL)

In principle, field *TIME TO LIVE* specifies how long, in seconds, the datagram is allowed to remain in the internet system. The idea is both simple and important: whenever a computer injects a datagram into the internet, it sets a maximum time that the datagram should survive. Routers and hosts that process datagrams must decrement the *TIME TO LIVE* field as time passes and remove the datagram from the internet when its time expires.

Estimating exact times is difficult because routers do not usually know the transit time for physical networks. A few rules simplify processing and make it easy to handle datagrams without synchronized clocks. First, each router along the path from source to destination is required to decrement the *TIME TO LIVE* field by *1* when it processes the datagram header. Furthermore, to handle cases of overloaded routers that introduce long delays, each router records the local time when the datagram arrives, and decrements the *TIME TO LIVE* by the number of seconds the datagram remained inside the router waiting for service†.

Whenever a *TIME TO LIVE* field reaches zero, the router discards the datagram and sends an error message back to the source. The idea of keeping a timer for datagrams is interesting because it guarantees that datagrams cannot travel around an internet forever, even if routing tables become corrupt and routers route datagrams in a circle.

Although once important, the notion of a router delaying a datagram for many seconds is now outdated — current routers and networks are designed to forward each datagram within a reasonable time. If the delay becomes excessive, the router simply discards the datagram. Thus, in practice, the *TIME TO LIVE* acts as a "hop limit" rather than an estimate of delay. Each router only decrements the value by 1.

†In practice, modern routers do not hold datagrams for multiple seconds.

7.7.8 Other Datagram Header Fields

Field *PROTOCOL* is analogous to the type field in a network frame; the value specifies which high-level protocol was used to create the message carried in the *DATA* area of the datagram. In essence, the value of *PROTOCOL* specifies the format of the *DATA* area. The mapping between a high level protocol and the integer value used in the *PROTOCOL* field must be administered by a central authority to guarantee agreement across the entire Internet.

Field *HEADER CHECKSUM* ensures integrity of header values. The IP checksum is formed by treating the header as a sequence of 16-bit integers (in network byte order), adding them together using one's complement arithmetic, and then taking the one's complement of the result. For purposes of computing the checksum, field *HEADER CHECKSUM* is assumed to contain zero.

It is important to note that the checksum only applies to values in the IP header and not to the data. Separating the checksum for headers and data has advantages and disadvantages. Because the header usually occupies fewer octets than the data, having a separate checksum reduces processing time at routers which only need to compute header checksums. The separation also allows higher level protocols to choose their own checksum scheme for the data. The chief disadvantage is that higher level protocols are forced to add their own checksum or risk having corrupted data go undetected.

Fields *SOURCE IP ADDRESS* and *DESTINATION IP ADDRESS* contain the 32-bit IP addresses of the datagram's sender and intended recipient. Although the datagram may be routed through many intermediate routers, the source and destination fields never change; they specify the IP addresses of the original source and ultimate destination†.

The field labeled *DATA* in Figure 7.3 shows the beginning of the data area of the datagram. Its length depends, of course, on what is being sent in the datagram. The *IP OPTIONS* field, discussed below, is variable length. The field labeled *PADDING*, depends on the options selected. It represents bits containing zero that may be needed to ensure the datagram header extends to an exact multiple of 32 bits (recall that the header length field is specified in units of 32-bit words).

7.8 Internet Datagram Options

The *IP OPTIONS* field following the destination address is not required in every datagram; options are included primarily for network testing or debugging. Options processing is an integral part of the IP protocol, however, so all standard implementations must include it.

The length of the *IP OPTIONS* field varies depending on which options are selected. Some options are one octet long; they consist of a single octet *option code*. Other options are variable length. When options are present in a datagram, they appear contiguously, with no special separators between them. Each option consists of a single octet option code, which may be followed by a single octet length and a set of data octets for that option. The option code octet is divided into three fields as Figure 7.10 shows.

†An exception is made when the datagram includes the source route options listed below.

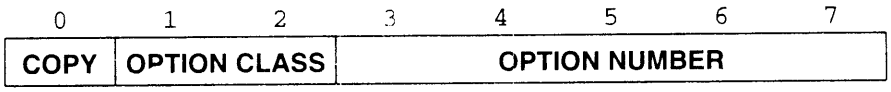


Figure 7.10 The division of the option code octet into three fields of length 1, 2, and 5 bits.

The fields of the *OPTION CODE* consist of a 1-bit *COPY* flag, a 2-bit *OPTION CLASS*, and the 5-bit *OPTION NUMBER*. The *COPY* flag controls how routers treat options during fragmentation. When the *COPY* bit is set to *1*, it specifies that the option should be copied into all fragments. When set to *0*, the *COPY* bit means that the option should only be copied into the first fragment and not into all fragments.

The *OPTION CLASS* and *OPTION NUMBER* bits specify the general class of the option and a specific option in that class. The table in Figure 7.11 shows how option classes are assigned.

Option Class	Meaning
0	Datagram or network control
1	Reserved for future use
2	Debugging and measurement
3	Reserved for future use

Figure 7.11 Classes of IP options as encoded in the *OPTION CLASS* bits of an option code octet.

The table in Figure 7.12 lists examples of options that can accompany an IP datagram and gives their *OPTION CLASS* and *OPTION NUMBER* values. As the list shows, most options are used for control purposes.

Option Class	Option Number	Length	Description
0	0	-	End of option list. Used if options do not end at end of header (see header padding field for explanation).
0	1	-	No operation. Used to align octets in a list of options.
0	2	11	Security and handling restrictions (for military applications).
0	3	var	Loose source route. Used to request routing that includes the specified routers.
0	7	var	Record route. Used to trace a route.
0	8	4	Stream identifier. Used to carry a SATNET stream identifier (obsolete).
0	9	var	Strict source route. Used to specify a exact path through the internet.
0	11	4	MTU Probe. Used for path MTU discovery.
0	12	4	MTU Reply. Used for path MTU discovery.
0	20	4	Router Alert. Router should examine this datagram even if not an addressee.
2	4	var	Internet timestamp. Used to record timestamps along the route.
2	18	var	Traceroute. Used by traceroute program to find routers along a path.

Figure 7.12 Examples of IP options with their numeric class and number codes. The value *var* in the length column stands for *variable*.

7.8.1 Record Route Option

The routing and timestamp options are the most interesting because they provide a way to monitor or control how internet routers route datagrams. The *record route* option allows the source to create an empty list of IP addresses and arrange for each router that handles the datagram to add its IP address to the list. Figure 7.13 shows the format of the record route option.

As described above, the *CODE* field contains the option class and option number (*0* and *7* for record route). The *LENGTH* field specifies the total length of the option as it appears in the IP datagram, including the first three octets. The fields starting with the one labeled *FIRST IP ADDRESS* comprise the area reserved for recording internet addresses. The *POINTER* field specifies the offset within the option of the next available slot.

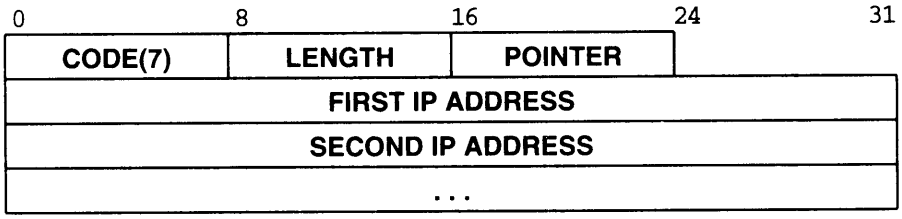


Figure 7.13 The format of the record route option in an IP datagram. The option begins with three octets immediately followed by a list of addresses. Although the diagram shows addresses in 32 bit units, they are not aligned on any octet boundary in a datagram.

Whenever a machine handles a datagram that has the record route option set, the machine adds its address to the record route list (enough space must be allocated in the option by the original source to hold all entries that will be needed). To add itself to the list, a machine first compares the pointer and length fields. If the pointer is greater than the length, the list is full, so the machine forwards the datagram without inserting its entry. If the list is not full, the machine inserts its 4-octet IP address at the position specified by the *POINTER*, and increments the *POINTER* by four.

When the datagram arrives, the destination machine can extract and process the list of IP addresses. Usually, a computer that receives a datagram ignores the recorded route. Using the record route option requires two machines that agree to cooperate; a computer will not automatically receive recorded routes in incoming datagrams after it turns on the record route option in outgoing datagrams. The source must agree to enable the record route option and the destination must agree to process the resultant list.

7.8.2 Source Route Options

Another idea that network builders find interesting is the *source route* option. The idea behind source routing is that it provides a way for the sender to dictate a path through the internet. For example, to test the throughput over a particular physical network, *N*, system administrators can use source routing to force IP datagrams to traverse network *N* even if routers would normally choose a path that did not include it. The ability to make such tests is especially important in a production environment, because it gives the network manager freedom to route users' datagrams over networks that are known to operate correctly while simultaneously testing other networks. Of course, source routing is only useful to people who understand the network topology; the average user has no need to know or use it.

IP supports two forms of source routing. One form, called *strict source routing*, specifies a routing path by including a sequence of IP addresses in the option as Figure 7.14 shows.

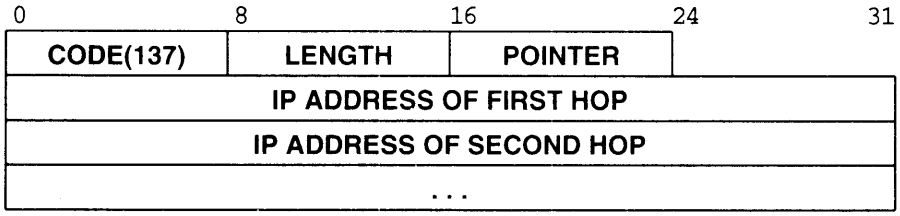


Figure 7.14 The strict source route option specifies an exact route by giving a list of IP addresses the datagram must follow.

Strict source routing means that the addresses specify the exact path the datagram must follow to reach its destination. The path between two successive addresses in the list must consist of a single physical network; an error results if a router cannot follow a strict source route. The other form, called *loose source routing*, also includes a sequence of IP addresses. It specifies that the datagram must follow the sequence of IP addresses, but allows multiple network hops between successive addresses on the list.

Both source route options require routers along the path to overwrite items in the address list with their local network addresses. Thus, when the datagram arrives at its destination, it contains a list of all addresses visited, exactly like the list produced by the record route option.

The format of a source route option resembles that of the record route option shown above. Each router examines the *POINTER* and *LENGTH* fields to see if the list has been exhausted. If it has, the pointer is greater than the length, and the router routes the datagram to its destination as usual. If the list is not exhausted, the router follows the pointer, picks up the IP address, replaces it with the router's address†, and routes the datagram using the address obtained from the list.

7.8.3 Timestamp Option

The *timestamp option* works like the record route option in that the timestamp option contains an initially empty list, and each router along the path from source to destination fills in one item in the list. Each entry in the list contains two 32-bit items: the IP address of the router that supplied the entry and a 32-bit integer timestamp. Figure 7.15 shows the format of the timestamp option.

†A router has one address for each interface; it records the address that corresponds to the network over which it routes the datagram.

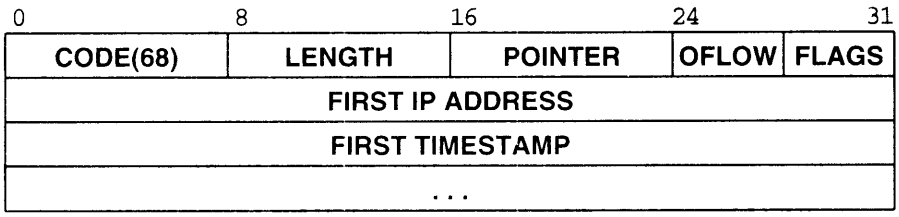


Figure 7.15 The format of the timestamp option. Bits in the FLAGS field control the exact format and rules routers use to process this option.

In the figure, the *LENGTH* and *POINTER* fields are used to specify the length of the space reserved for the option and the location of the next unused slot (exactly as in the record route option). The 4-bit *OFLOW* field contains an integer count of routers that could not supply a timestamp because the option was too small.

The value in the 4-bit *FLAGS* field controls the exact format of the option and tells how routers should supply timestamps. The values are:

Flags value	Meaning
0	Record timestamps only; omit IP addresses.
1	Precede each timestamp by an IP address (this is the format shown in Figure 7.15).
3	IP addresses are specified by sender; a router only records a timestamp if the next IP address in the list matches the router's IP address.

Figure 7.16 The interpretation of values in the FLAGS field of a timestamp option.

Timestamps give the time and date at which a router handles the datagram, expressed as milliseconds since midnight, Universal Time†. If the standard representation for time is unavailable, the router can use any representation of local time provided it turns on the high-order bit in the timestamp field. Of course, timestamps issued by independent computers are not always consistent even if represented in universal time; each machine reports time according to its local clock, and clocks may differ. Thus, timestamp entries should always be treated as estimates, independent of the representation.

It may seem odd that the timestamp option includes a mechanism to have routers record their IP addresses along with timestamps because the record route option already provides that capability. However, recording IP addresses with timestamps eliminates

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

ambiguity. Having an address recorded along with each timestamp is also useful because it allows the receiver to know exactly which path the datagram followed.

7.8.4 Processing Options During Fragmentation

The idea behind the *COPY* bit in the option *CODE* field should now be clear. When fragmenting a datagram, a router replicates some IP options in all fragments while it places others in only one fragment. For example, consider the option used to record the datagram route. We said that each fragment will be handled as an independent datagram, so there is no guarantee that all fragments follow the same path to the destination. If all fragments contained the record route option, the destination might receive a different list of routes from each fragment. It could not produce a single, meaningful list of routes for the reassembled datagram. Therefore, the IP standard specifies that the record route option should only be copied into one of the fragments.

Not all IP options can be restricted to one fragment. Consider the source route option, for example, that specifies how a datagram should travel through the internet. Source routing information must be replicated in all fragment headers, or fragments will not follow the specified route. Thus, the code field for source route specifies that the option must be copied into all fragments.

7.9 Summary

The fundamental service provided by TCP/IP internet software is a connectionless, unreliable, best-effort packet delivery system. The Internet Protocol (IP) formally specifies the format of internet packets, called *datagrams*, and informally embodies the ideas of connectionless delivery. This chapter concentrated on datagram formats; later chapters will discuss IP routing and error handling.

Analogous to a physical frame, the IP datagram is divided into header and data areas. Among other information, the datagram header contains the source and destination IP addresses, fragmentation control, precedence, and a checksum used to catch transmission errors. Besides fixed-length fields, each datagram header can contain an options field. The options field is variable length, depending on the number and type of options used as well as the size of the data area allocated for each option. Intended to help monitor and control an internet, options allow one to specify or record routing information, or to gather timestamps as the datagram traverses an internet.

FOR FURTHER STUDY

Postel [1980] discusses possible ways to approach internet protocols, addressing, and routing. In later publications, Postel [RFC 791] gives the standard for the Internet Protocol. Braden [RFC 1122] further refines the standard. Hornig [RFC 894] specifies

the standard for the transmission of IP datagrams across an Ethernet. Clark [RFC 815] describes efficient reassembly of fragments; Kent and Mogul [1987] discusses the disadvantages of fragmentation.

Nichols et. al. [RFC 2474] specifies the differentiated service interpretation of the service type bits in datagram headers, and Blake et. al. [RFC 2475] discusses an architecture for differentiated services. In addition to the packet format, many constants needed in the network protocols are also standardized; the values can be found in the Official Internet Protocols RFC, which is issued periodically.

An alternative internet protocol suite known as *XNS*, is given in Xerox [1981]. Boggs *et. al.* [1980] describes the PARC Universal Packet (PUP) protocol, an abstraction from *XNS* closely related to the IP datagram.

EXERCISES

7.1 What is the single greatest advantage of having the IP checksum cover only the datagram header and not the data? What is the disadvantage?

7.2 Is it ever necessary to use an IP checksum when sending packets over an Ethernet? Why or why not?

7.3 What is the MTU size for a Frame Relay network? Hyperchannel? an ATM network?

7.4 Do you expect a high-speed local area network to have larger or smaller MTU size than a wide area network?

7.5 Argue that fragments should have small, nonstandard headers.

7.6 Find out when the IP protocol version last changed. Is having a protocol version number useful?

7.7 Extend the previous exercise by arguing that if the IP version changes, it makes more sense to assign a new frame type than to encode the version number in the datagram.

7.8 Can you imagine why a one's complement checksum was chosen for IP instead of a cyclic redundancy check?

7.9 What are the advantages of doing reassembly at the ultimate destination instead of doing it after the datagram travels across one network?

7.10 What is the minimum network MTU required to send an IP datagram that contains at least one octet of data?

7.11 Suppose you are hired to implement IP datagram processing in hardware. Is there any rearrangement of fields in the header that would have made your hardware more efficient? Easier to build?

7.12 If you have access to an implementation of IP, revise it and test your locally available implementations of IP to see if they reject IP datagrams with an out-of-date version number.

7.13 When a minimum-size IP datagram travels across an Ethernet, how large is the frame?

7.14 The differentiated services interpretation of the *SERVICE TYPE* field allows up to 64 separate service levels. Argue that fewer levels are needed (i.e., make a list of all possible services that a user might access).

7.15 The differentiated service definition was chosen to make it backward compatible with the original type-of-service priority bits. Will the backward compatibility force implementations to be less efficient than an alternative scheme? Explain.

8

Internet Protocol: Routing IP Datagrams

8.1 Introduction

We have seen that all internet services use an underlying, connectionless packet delivery system, and that the basic unit of transfer in a TCP/IP internet is the IP datagram. This chapter adds to the description of connectionless service by describing how routers forward IP datagrams and deliver them to their final destinations. We think of the datagram format from Chapter 7 as characterizing the static aspects of the Internet Protocol. The description of routing in this chapter characterizes the operational aspects. The next chapter completes our basic presentation of IP by describing how errors are handled. Chapter 10 then describes extensions for classless and subnet addressing, and later chapters show how other protocols use IP to provide higher-level services.

8.2 Routing In An Internet

In a packet switching system, *routing* refers to the process of choosing a path over which to send packets, and *router* refers to a computer making the choice. Routing occurs at several levels. For example, within a wide area network that has multiple physical connections between packet switches, the network itself is responsible for routing packets from the time they enter until they leave. Such internal routing is completely self-contained inside the wide area network. Machines on the outside cannot participate in decisions; they merely view the network as an entity that delivers packets.

Remember that the goal of IP is to provide a virtual network that encompasses multiple physical networks and offers a connectionless datagram delivery service. Thus, we will focus on *IP forwarding*, which is also called *internet routing* or *IP routing*†. The information used to make routing decisions is known as *IP routing information*. Like routing within a single physical network, IP routing chooses a path over which a datagram should be sent. Unlike routing within a single network, the IP routing algorithm must choose how to send a datagram across multiple physical networks.

Routing in an internet can be difficult, especially among computers that have multiple physical network connections. Ideally, the routing software would examine network load, datagram length, or the type of service specified in the datagram header when selecting the best path. Most internet routing software is much less sophisticated, however, and selects routes based on fixed assumptions about shortest paths.

To understand IP routing completely, we must review the architecture of a TCP/IP internet. First, recall that an internet is composed of multiple physical networks interconnected by computers called *routers*. Each router has direct connections to two or more networks. By contrast, a host computer usually connects directly to one physical network. We know that it is possible, however, to have a multi-homed host connected directly to multiple networks.

Both hosts and routers participate in routing an IP datagram to its destination. When an application program on a host attempts to communicate, the TCP/IP protocols eventually generate one or more IP datagrams. The host must make an initial routing decision when it chooses where to send the datagrams. As Figure 8.1 shows, hosts must make routing decisions even if they have only one network connection.

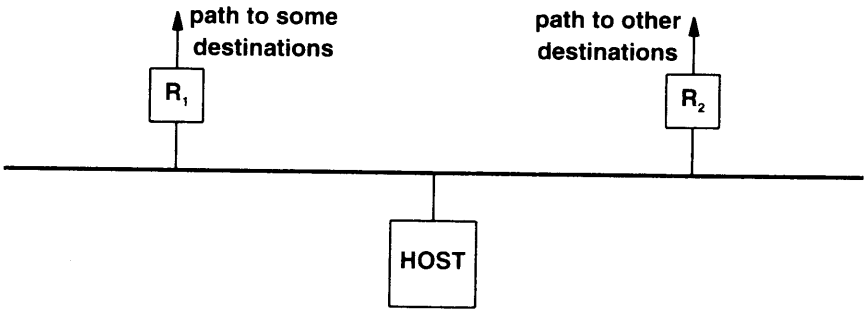


Figure 8.1 An example of a singly-homed host that must route datagrams. The host must choose to send a datagram either to router R_1 or to router R_2, because each router provides the best path to some destinations.

The primary purpose of routers is to make IP routing decisions. What about multi-homed hosts? Any computer with multiple network connections can act as a router, and as we will see, multi-homed hosts running TCP/IP have all the software

†Chapter 18 describes a related topic known as *layer 3 switching* or *IP switching*.

needed for routing. Furthermore, sites that cannot afford separate routers sometimes use general-purpose timesharing machines as both hosts and routers. However, the TCP/IP standards draw a sharp distinction between the functions of a host and those of a router, and sites that try to mix host and router functions on a single machine sometimes find that their multi-homed hosts engage in unexpected interactions. For now, we will distinguish hosts from routers, and assume that hosts do not perform the router's function of transferring packets from one network to another.

8.3 Direct And Indirect Delivery

Loosely speaking, we can divide routing into two forms: *direct delivery* and *indirect delivery*. Direct delivery, the transmission of a datagram from one machine across a single physical network directly to another, is the basis on which all internet communication rests. Two machines can engage in direct delivery only if they both attach directly to the same underlying physical transmission system (e.g., a single Ethernet). *Indirect delivery* occurs when the destination is not on a directly attached network, forcing the sender to pass the datagram to a router for delivery.

8.3.1 Datagram Delivery Over A Single Network

We know that one machine on a given physical network can send a physical frame directly to another machine on the same network. To transfer an IP datagram, the sender encapsulates the datagram in a physical frame, maps the destination IP address into a physical address, and uses the network hardware to deliver it. Chapter 5 presented two possible mechanisms for address resolution, including using the ARP protocol for dynamic address binding on Ethernet-like networks. Chapter 7 discussed datagram encapsulation. Thus, we have reviewed all the pieces needed to understand direct delivery. To summarize:

> *Transmission of an IP datagram between two machines on a single physical network does not involve routers. The sender encapsulates the datagram in a physical frame, binds the destination IP address to a physical hardware address, and sends the resulting frame directly to the destination.*

How does the sender know whether the destination lies on a directly connected network? The test is straightforward. We know that IP addresses are divided into a network-specific prefix and a host-specific suffix. To see if a destination lies on one of the directly connected networks, the sender extracts the network portion of the destination IP address and compares it to the network portion of its own IP address(es). A match means the datagram can be sent directly. Here we see one of the advantages of the Internet address scheme, namely:

Because the internet addresses of all machines on a single network include a common network prefix and extracting that prefix requires only a few machine instructions, testing whether a machine can be reached directly is extremely efficient.

From an internet perspective, it is easiest to think of direct delivery as the final step in any datagram transmission, even if the datagram traverses many networks and intermediate routers. The final router along the path between the datagram source and its destination will connect directly to the same physical network as the destination. Thus, the final router will deliver the datagram using direct delivery. We can think of direct delivery between the source and destination as a special case of general purpose routing – in a direct route the datagram does not happen to pass through any intervening routers.

8.3.2 Indirect Delivery

Indirect delivery is more difficult than direct delivery because the sender must identify a router to which the datagram can be sent. The router must then forward the datagram on toward its destination network.

To visualize how indirect routing works, imagine a large internet with many networks interconnected by routers but with only two hosts at the far ends. When one host wants to send to the other, it encapsulates the datagram and sends it to the nearest router. We know that the host can reach a router because all physical networks are interconnected, so there must be a router attached to each network. Thus, the originating host can reach a router using a single physical network. Once the frame reaches the router, software extracts the encapsulated datagram, and the IP software selects the next router along the path towards the destination. The datagram is again placed in a frame and sent over the next physical network to a second router, and so on, until it can be delivered directly. These ideas can be summarized:

Routers in a TCP/IP internet form a cooperative, interconnected structure. Datagrams pass from router to router until they reach a router that can deliver the datagram directly.

How can a router know where to send each datagram? How can a host know which router to use for a given destination? The two questions are related because they both involve IP routing. We will answer them in two stages, considering the basic table-driven routing algorithm in this chapter and postponing a discussion of how routers learn new routes until later.

8.4 Table-Driven IP Routing

The usual IP routing algorithm employs an *Internet routing table* (sometimes called an *IP routing table*) on each machine that stores information about possible destinations and how to reach them. Because both hosts and routers route datagrams, both have IP routing tables. Whenever the IP routing software in a host or router needs to transmit a datagram, it consults the routing table to decide where to send the datagram.

What information should be kept in routing tables? If every routing table contained information about every possible destination address, it would be impossible to keep the tables current. Furthermore, because the number of possible destinations is large, machines would have insufficient space to store the information.

Conceptually, we would like to use the principle of information hiding and allow machines to make routing decisions with minimal information. For example, we would like to isolate information about specific hosts to the local environment in which they exist and arrange for machines that are far away to route packets to them without knowing such details. Fortunately, the IP address scheme helps achieve this goal. Recall that IP addresses are assigned to make all machines connected to a given physical network share a common prefix (the network portion of the address). We have already seen that such an assignment makes the test for direct delivery efficient. It also means that routing tables only need to contain network prefixes and not full IP addresses.

8.5 Next-Hop Routing

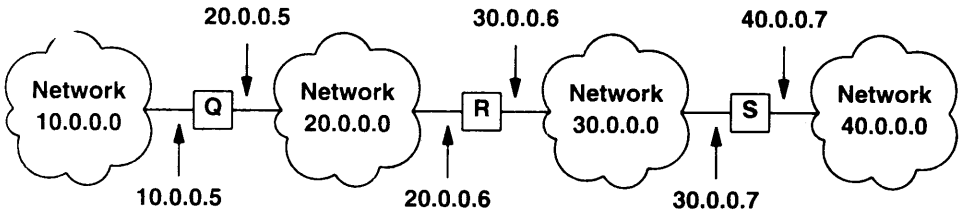
Using the network portion of a destination address instead of the complete host address makes routing efficient and keeps routing tables small. More important, it helps hide information, keeping the details of specific hosts confined to the local environment in which those hosts operate. Typically, a routing table contains pairs (N, R), where N is the IP address of a destination *network*, and R is the IP address of the "next" router along the path to network N. Router R is called the *next hop*, and the idea of using a routing table to store a next hop for each destination is called *next-hop routing*. Thus, the routing table in a router R only specifies one step along the path from R to a destination network – the router does not know the complete path to a destination.

It is important to understand that each entry in a routing table points to a router that can be reached across a single network. That is, all routers listed in machine M's routing table must lie on networks to which M connects directly. When a datagram is ready to leave M, IP software locates the destination IP address and extracts the network portion. M then uses the network portion to make a routing decision, selecting a router that can be reached directly.

In practice, we apply the principle of information hiding to hosts as well. We insist that although hosts have IP routing tables, they must keep minimal information in their tables. The idea is to force hosts to rely on routers for most routing.

Figure 8.2 shows a concrete example that helps explain routing tables. The example internet consists of four networks connected by three routers. In the figure, the rout-

ing table gives the routes that router *R* uses. Because *R* connects directly to networks 20.0.0.0 and 30.0.0.0, it can use direct delivery to send to a host on either of those networks (possibly using ARP to find physical addresses). Given a datagram destined for a host on network 40.0.0.0, *R* routes it to the address of router *S*, 30.0.0.7. *S* will then deliver the datagram directly. *R* can reach address 30.0.0.7 because both *R* and *S* attach directly to network 30.0.0.0.



(a)

TO REACH HOSTS ON NETWORK	ROUTE TO THIS ADDRESS
20.0.0.0	DELIVER DIRECTLY
30.0.0.0	DELIVER DIRECTLY
10.0.0.0	20.0.0.5
40.0.0.0	30.0.0.7

(b)

Figure 8.2 (a) An example internet with 4 networks and 3 routers, and (b) the routing table in *R*.

As Figure 8.2 demonstrates, the size of the routing table depends on the number of networks in the internet; it only grows when new networks are added. However, the table size and contents are independent of the number of individual hosts connected to the networks. We can summarize the underlying principle:

> *To hide information, keep routing tables small, and make routing decisions efficient, IP routing software only keeps information about destination network addresses, not about individual host addresses.*

Choosing routes based on the destination network ID alone has several consequences. First, in most implementations, it means that all traffic destined for a given network takes the same path. As a result, even when multiple paths exist, they may not be used concurrently. Also, all types of traffic follow the same path without regard to the delay or throughput of physical networks. Second, because only the final router along the path attempts to communicate with the destination host, only it can determine if the host exists or is operational. Thus, we need to arrange a way for that router to send reports of delivery problems back to the original source. Third, because each router forwards traffic independently, datagrams traveling from host *A* to host *B* may follow an entirely different path than datagrams traveling from host *B* back to host A. We need to ensure that routers cooperate to guarantee that two-way communication is always possible.

8.6 Default Routes

Another technique used to hide information and keep routing table sizes small consolidates multiple entries into a default case. The idea is to have the IP routing software first look in the routing table for the destination network. If no route appears in the table, the routing routines send the datagram to a *default router*.

Default routing is especially useful when a site has a small set of local addresses and only one connection to the rest of the internet. For example, default routes work well in host computers that attach to a single physical network and reach only one router leading to the remainder of the internet. The routing decision consists of two tests: one for the local net and a default that points to the only router. Even if the site contains a few local networks, the routing is simple because it consists of a few tests for the local networks plus a default for all other destinations.

8.7 Host-Specific Routes

Although we said that all routing is based on networks and not on individual hosts, most IP routing software allows per-host routes to be specified as a special case. Having per-host routes gives the local network administrator more control over network use, permits testing, and can also be used to control access for security purposes. When debugging network connections or routing tables, the ability to specify a special route to one individual machine turns out to be especially useful.

8.8 The IP Routing Algorithm

Taking into account everything we have said, the IP algorithm used to forward datagrams becomes†:

†Chapter 10 discusses a slightly modified algorithm used with classless IP addresses.

```
Algorithm:

RouteDatagram(Datagram, RoutingTable)

Extract destination IP address, D, from the datagram
    and compute the network prefix, N;
if N matches any directly connected network address
    deliver datagram to destination D over that network
    (This involves resolving D to a physical address,
    encapsulating the datagram, and sending the frame.)
else if the table contains a host-specific route for D
    send datagram to next-hop specified in table
else if the table contains a route for network N
    send datagram to next-hop specified in table
else if the table contains a default route
    send datagram to the default router specified in table
else declare a routing error;
```

Figure 8.3 The algorithm IP uses to forward a datagram. Given an IP datagram and a routing table, this algorithm selects the next hop to which the datagram should be sent. All routes must specify a next hop that lies on a directly connected network.

8.9 Routing With IP Addresses

It is important to understand that except for decrementing the time to live and recomputing the checksum, IP routing does not alter the original datagram. In particular, the datagram source and destination addresses remain unaltered; they always specify the IP address of the original source and the IP address of the ultimate destination†. When IP executes the routing algorithm, it selects a new IP address, the IP address of the machine to which the datagram should be sent next. The new address is most likely the address of a router. However, if the datagram can be delivered directly, the new address is the same as the address of the ultimate destination.

We said that the IP address selected by the IP routing algorithm is known as the *next hop* address because it tells where the datagram must be sent next. Where does IP store the next hop address? Not in the datagram; no place is reserved for it. In fact, IP does not "store" the next hop address at all. After executing the routing algorithm, IP passes the datagram and the next hop address to the network interface software responsible for the physical network over which the datagram must be sent. The network in-

†The only exception occurs when the datagram contains a source route option.

terface software binds the next hop address to a physical address, forms a frame using that physical address, places the datagram in the data portion of the frame, and sends the result. After using the next hop address to find a physical address, the network interface software discards the next hop address.

It may seem odd that routing tables store the IP address of a next hop for each destination network when those addresses must be translated into corresponding physical addresses before the datagram can be sent. If we imagine a host sending a sequence of datagrams to the same destination address, the use of IP addresses will appear incredibly inefficient. IP dutifully extracts the destination address in each datagram and uses the routing table to produce a next hop address. It then passes the datagram and next hop address to the network interface, which recomputes the binding to a physical address. If the routing table used physical addresses, the binding between the next hop's IP address and physical address could be performed once, saving unneeded computation.

Why does IP software avoid using physical addresses when storing and computing routes? As Figure 8.4 illustrates, there are two important reasons.

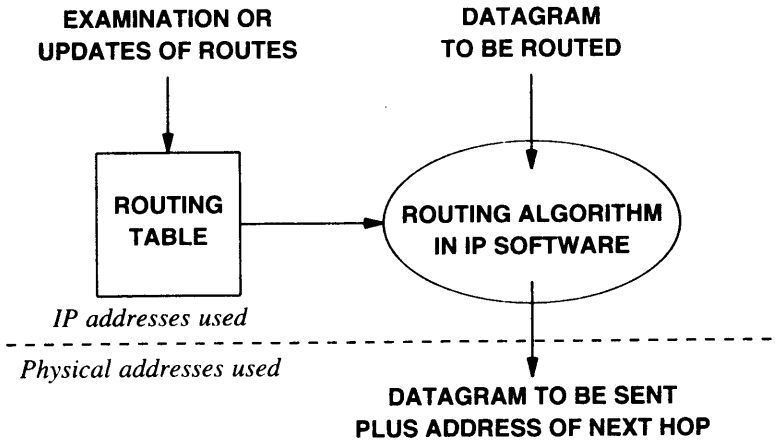


Figure 8.4 IP software and the routing table it uses reside above the address boundary. Using only IP addresses makes routes easy to examine or change and hides the details of physical addresses.

First, the routing table provides an especially clean interface between IP software that routes datagrams and high-level software that manipulates routes. To debug routing problems, network managers often need to examine the routing tables. Using only IP addresses in the routing table makes it easy for managers to understand and to determine whether software has updated the routes correctly. Second, the whole point of the Internet Protocol is to build an abstraction that hides the details of underlying networks.

Figure 8.4 shows the *address boundary*, the important conceptual division between low-level software that understands physical addresses and internet software that only uses high-level addresses. Above this boundary, all software can be written to communicate using internet addresses; knowledge of physical addresses is relegated to a few small, low-level routines. We will see that observing the boundary also helps keep the implementation of remaining TCP/IP protocols easy to understand, test, and modify.

8.10 Handling Incoming Datagrams

So far, we have discussed IP routing by describing how forwarding decisions are made about outgoing packets. It should be clear, however, that IP software must process incoming datagrams as well.

When an IP datagram arrives at a host, the network interface software delivers it to the IP module for processing. If the datagram's destination address matches the host's IP address, IP software on the host accepts the datagram and passes it to the appropriate higher-level protocol software for further processing. If the destination IP address does not match, a host is required to discard the datagram (i.e., hosts are forbidden from attempting to forward datagrams that are accidentally routed to the wrong machine).

Unlike hosts, routers perform forwarding. When an IP datagram arrives at a router, it is delivered to the IP software. Again, two cases arise: the datagram could have reached its final destination, or it may need to travel further. As with hosts, if the datagram destination IP address matches the router's own IP address, the IP software passes the datagram to higher-level protocol software for processing†. If the datagram has not reached its final destination, IP routes the datagram using the standard algorithm and the information in the local routing table.

Determining whether an IP datagram has reached its final destination is not quite as trivial as it seems. Remember that even a host may have multiple physical connections, each with its own IP address. When an IP datagram arrives, the machine must compare the destination internet address to the IP address for each of its network connections. If any match, it keeps the datagram and processes it. A machine must also accept datagrams that were broadcast on the physical network if their destination IP address is the limited IP broadcast address or the directed IP broadcast address for that network. As we will see in Chapters 10 and 17, classless, subnet, and multicast addresses make address recognition even more complex. In any case, if the address does not match any of the local machine's addresses, IP decrements the time-to-live field in the datagram header, discarding the datagram if the count reaches zero, or computing a new checksum and routing the datagram if the count remains positive.

Should every machine forward the IP datagrams it receives? Obviously, a router must forward incoming datagrams because that is its main function. We have also said that some multi-homed hosts act as routers even though they are really general purpose computing systems. While using a host as a router is not usually a good idea, if one chooses to use that arrangement, the host must be configured to route datagrams just as a router does. But what about other hosts, those that are not intended to be routers?

†Usually, the only datagrams destined for a router are those used to test connectivity or those that carry router management commands, but a router must also keep a copy of datagrams that are broadcast on the network.

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.

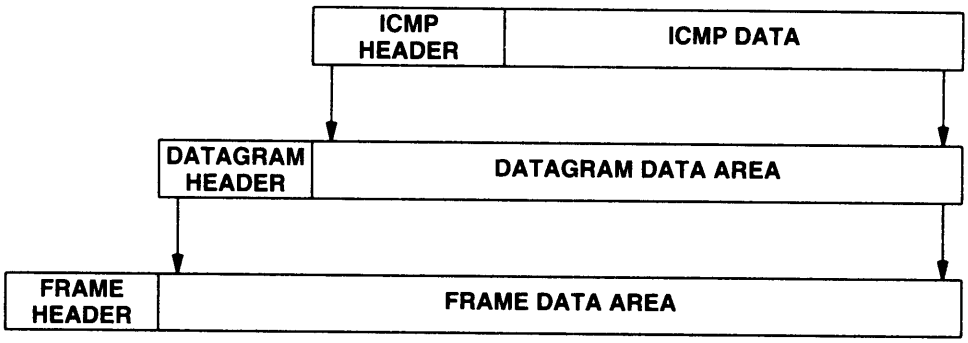


Figure 9.1 Two levels of ICMP encapsulation. The ICMP message is encapsulated in an IP datagram, which is further encapsulated in a frame for transmission. To identify ICMP, the datagram protocol field contains the value *1*.

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: