Chapter 2

The Internet Protocol

This chapter describes the Internet Protocol (IP), which is the fundamental building block for all control and data exchanges across and within the Internet. There is a chicken-and-egg definition that describes the Internet as the collection of all networked computers that interoperate using IP, and IP as the protocol that facilitates communication between computers within the Internet. IP and the Internet are so closely woven that the ordering of the definition doesn't really matter, but it is indubitable that the Internet is deeply dependent on the definition and function of IP. IP version four (IPv4) is the most common version of the protocol in use, and this chapter focuses on that version.

This chapter includes a brief section that examines the motivation for IP before moving on to examine the format of IPv4 messages, the meanings of the standard fields that they carry, and the checksum algorithm used to safeguard individual messages against accidental corruption. The way data is packaged into individual messages is explained.

Fundamental to the operation of IP are the addresses that are used to identify the senders and receivers of individual messages. IPv4 addressing is the subject of Section 2.3. There is information on how the address space is subdivided for ease of management and routing.

Section 2.4 describes the basic operation of IP. It shows how messages are delivered based on the destination IP addresses and introduces three protocols designed to help discover and manage IP addresses within a network: the Address Resolution Protocol (ARP), the Bootstrap Protocol (BOOTP), and the Dynamic Host Configuration Protocol (DHCP).

IP also defines some optional fields that may be included in IP messages as needed. These fields manage a set of advanced features that are not used in standard operation but may be added to data flows to enhance the function provided to the applications that are using IP to transfer their data. Section 2.5 describes some of the common IP options, including those to manage and control the route taken by an IP message as it traverses the network.

Finally, there is a section explaining the Internet Control Message Protocol (ICMP). ICMP is a separate protocol and is actually carried as a payload by IP. However, ICMP is fundamental to the operation of IP networks and is so closely

related to IP that it is not possible to operate hosts within an IP network without supporting ICMP.

2.1 Choosing to Use IP

It is pretty central to the success of this book that IP is chosen as the core network protocol for your network. Everything else in this book depends on that choice since all of the many protocols described utilize IP to carry their messages, use IP addresses to identify nodes and links, and assume that data is being carried using IP.

Fortunately, IP has already been chosen as the network protocol for the Internet and we don't have to decide for ourselves whether to use IP. If we want to play in the Internet we must use IP. It is, however, worth examining some of the motivation behind IP to discover why it was developed, what problems it solves, and why it continues to be central to the Internet.

2.1.1 Connecting Across Network Types

Hosts on a network use the data-link layer to move data between them in *frames* across the physical network. Each type of physical network has a different data-link layer protocol (Ethernet, Token Ring, ATM, etc.) responsible for delivering the data. Each of these protocols has different formats for its frames and addresses (as described in the previous chapter), and these formats are not interchangeable. That is, you cannot pick up a frame from a Token Ring network and drop it onto an Ethernet—it doesn't belong there and will not work.

As long as all hosts are attached to the same physical medium there is no issue, but as we begin to construct networks from heterogeneous physical network types we must install special points of connection that convert from one data-link protocol to another. There are some issues that immediately raise their heads when we try to do this.

First, the addressing schemes on the two connected networks are different. When a frame with an Ethernet address is moved to a Token Ring the addresses must be mapped. As the number of network types increases, this addressing mapping function gets ever more complicated. A simpler solution is to provide an overarching addressing scheme that requires every node to implement just one address mapping function between the local physical addressing and the systemwide IP addressing.

The next problem is that the different networks do not all support the same size data frame. To demonstrate this in an extreme case, if a Token Ring network sends frames to an X.25 network the interconnecting node may receive a frame of 17,756 bytes and not be able to present it to the X.25 network because that can only support frames of 512 bytes. What is needed is a higher-level protocol that can be invoked to fragment the data into smaller pieces.

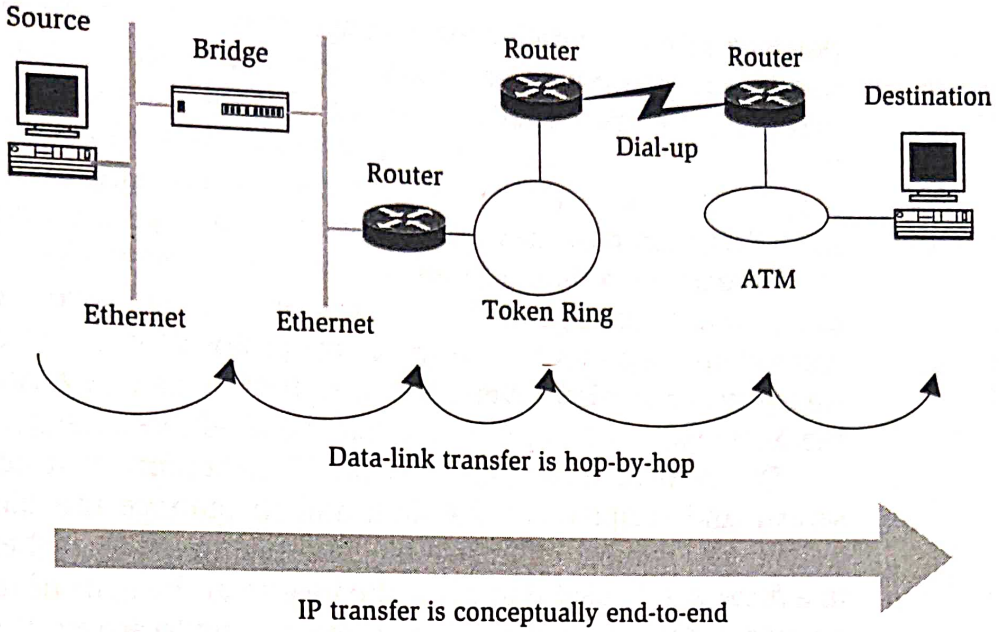

Figure 2.1 IP provides a uniform network layer protocol to operate over any collection of data-link protocols and physical links.

Many physical link types and data-link layer protocols are fundamentally unreliable—that is, they may drop frames without warning. Some are capable of detecting and reporting errors, but few can recover from such problems and retransmit the lost data such that the higher-layer protocols (that is, the transport and application protocols that use the links) are protected from knowledge of the problems. This means that a protocol must be run at a higher level to reliably detect and report problems. IP does this, but does not attempt to recover from problems—this function is devolved further up the stack to become the responsibility of transport or application layer protocols.

Ultimately, we need a single protocol that spans multiple physical network types to deliver data in a uniform way for the higher-level protocols. The path taken by the data is not important to the higher-level protocols (although they may wish to control the path in some way) and individual pieces of data are free to find their different ways across the network as resources are available for their transmission.

IP provides all of these functions, as shown in Figure 2.1.

2.2 IPv4

The Internet Protocol went through several revisions before it stabilized at version four. IPv4 is now ubiquitous, although it should be noted that version six of the protocol (IPv6; see Chapter 4) is gaining support in certain quarters.

IP is a protocol for universal data delivery across all network types. Data is packaged into *datagrams* that comprise some control information and the

payload data to be delivered. (Datagram is a nice word invented to convey that this is a record containing some data, and with overtones from telegram and aerogram it gives a good impression that the data is being sent from one place to another.) Datagrams are *connectionless* because each is sent on its own and may find its own way across the network independent of the other datagrams. Each datagram may take a different route through the network.

Note a very subtle difference between a packet and a datagram: A packet is any protocol message at the network or transport layer, and a datagram is any connectionless protocol message at any protocol layer, but usually at the network, transport, or session layers. Thus, in IPv4, which is a connectionless protocol, the words *packet* and *datagram* may be used interchangeably.

The control information in an IP datagram is necessary to identify the sender and recipient of the data and to manage the datagram while it is in transit. The control information is grouped together at the start of the datagram in a *header*. It is useful to place the header at the start of the datagram to enable a computer to access it easily without having to search through the entire datagram. All of the datagram headers are formatted in the same way so that a program processing the datagrams can access the information it needs with the minimum of fuss.

The remainder of this section is dedicated to a description of the IPv4 header and to details of how IPv4 carries data within datagrams.

> **data** \'dā-tə, 'dat-, 'dät-\ *n pl but sing or pl in constr* [pl. of *datum* L, fr. neut. of *datus*]: factual information (as measurement or statistics) used as a basis for reasoning, discussion, or calculation.
> **-gram** \ˌgram\ *n comb form* [L. -gramma, fr. Gk., fr. *gramma*]: drawing: writing: record <chronogram> <telegram> <aerogram>

2.2.1 IP Datagram Formats

Each IP datagram begins with a common header. This is shown as a byte-by-byte, bit-by-bit structure in Figure 2.2. The first nibble shows the protocol version (version four indicates IPv4; a value of 6 would be used for IPv6—see Chapter 4). The next nibble gives the length of the header, and because there are only 4 bits available in the length field and we need to be able to have a header length of more than 15, the length is counted in units of 4-byte words. The length field usually contains the value 5 because the count includes all bytes of the header (that is, 20), but may be greater if IP options are included in the header (see Section 2.5). The Type of Service byte is used to classify the datagram for prioritization, use of network resources, and routing within the network; this important function is described further in Section 2.4.4. Next comes a 2-byte field that gives the length of the entire datagram. The length of the data carried by the datagram can be calculated by subtracting the header length from the

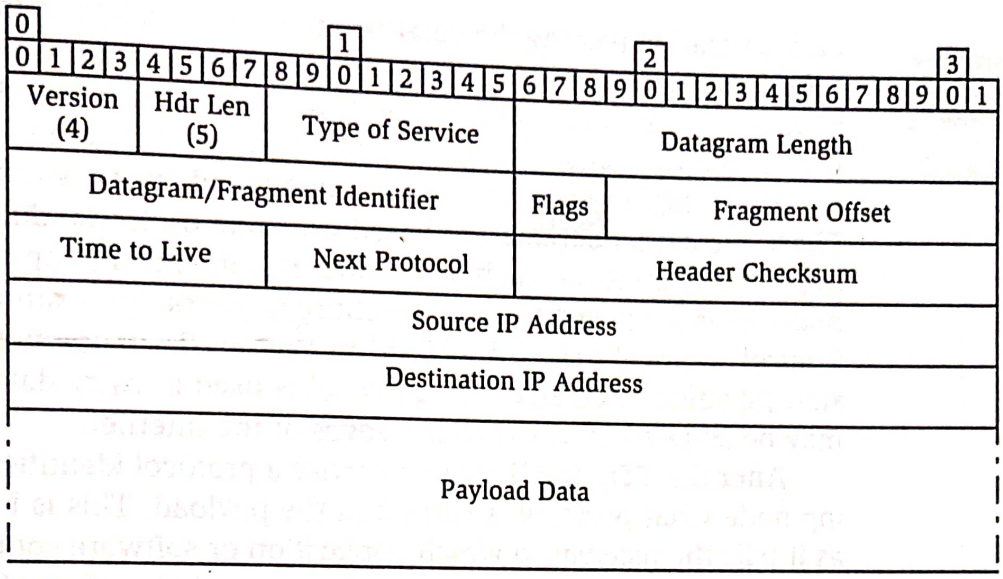

Figure 2.2 The IP datagram header.

length of the entire datagram. Obviously, this places a limit on the amount of data carried by one IP datagram to 65,535 bytes less the size of the header.

The next three fields are concerned with how a datagram is handled if, part way across the network, it reaches a hop that must break the datagram up into smaller pieces to forward it. This process is called *fragmentation* and is covered in the next section. The fields give an identifier for the original datagram so that all fragments can be grouped together, flags for the control of the fragmentation process, and an offset within the original datagram of the start of the fragment.

The next field is called Time to Live (TTL). It is used to prevent datagrams from being forwarded around the network indefinitely through a process called *looping*, as shown in Figure 2.3. The original intent of the TTL was to measure the number of seconds that a datagram was allowed to live in the network, but this quickly became impractical because each node typically forwards a datagram within 1 second of receiving it and there is no way to measure how long a packet took to be transmitted between nodes. So, instead, the TTL is used as a count of the number of hops the datagram may traverse before it is timed out. Each node decrements the TTL before it forwards the packet and, to quote RFC 791, "If the time to live reaches zero before the Internet datagram reaches its destination, the Internet datagram is destroyed." This is generally interpreted as meaning that a datagram may not be transmitted with a TTL of zero. Implementations vary as to whether they decrement the TTL for the first hop from the source. This is important since it controls the meaning of the value 1 in the TTL when a datagram is created; it may mean that the packet may traverse just one hop, or it may mean that the packet is only available for local delivery to another application on the source node (see Section 2.4 for a description of local delivery). In Figure 2.3 a datagram is created with a TTL of 7 and is forwarded through node A to node B. Node B is misconfigured

and, instead of passing the datagram to the destination, it forwards it to node C—a forwarding loop exists. When the datagram arrives at node C for the second time, node C prepares to forward it to node A. But when it decrements the TTL it sees that the value has gone to zero so it discards the packet.

Many higher-layer protocols recommend initial values for the TTL field. These recommendations are based on some understanding of the scope of the higher-layer protocol, such as whether it is intended to operate between adjacent nodes or is supposed to span the entire network. For example, the Transmission Control Protocol (TCP) described in Section 7.3 recommends a relatively high starting value of 60 since the protocol is used to carry data between nodes that may be situated on the extreme edges of the Internet.

After the TTL, the IP header carries a protocol identifier that tells the receiving node what protocol is carried in the payload. This is important information as it tells the receiver to which application or software component the datagram should be delivered. Table 2.1 lists the protocol identifiers of some of the common protocols. Note that there are only 256 values that can be defined here and, although obviously when IP was invented this was thought to be plenty, the list of protocols defined and maintained by the Internet Assigned Numbers Authority

Table 2.1 Some Common IP Payload Protocols and Their Identifiers

Protocol Number	*Protocol*	*RFC*	*Reference*
1	Internet Control Message Protocol (ICMP)	RFC 792	2.6
2	Internet Group Message Protocol (IGMP)	RFC 1112	3.3
4	IP encapsulated within IP	RFC 2003	15.1.3
6	Transmission Control Protocol (TCP)	RFC 793	7.3
17	User Datagram Protocol (UDP)	RFC 768	7.2
46	Resource Reservation Protocol (RSVP)	RFC 2205	6.4
47	General Routing Encapsulation (GRE)	RFC 2784	15.1.2
89	Open Shortest Path First (OSPF)	RFC 2328	5.5
124	OSI IS-IS Intradomain Routing Protocol (IS-IS)	RFC 1142	5.6
132	Stream Control Transmission Protocol (SCTP)	RFC 2960	7.4

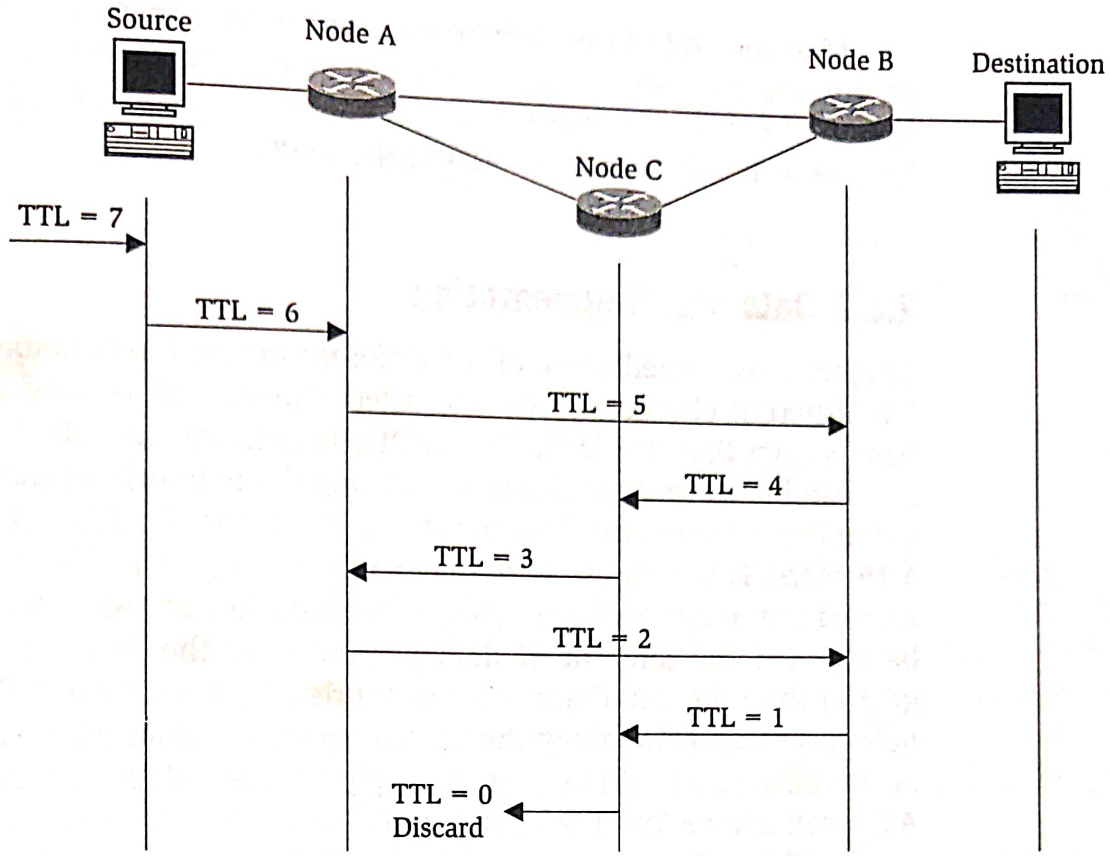

Figure 2.3 The Time to Live value controls the life of an IP datagram within a network and prevents packets from looping forever.

(IANA) at their Web site (http://www.iana.org/assignments/protocol-numbers) has grown to 135, giving rise to serious concerns that they may soon run out of identifiers for protocols that can be carried by IP. The current solution to this is to make new protocols use a transport protocol such as UDP (see Chapter 7), which has the facility to carry far more client protocols.

After the protocol identifier comes a 2-byte field that carries the Header Checksum used to verify that the whole of the header has been received without any accidental corruption. The checksum processing is described in greater length in Section 2.2.3.

Finally, within the standard IP header come two address fields to identify the sender of the message and its intended destination. IP addresses are 4-byte quantities that are usually broken out and expressed as 4-decimal digits with dots between them. Thus, the IP address field carrying the hexadecimal number 0xac181004 would be represented as 172.24.16.4. The choice of addresses for nodes within the network is not a random free-for-all because structure is needed both to ensure that no two nodes have the same address and to enable the address to be used for directing the datagram as it passes through the network. Section 2.3 expands upon IP addressing, and the whole of Chapter 5 is dedicated to routing protocols that make it possible for transit nodes to work out which way to forward datagrams based on the destination address they carry.

After the end of the 20-byte standard header there may be some IP options. IP options are used to add selective control to datagrams and are optional. Section 2.5 describes some of the common IP options. Finally, there is the payload data that IP is carrying across the network.

2.2.2 Data and Fragmentation

Chapter 1 examined some of the different network technologies that exist. Each has different characteristics that affect the way IP is used. For example, each has its own Protocol Data Unit (PDU) maximum size (the largest block of data that can be transmitted in one shot using the network technology). For X.25 this value is 576 bytes, for Ethernet it is 1,500 bytes, for FDDI it is 4,352 bytes, and a 16 Mbps Token Ring can manage a PDU of up to 17,756 bytes. IP itself allows a datagram of up to 65,535 bytes, as we have already seen, but some action has to be taken if the amount of data presented to the IP layer for transmission is greater than the maximum PDU supported by the network. Some network technologies support breaking the packet up into smaller pieces and reassembling it at the destination. ATM is an example of this, which is a good thing since the ATM cell allows for only 48 bytes of data—if IP datagrams had to be made to fit inside ATM cells there would only be 28 bytes of data in each datagram!

Other technologies, however, cannot segment and reassemble data, so there is a need for IP to limit the size of packets that are presented to the network. Figure 2.4 shows how data presented to IP by the application may be

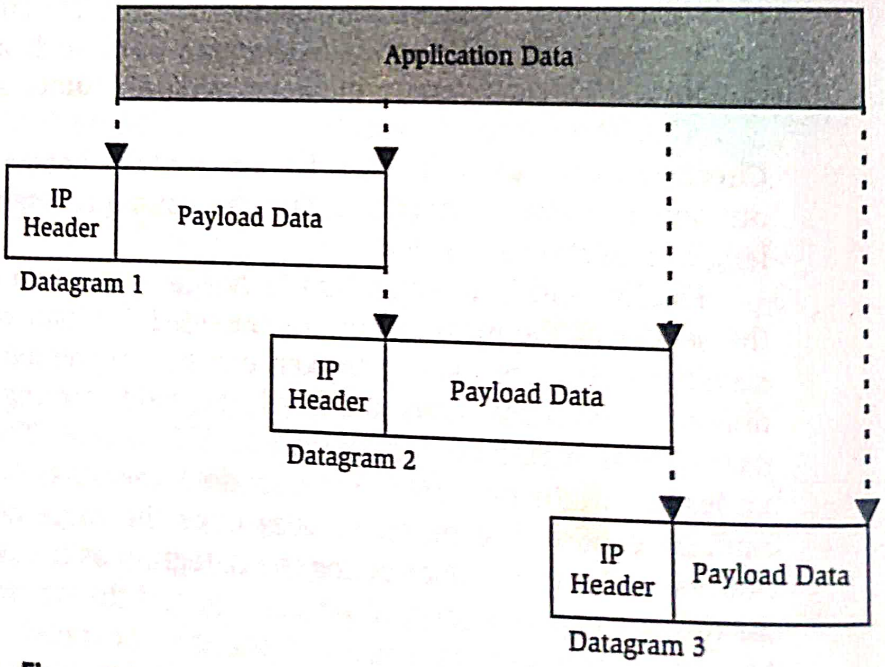

Figure 2.4 Application data may be segmented into separate datagrams.

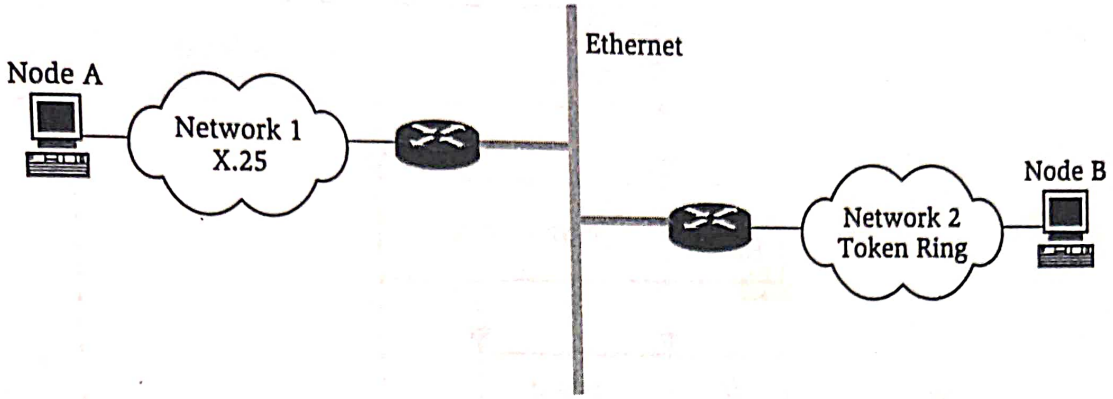

Figure 2.5 The need for fragmentation.

chopped up and transmitted in a series of IP datagrams. Each datagram is assigned a unique Datagram Identifier which is placed in the IP header. This field might have been used to allow the datagrams to be reordered or to detect lost datagrams, and so it would not be unreasonable to make the value increase by one for each datagram, but this is not a requirement in RFC 791 and must not be assumed. The real purpose of this field comes into play if a datagram must be fragmented part-way across the network.

Some applications may not be willing to have their data chopped up into separate datagrams—it may cause them a problem if they have to wait for all of the datagrams in a sequence to arrive before they can start to process the first one. This may be more of an issue for control protocols than it is for data transfer, since a control message must be processed in its entirety but data can simply be written to the output device as it arrives. Applications that don't want their messages broken up need to agree with the IP layer what the maximum PDU is for the attached network, and must present the data in lumps that are small enough to be carried within one IP datagram (taking into account the bytes needed for the IP header).

Consider the network shown in Figure 2.5. Here Node A is attached to an X.25 network where the maximum PDU is 576 bytes. When Node A sends packets to Node B it makes sure that no packet is larger than 512 bytes. As the packet progresses, it traverses the X.25 network and encounters an Ethernet. Since the maximum PDU for the Ethernet is 1,500 bytes, the IP datagrams can simply be forwarded. When they transition to the Token Ring where the maximum PDU is 17,756 bytes, the packets can continue to be forwarded to the destination.

However, suppose Node B wishes to send a reply to Node A. Since Node B is attached to the Token Ring it may prepare IP datagrams up to 17,756 bytes long. These are forwarded toward Node A until they encounter the Ethernet, where they are too large. For the datagram to be forwarded, it must be fragmented into pieces that are no larger than 1,500 bytes. Again, when the fragments reach the X.25 network they must be fragmented still further to be carried across the network. The process of fragmentation is illustrated in Figure 2.6.

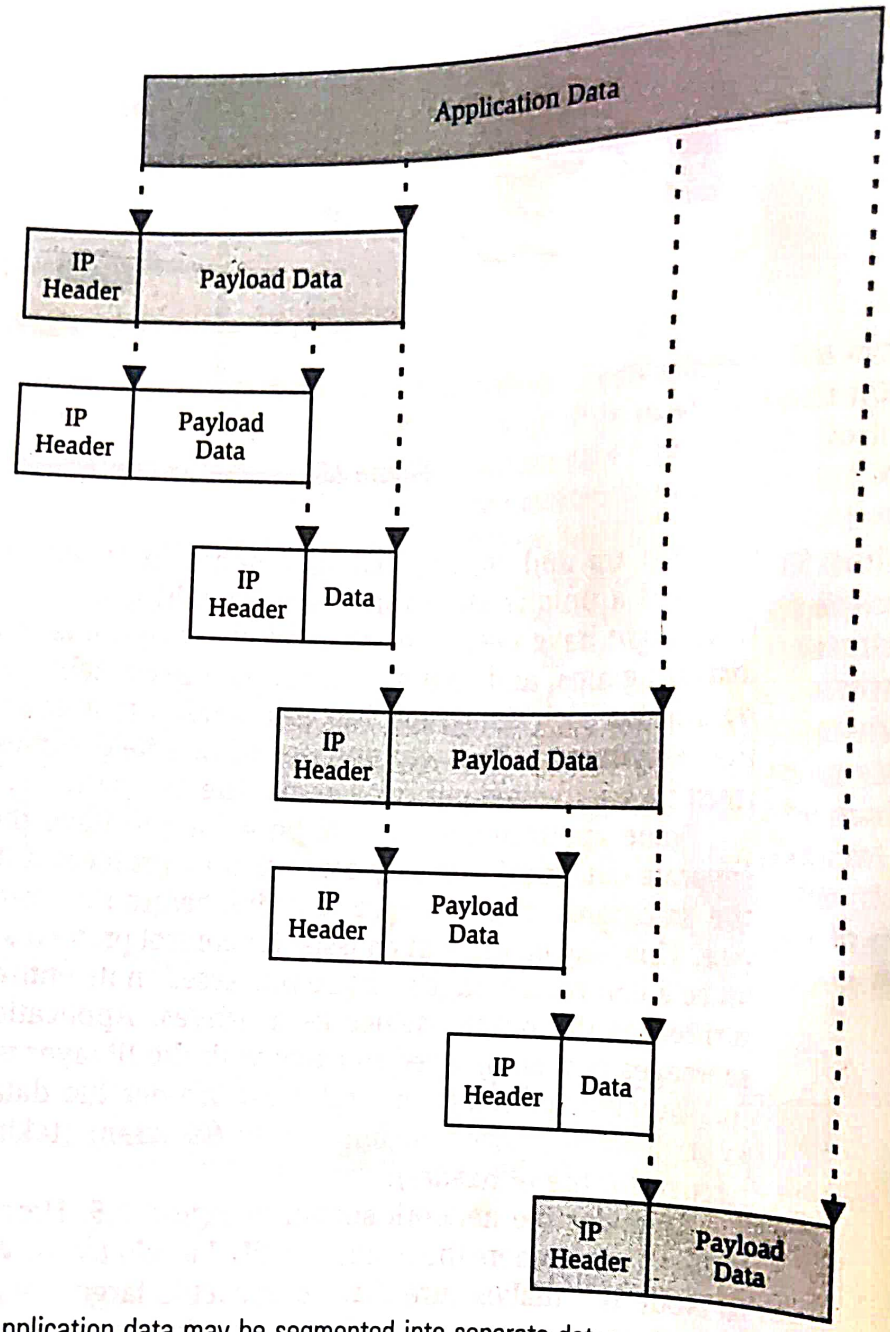

Figure 2.6 Application data may be segmented into separate datagrams, each of which may later require further fragmentation.

When data is presented to IP by an application it is broken up to be carried in separate datagrams, as already described. When the datagrams reach a network where the maximum PDU is smaller than the datagram size, they are fragmented into smaller datagrams. The IP header of each of the fragments is copied from the original datagram so that the TTL, source, and destination are identical. The datagram identifier of each of the fragments is the same so that all fragments of the original datagram can be easily identified

Fragment reassembly is necessary at the destination. Each of the fragments must be collected and assembled into a single data stream to be passed to the application as if the whole original datagram had been received intact. This should simply be a matter of concatenating the data from the fragments, but there are two issues caused by the fact that datagrams might arrive out of order because of different paths or processing through the network. We need to know where each fragment fits in the original datagram. This is achieved by using the Fragment Offset field in the IP header. Note that the offset field is only 13 bits long, so it can't be used as a simple offset count since the datagram itself may be 2^{16} bytes long. This is handled by insisting that fragmentation may only occur on 8-byte boundaries. There are 2^{13} (that is 8,191) possible values here, and $2^{13} * 8 = 2^{16}$ so all of the data is covered. If fragmentation of the data into blocks of less than 8 bytes were required, performance would be so bad that we might as well give up anyway.

So, as fragments arrive they can be ordered and the data can be reassembled. Implementations typically run a timer when the first fragment in a series arrives so that sequences that never complete (because a datagram was lost in the network) are not stored forever. A reasonable approach would be to use the remaining TTL measured in seconds to define the lifetime of the fragments pending reassembly, giving a maximum life of up to 4 ½ minutes, but many implementations don't use this and actually run timers of around 15 seconds, as recommended by RFC 791. Many applications do not support receipt of out-of-order fragments and will reject the whole datagram if this happens, but they still use the fragment Offset and the Datagram Length to reassemble fragments and to detect when fragments are out of order. Failed reassembly results in discarding of the entire original datagram.

The second issue is determining the end of the original datagram. Initially, this was obvious because the Datagram Length less the Header Length indicated the size of the Payload Data, but each fragment must carry its own length. When the fragments are reassembled there is no way of knowing when to stop. We could wait for the receipt of a fragment with a different Datagram Identifier, but this would not help us if a fragment was lost or arrived out of order. The problem is solved by using the third bit of the Flag field to indicate when there are more fragments—the More Fragments (MF) bit is set to 1 whenever there is another fragment coming and to zero on the last fragment. Note that the rule for fragmenting existing fragments is that if the original datagram has the MF bit set to 1, then all resultant fragments must also have this bit set to 1. If the original fragment has the bit set to zero, then all fragments except the last must have the MF bit set to 1 (the last must retain the zero value).

Unfragmented datagrams carry a Fragment Offset of zero and the MF bit set to zero. Note that IP uses lazy reassembly of fragments. That is, reassembly is only done at the destination node and not at transit nodes within the network even if the datagrams are passing from a network with small PDUs to one that can handle larger PDUs. This is a pragmatic reduction in processing since it is

unclear to a transit node that further fragmentation will not be needed further along the path. It also helps to reduce the buffer space that would be needed on transit nodes to store and reassemble fragments.

An application may want to prevent fragmentation within the network. This is particularly useful if it is known that the receiving node does not have the ability or resources to handle reassembly, and is achieved by setting the second bit of the Flags field in the IP header (that first bit is reserved and should be set to zero, and the third bit is the MF bit already described). The Don't Fragment (DF) bit is zero by default, allowing fragmentation, and is set to 1 to prevent fragmentation. A transit node that receives a datagram with the DF bit set to 1 must not fragment the datagram, and may choose a route that does not require fragmentation of the packet or must otherwise discard any datagram that cannot be forwarded because of its size.

Alternatively, fragmentation can be avoided by discovering the *Maximum Transmission Unit* (MTU) between the source and destination. This is the lowest maximum PDU on all the links between the source and destination. Some higher-level protocols attempt to discover this value through information exchanges between the nodes along the route. They then use this information to choose specific routes or to present the data to IP in small enough chunks that will never need to be fragmented.

2.2.3 Choosing to Detect Errors

Whenever messages are transmitted between computers they are at risk of corruption. Electrical systems, in particular, are subject to bursts of static that may alter the data before it reaches its destination. If the distortion is large, the receiver will not be able to understand the message and will discard it, but the risk is that the corruption is only small so that the message is misunderstood but treated as legal. IP needs a way to protect itself against corrupt messages so that they may be discarded or responded to with an error message.

There are several options to safeguard messages sent between computers. The first is to place *guard bytes* within the message. These bytes contain a well-known pattern and can be checked by the receiver. This approach is fine for catching major corruptions, but works only if the guard bytes themselves are damaged—if the error is in any other part of the message, no problem will be detected.

A better approach is to perform some form of processing on all of the bytes in the message and include the answer in the transmitted message. The receiver can perform the same processing and verify that no corruption has occurred. Such processing needs to be low cost if it is not to affect the throughput of data. The simplest method is to sum the values of the bytes, discarding overflow, and to transmit the total to the receiver, which can repeat the sum. This technique is vulnerable on two counts. First, it is sensitive only within the size of the field used to carry the sum. That is, if a 1-byte field is used, there are only

Table 2.2 To Perform the IP Checksum, a Stream of Bytes Is Broken into 16-Bit Integers

Byte Stream	*Sequence of Integers*
A, B, C, D, . . . , W, X, Y, Z	[A,B] + [C,D] + · · · + [W,X] + [Y, Z]
A, B, C, D, . . . , W, X, Y	[A,B] + [C,D] + · · · + [W,X] + [Y, 0]

256 possible values of the sum and so there is a relatively high chance that data corruptions will result in the same sum being generated. Second, such a simple sum is also exposed to self-canceling errors, such as, a corruption that adds one to the value of one byte and subtracts one from the value of another byte.

A slight improvement to the simple sum is achieved using the *one's complement checksum* that has been chosen as the standard for IP. In this case, overflows (that is, results that exceed the size of the field used to carry the sum within the protocol) are "wrapped" and cause one to be added to the total.

In IP, the checksum is applied to the IP header only. The transport and application protocols are responsible for protecting their own data. This has the benefit of reducing the amount of checksum processing that must be done at each step through the network. It places the responsibility for detection and handling of errors firmly with the protocol layer that is directly handling a specific part of the data. If that layer does not need or choose to use any error detection, then no additional processing is performed by the lower layers.

The standard IP checksum is performed on a stream of bytes by breaking them up into 16-bit integers. That is, the pair of bytes (a, b) is converted into the integer 256*a+b, which is represented by the notation [a, b]. Table 2.2 shows how byte streams are broken into integers depending on whether there is an even or an odd number of bytes.

These integers are simply added together using 1's complement addition to form a 16-bit answer. The answer is logically reversed (that is, the 1's complement of the sum is taken), and this is the checksum. One's complement addition is the process of adding two numbers together and adding one to the result if there is an overflow (that is, if the sum is greater than 65,535). The consequence of this is that if the checksum value is added using 1's complement addition to the 1's complement sum of the other integers, the answer is all ones (0xffff). Figure 2.7 works a trivial example to show how this operates.

Now, for the checksum to be of any use, it has to be transmitted with the data. There is a field in the IP header to carry the checksum. When the checksum is computed on the header, this field is set to zero so that it has no effect on the computed checksum. The computed value is inserted into the header just before it is transmitted. When the IP packet is received, the checksum is calculated across the whole header and the result is compared with 0xffff to see whether the header has arrive uncorrupted.

Consider the byte stream (0x91, 0xa3, 0x82, 0x11)

This is treated as two integers 0x91a3 and 0x8211

0x91a3+0x8211=0x113b4

So the one's complement sum is 0x13b4+0x01=0x13b5

Now 0x13b5=0b0001001110110101

So the one's complement of 0x1365, is 0b1110110001001010=0xec4a

Thus the checksum is 0xec4a

See that the one's complement sum of the integers and the checksum is as follows:

0xec4a+0x91a3+0x8211=0x17ded+0x8211
=0x7ded+0x01+0x8211
=0xffff

Figure 2.7 A simple example of the 1's complement checksum and the way it can be checked to show no accidental corruption.

Note that some implementations choose to process a received header by copying out the received checksum, setting the field to zero in the packet, computing the checksum, and comparing the answer with the saved value. There is no difference in the efficacy of this processing.

One of the benefits of this checksum algorithm is that it does not matter where in the sequence the checksum value itself is placed. It doesn't even matter if the checksum field starts on an odd or an even byte boundary. There are even some neat tricks that can be played on machines on which the native integer size is four bytes (32 bits) to optimize checksum calculations for native processing. Essentially, the one's complement sum can be computed using 32-bit integers and the final result is then broken into two 16-bit integers, which are summed.

A further benefit is that it is possible to handle modifications to the IP header without having to recompute the entire checksum. This is particularly useful in IP as the packet is passed from node to node and the TTL field is decremented. It would be relatively expensive to recompute the entire checksum each time, but knowing that the TTL field value has decreased by one and knowing that the field is placed in the high byte of a 16-bit integer, the checksum can be decremented by 0x0100 (taking care of underflow by subtracting 0x0001). Some implementations may prefer to stick to 1's complement addition, in which case the minus 0x0100 is first represented as 0xff00 and then added.

This checksum processing effectively protects the header against most random corruptions. There are some situations that are not caught. For example, if

a bit that was zero in one 16-bit integer is corrupted to one, and the same bit in another 16-bit integer that was one is corrupted to zero, the problem will not be detected. It is statistically relatively unlikely that such an event will occur.

And what happens if an error is detected? Well, IP is built on the assumption that it is an unreliable delivery protocol, and that datagrams may be lost. Bearing this in mind, it is acceptable for a node that detects an error to silently discard a packet. Transit nodes might not detect errors because they can simply use the checksum update rules when modifying the TTL to forward a datagram, so it may be that checksum errors are not noticed until the datagram reaches the egress. But wherever the problem is first noticed it will be helpful to distinguish discarded packets from packet loss, and to categorize the reasons for discarding packets (contrasting checksum errors with TTL expiration). Nodes typically retain counters of received and transmitted packets and also count errors by category, but more useful is to notify the sender of the problem so that it can take precautions or actions to avoid the problem. The Internet Control Message Protocol (ICMP) described in Section 2.6 can be used to return notifications when packets are discarded.

2.3 IPv4 Addressing

Every node in an IP network is assigned one or more IP addresses. These addresses are unique to the node within the context of the network and allow the source and destination of packets to be clearly identified. The destination addresses on packets tell nodes within the network where the packets are headed and enable them to forward the packets toward their destinations.

2.3.1 Address Spaces and Formats

All IPv4 addresses are four bytes long (see Figure 2.2). This means that they are not large enough to hold common data-link layer addresses, which are often six bytes and must be assigned from a different address space. The four bytes of the address are usually presented in a *dotted decimal* notation which is easy for a human operator to read and remember. Table 2.3 illustrates this for a few sample addresses.

Table 2.3 IPv4 Addresses Presented in Dotted Decimal Format

Hexadecimal IP Address	*Dotted Decimal Representation*
0x7f000001	127.0.0.1
0x8a5817bf	138.88.23.191
0xc0a80a64	192.168.10.100

Some structure is applied to IPv4 addresses, as we shall see in subsequent sections, and in that context the bits of the address are all significant, with the leftmost bit carrying the most significance just as it would if the address were a number.

IP addresses are assigned through national registries, each of which has been delegated the responsibility for a subset of all the available addresses by the overseeing body, the Internet Corporation for Assigned Names and Numbers (ICANN). This function is critical to the correct operation of the Internet because if there were two nodes with the same IP address attached to the Internet they would receive each other's datagrams and general chaos would ensue.

To that extent, an IP address is a sufficiently unique identifier to precisely point to a single node. But just as people have aliases and family nicknames, so a single node may have multiple addresses and some of these addresses may be unique only within limited contexts. For example, a node may have an IP address by which it is known across the whole Internet, but use another address within its local network. If the node were to publicize its local address, it would discover that many other nodes in the wider Internet are also using the same alias within their local networks.

The address space 0.0.0.0 to 255.255.255.255 is broken up into bands or *classes* of address. The idea is to designate any network as belonging to a particular class determined by the number of nodes in the network. The network is then allocated an address range from the class and can administratively manage the addresses for its nodes. Table 2.4 shows the five address classes. The table shows the range of addresses from which class ranges will be allocated, an example class range, the number of ranges within the class (that is, the number of networks that can exist within the class), and the number of addresses within a class range (that is, the number of nodes that can participate in a network that belongs to the class). Examination of the first byte of an IP address can tell us to which class it belongs.

Table 2.4 IPv4 Address Classes

Class	*Address Range*	*Example Class Range*	*Networks in Class*	*Hosts in Network*
A	0.0.0.0 to 127.255.255.255	100.0.0.1 to 100.255.255.254	126	16,777,214
B	128.0.0.0 to 191.255.255.255	181.23.0.1 to 181.23.255.254	16,384	65,534
C	192.0.0.0 to 223.255.255.255	192.168.20.1 to 192.168.20.254	2,097,152	254
D	224.0.0.0.0 to 239.255.255.255	Addresses reserved for multicast (see Chapter 3)		
E	240.0.0.0 to 247.255.255.255.255	Reserved for future use		

2.3.2 Broadcast Addresses

Some addresses have special meaning. The addresses 0.0.0.0 and 255.255.255.255 may not be assigned to a host. In fact, within any address class range the equivalent addresses with all zeros or all ones in the bits that are available for modification are not allowed. So, for example, the Class C address range shown in Table 2.4 runs from 192.168.20.1 to 192.168.20.254, excluding 192.168.20.0 and 192.168.20.255. The "dot zero" address is used as a shorthand for the range of addresses within a class so that the Class C address range in Table 2.4 can be expressed as 192.168.20.0.

The addresses ending in 255 are designated as broadcast addresses. The broadcast address for the Class C range 192.168.20.0 is 192.168.20.255. When a packet is sent to a broadcast address it is delivered to every host within the network, that is, every host that belongs to the address class range. The broadcast address is a wild card.

Broadcasts have very specific uses that are advantageous when one host needs to communicate with all other hosts. A particular use can be seen in ICMP in Section 2.6, where a host needs to find any routers on its network and so issues a broadcast query to all stations on the network asking them to reply if they are routers. On the other hand, broadcast traffic must be used with caution because it can easily gum up a network.

2.3.3 Address Masks, Prefixes, and Subnetworks

A useful way to determine whether an address belongs to a particular network is to perform a logical AND with a *netmask*. Consider again the example Class C network from Table 2.4 that uses the addresses in the range 192.168.20.1 to 192.168.20.254. To determine whether an address is a member of this network we simply AND it with the value 255.255.255.0 and compare the answer with 192.168.20.0. So in Figure 2.8, 192.168.20.99 is a member of the network, but 192.169.20.99 is not.

The netmask value chosen is based on the class of the address group, that is, the number of trailing bits that are open for use within the network. In this case, the Class C case, the last eight bits do not factor into the decision as to whether the address belongs to the network and are available for distinguishing the nodes within the network.

With this knowledge, the network address can be represented as 192.168.20.0/24, where the number after the forward slash is a count of the

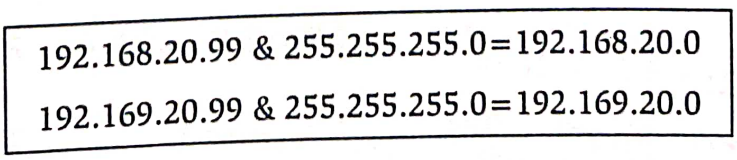

Figure 2.8 Use of the netmask to determine whether an address belongs in a network.

number of bits in the address that define the network. In other words, it is a count of the number of ones set in the netmask. The network is said to have a *slash 24* address. The part of the address that is common to all of the nodes in the network is called the *prefix* and the number after the slash gives the *prefix length*.

This may all seem rather academic because we already know from the first byte of the address which class it falls into and so what the prefix length is. Class A addresses are slash 8 addresses, Class B addresses are slash 16s, and Class C gives us slash 24 addresses. But the segmentation of the IPv4 address space doesn't stop here, because there is an administrative need to break the address space up into smaller chunks within each of the address groups. This process is called *subnetting*.

Consider an Internet Service Provider (ISP) that applies to its local IP addressing authority for a block of addresses. It doesn't expect to have many customers, so it asks for three Class C address groups (a total of 762 hosts). As customers sign up with the ISP it needs to allocate these addresses to the hosts in the customers' networks. Obviously, if it allocates each customer a whole Class C address group it will run out of addresses after just three customers. This wastes addresses if each customer has only a few hosts.

A more optimal way for the ISP to allocate its addresses is to break its address groups into smaller groups, one for each subnetwork that it manages. This can be done in a structured way, as shown in Table 2.5, although some care has to be taken that the blocks of addresses carved out in this way fit together correctly without leaving any unused addresses.

Table 2.5 An Example of Subnetting an Address Group

Address Range	*Subnet*	*Subnet Mask*	*Number of Hosts*
192.168.20.1 to 192.168.20.14	192.168.20.0/28	255.255.255.240	14
192.168.20.17 to 192.168.20.30	192.168.20.16/28	255.255.255.240	14
192.168.20.33 to 192.168.20.40	192.168.20.32/30	255.255.255.252	6
192.168.20.41 to 192.168.20.48	192.168.20.40/30	255.255.255.252	6
192.168.20.49 to 192.168.20.112	192.168.20.48/26	255.255.255.192	62

In each subnet, we can start numbering the hosts from 1 upwards. So, for example, in the subnet represented by the second row in Table 2.5, the first host has the address 192.168.20.16+1, that is, 192.168.20.17. Recall that in each subnetwork the zero address (for example, 192.168.20.16) and the all ones address (for example, 192.168.20.15) are reserved.

2.3.4 Network Address Translation (NAT)

Some groups of addresses are reserved for use in private networks and are never exposed to the wider Internet. The ranges appear to be random and have historic reasons for their values, but note that there is one address range chosen from each of Class A, B, and C. They are shown in Table 2.6.

Of course, in a genuinely private network any addresses could be used, but it is a good exercise in self-discipline to use the allocated ranges. Further, if the private network does become attached to the public Internet at some point, it is much easier to see whether internal addresses are leaking into the Internet and simple for the ISP to configure routers to filter out any packets to or from the private address ranges.

If a network using one of the private address ranges is connected to the Internet, Network Address Translation (NAT) must be applied to map local addresses into publicly visible addresses. This process provides a useful security barrier since no information about the internal addressing or routing structure will leak out into the wider Internet. Further, the private network can exist with only a small number of public addresses because only a few of the hosts in the private network will be attached to the Internet at any time.

This scheme and the repeated use of private address ranges across multiple networks is an important step in the conservation of IP addresses. As more and more devices from printers and photocopiers to heating systems and refrigerators were made IP-capable for office or home networking, there was a serious concern that all of the 2^{32} IPv4 addresses would be used up. However, by placing hosts within private networks the speed of the address space depletion has been dramatically reduced.

Table 2.6 Addresses Reserved for Use on Private Networks

Address Range	*Subnet*
10.0.0.0 to 10.255.255.255	10.0.0.0/8
172.16.0.0 to 172.31.255.255	172.16.0.0/12
192.168.0.0 to 192.168.255.255	192.168.0.0/16

2.4 IP in Use

This section examines how IP is used to deliver datagrams to their target hosts. This is largely an issue of addressing since, on the local network segment, IP datagrams are encapsulated in data-link layer frames that are delivered according to their Media Access Control (MAC) addresses. This section looks at how network segments can be extended by devices called *bridges*, which selectively forward frames based on their MAC addresses.

IP has its own addressing, which was introduced in the previous section. When bridges don't provide sufficient function, devices called *routers* are used to direct IP datagrams based on the destination IP addresses they contain. Routers may perform direct routing using look-up tables such as the one produced by ARP (see Section 2.4.5) to map IP addresses to MAC addresses and so package the IP datagrams into frames that are addressed to their ultimate destinations. Direct routing is useful on a single physical network such as an Ethernet, but is impractical when many separate networks are linked together.

Indirect routing lets routers forward IP datagrams based on their destination IP addresses. For any one datagram, a router determines the *next hop* router along the path to the destination and packages the datagram in a data-link level frame addressed to that router. The next hop router is the gateway into the next segment of the network.

Routers determine the next hop router by using routing tables that may be statically configured or dynamically computed using the information distributed by the routing protocols described in Chapter 5.

2.4.1 Bridging Function

The reach of a network such as an Ethernet can be extended by joining two segments together with a *repeater*. A repeater is simply a piece of equipment that listens on one port and regenerates everything it hears, indiscriminately, out of another port. Repeaters usually help overcome the effects of signal distortion, attenuation, and interference from noise by regenerating a clean signal.

Repeaters do not, however, solve the problem of congestion that can occur as the number of nodes on a network grows, because repeaters forward all traffic between the network segments. On an Ethernet this gives rise to an unacceptable number of collisions, and on a Token Ring or FDDI network the amount of "jitter" may become excessive as each node must wait longer for the token.

The solution is to segment the network and filter the traffic that passes through the connections between the segments. The function of connecting segments and filtering traffic is provided by a *bridge*, which is a sort of intelligent repeater. Bridges act by learning which MAC addresses exist on each segment. They build a table that associates each MAC address with one of the bridge's output ports according to which segment it belongs to. When a bridge receives

a frame it first looks up the source MAC address and if it hasn't seen it before it adds an entry to its bridging table saying that the address can be reached through the port on which the frame was received. Second, it forwards the frame according to the entry for the destination address in its bridging table. If the destination address is not in its bridging table or if the address is a broadcast address, the bridge simply acts as a repeater and sends the frame out of all of the ports except the one through which the frame was received. Further, if the bridge receives a frame and notices that the destination address is reachable through the port on which the frame was received, the bridge simply discards the frame. In this way, once the network has been run for a while, the bridge effectively filters the traffic that it receives and only forwards traffic onto the correct network segments. Figure 2.9 shows a simple network segmented with the use of a bridge and also shows the view that a host might have of the same network. Table 2.7 shows the bridging table that the bridge would construct after listening to the network for a while.

Bridges can be daisy-chained so that the network can be broken up still further, but great care must be taken to avoid building looped network segments. This would result in broadcast traffic looping forever and also could result in targeted frames looping. This is a network planning issue—there must be no loops in bridged networks.

Note that while bridges facilitate smoother operation of IP, they are in no way involved in the protocol or in IP addressing.

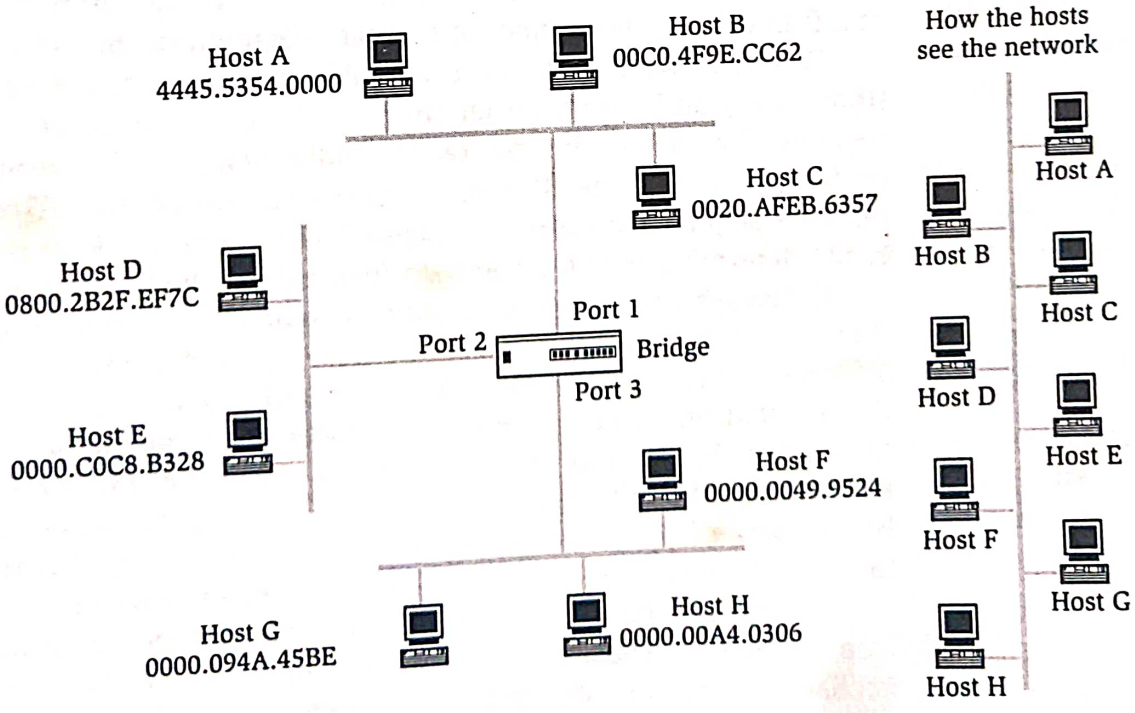

Figure 2.9 An Ethernet broken into three segments by a bridge still appears to be a single Ethernet when viewed by the attached hosts.

Table 2.7 The Bridging Table for the Router in Figure 2.9

Hardware Address	*Bridge Port*
0000.0049.9524	3
0000.00A4.0306	3
0000.094A.45BE	3
0000.C0C8.B328	2
0020.AFEB.6357	1
00C0.4F9E.CC62	1
0800.2B2F.EF7C	2
4445.5354.0000	1

2.4.2 IP Switching and Routing

Although bridges greatly improve the scalability of networks, they aren't a complete solution. As described in the previous section, bridges have an intrinsic risk of looping so that frames circulate forever if the network is not connected correctly. Further, they don't even handle scaling completely, since all broadcast frames must be copied on to every segment of the network.

Gateways were introduced as smart bridges that act on the network layer (that is, IP) addressing rather than on the data-link layer. The network is still segmented as before, but packets are only forwarded between network segments by the gateway if the IP address meets certain criteria. The addresses on one network segment are usually assigned as a subnetwork so that the gateway can easily determine which packets to forward in which direction.

Gateways are now called *routers*, which is a pretty good name for what they do. Bridges simply choose the next network fragment onto which to send a frame, but routers have a wider view and can see a logical, multi-hop path to the destination, which they use to choose the next router to which to forward the packet. In this way, routers can safely be built into a *mesh network*, which has loops and multiple alternative paths. Using the properties of IP and their broad view of the network, routers can select preferred paths based on many factors, of which path length is usually the most important.

Routers get their knowledge of the network topology from *routing protocols* (see Chapter 5). These protocols allow routers to communicate with each other, exchanging information such as connectivity and link types so that they build up a picture of the whole network and can choose the best way to forward a packet. A key distinction should be made between hosts and routers since both

```
C:\> route print
Active Routes:
Network Address          Netmask          Gateway Address       Interface       Metric
0.0.0.0                  0.0.0.0          138.88.23.191         138.88.23.191   1
127.0.0.0                255.0.0.0        127.0.0.1             127.0.0.1       1
138.88.0.0               255.255.0.0      138.88.23.191         138.88.23.191   1
138.88.23.191            255.255.255.255  127.0.0.1             127.0.0.1       1
138.88.255.255           255.255.255.255  138.88.23.191         138.88.23.191   1
224.0.0.0                224.0.0.0        138.88.23.191         138.88.23.191   1
255.255.255.255          255.255.255.255  138.88.23.191         138.88.23.191   1
```

Figure 2.10 The output from a standard route table application on a dial-up PC.

include a routing function: A router is capable of forwarding received packets, but a host is only capable of originating or terminating packets.

The routing function within a host determines what the host does with a packet it has built and needs to send. Typically, a host is directly attached to a router or is connected to a network segment with only one router attached. In this case, the host has a very simple route table like the one shown in Figure 2.10 for a dial-up host. The output from a standard route table application shows us how the host must operate. If the host had a packet to send to the destination address 192.168.20.59, it would start at the top of the table and work down the Network Address column using the Netmask to compare entries with the destination address. Most of the table entries are concerned with connections to the local PC itself (through its address 138.88.23.191 or local host address 127.0.0.1—see the next section), or with delivery to the directly attached subnet 138.88.0.0/16. It is only the last line in the table that tells the host how to handle its packet. This line represents the *default route* since it catches every IP address that has not been handled by another line in the table. It tells the host that it must forward the packet through the gateway 138.88.23.191 (that is, itself) and out of the interface 138.88.23.191 (that is, the modem providing dial-up connectivity).

Routers have similar but far more complex routing tables built by examining the information provided by the routing protocols.

IP switches are a hybrid of bridging and routing technology. They operate by building a table of IP addresses mapped to outgoing interfaces, and look up the destination address of a packet to determine the interface out of which to forward the packet. They switch the packet from one interface to the next at a network level in the same way that a bridge switches a packet at a data-link level. Switches are explicitly programmed to forward traffic on specific paths either through configuration and management options or through special protocols devised to distribute this information. Refer to Multiprotocol Label Switching (MPLS) in Chapter 9 and the General Switch Management Protocol (GSMP) in

Chapter 11. Switches do not learn network topology through routing protocols and do not calculate their routing tables dynamically.

For a while, IP switches were the shining hope of the Internet. They did not require continual calculation of routes each time the routing protocols distributed new information, and they could place their switching table in integrated circuitry (ASICs) for much more rapid forwarding. However, it soon became apparent that a new generation of routers could be built that also used ASICs for forwarding while retaining the flexibility and reactivity of the routing protocols. IP switches have pretty much died out except for simple programmable switches that offer a level of function somewhere between a bridge and a router (popular in home networking). However, packet switching is alive and well in the form of MPLS (described in Chapter 9).

2.4.3 Local Delivery and Loopbacks

The routing table in Figure 2.10 contains some unexpected entries as well as the more obvious ones for the directly connected network and the default route. The entry for the subnet 127.0.0.0/8 says that all matching addresses should be forwarded out of the interface 127.0.0.1. What is more, addresses matching 138.88.23.191/32, the host's own address, should also be forwarded out of the same interface.

127.0.0.1 is the address used to represent the *localhost*. Any packet carrying this address will be delivered to the local IP stack without actually leaving the host. So, any packet addressed to the host's external IP address 138.88.23.191 will be delivered to the local IP stack. Try typing "ping 127.0.0.1" on a PC that is not connected to the network and you will see that you get a response.

When a node (host or router) has many interfaces, each is assigned its own external IP address known as the interface address. In this case it is useful to assign another address that represents the node itself and is not tied to a specific interface. We are looking for an address that has external scope but that applies to the whole of the node (that is, an address that is not limited to the identification of a single interface). Such addresses are known as *loopback* addresses because if a node sends an IP packet to its own loopback address, the packet is looped back and returned up its own IP stack. The concept of loopbacks is useful not just in providing an external view of the whole node, but also in allowing client and server applications to be run on one node and use the same processing that would be used if one of the applications (say the server) were located on a remote box. This similarity of processing extends through all of the application and transport software, right up to the IP stack which, when it is asked to send a packet to a loopback address, turns the packet around as if it had just been received. This is illustrated in Figure 2.11. Note that 127.0.0.1 is a loopback address, but it is not an externally visible loopback address, that is, it cannot be used by a remote node to send a packet to this node because all nodes use the same value, 127.0.0.1, as their localhost loopback address. An individual node may define as many loopback addresses as it wishes.

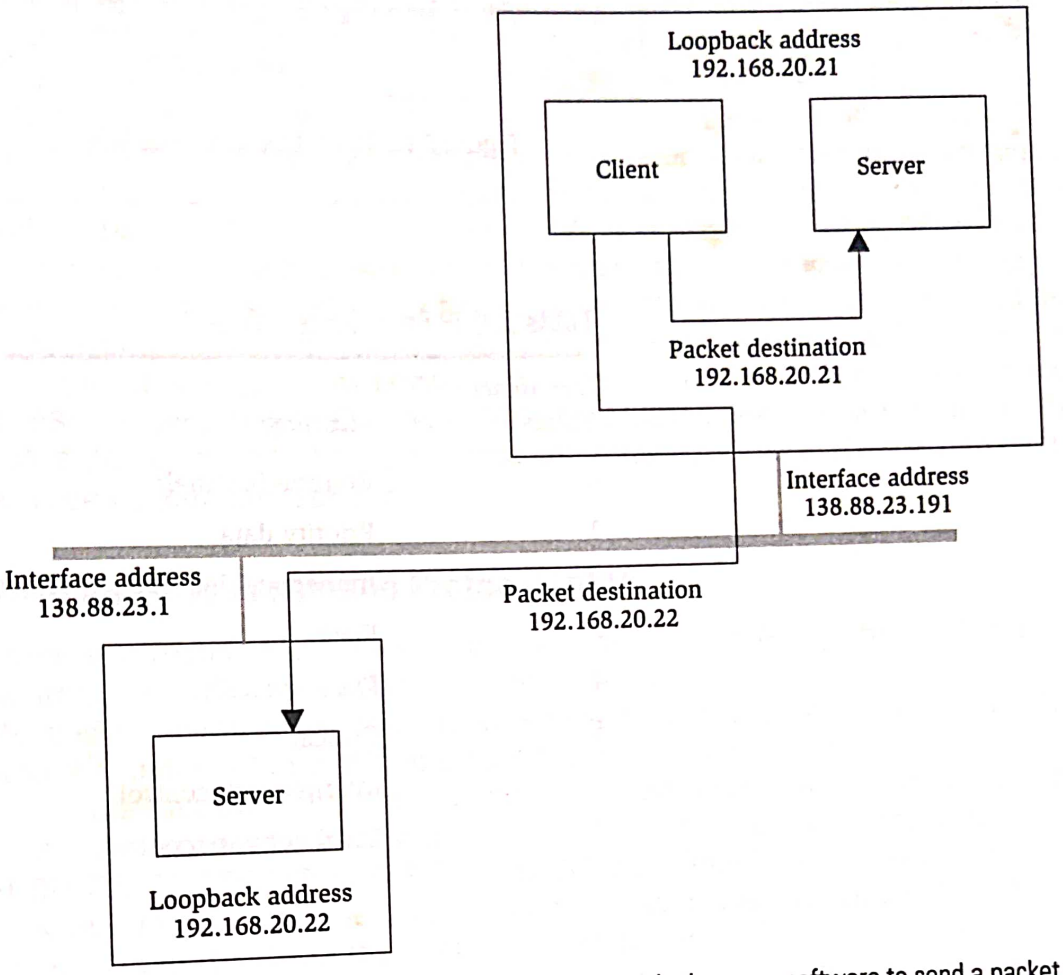

Figure 2.11 Loopback addresses enable a host to use exactly the same software to send a packet to a remote or a local application.

Great care must be taken to distinguish between a loopback address and a router ID. Router IDs are used to distinguish and identify routers in a network, and the concept is important in the routing protocols described in Chapter 5. Since router IDs and IPv4 addresses are both 32-bit fields, it is common practice to assign the router ID to be the first defined loopback address, but this need not be the case and the router ID should not be assumed to be an IP address.

2.4.4 Type of Service

The routing tables just described provide a simple address-based way of forwarding packets. The destination address of the packet is checked against each entry in the table in turn until a match is made, identifying the interface out of which the packet should be forwarded.

Routing allows an extra level of flexibility so that packets may be forwarded based on the service type requested by the application and may be prioritized for

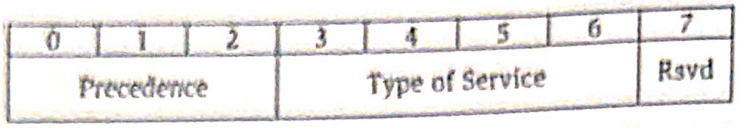

Figure 2.12 The IP type of service byte.

Table 2.8 IP Precedence Values

Precedence Value	*Meaning*
0	Routine (normal)
1	Priority data
2	Immediate delivery required
3	Flash
4	Flash override
5	Critical
6	Internetwork control
7	Local network control

Table 2.9 IP Type of Service Values

ToS Value	*Meaning*
0	Normal service. In practice almost all IP datagrams are sent with this ToS and with precedence zero.
1	Minimize delay. Select a route with emphasis on low latency.
2	Maximize throughput. Select a route that provides higher throughput.
4	Maximize reliability. Choose a route that offers greater reliability as measured by whatever mechanisms are available (such as bit error rates, packet failure rates, service up time, etc.).
8	Minimize cost. Select the least expensive route. Cost is usually inversely associated with the length of the route and this is the default operating principle of the algorithms that routers use to build their routing tables.
15	Maximize security. Pick the most secure route available.

processing within the routers that handle the packets. The service level requested by the application is carried in the Type of Service (ToS) byte in the IP header.

The ToS byte is broken into three fields, as shown in Figure 2.12. The Precedence field defines the priority that the router should assign to the packet. This allows a high-priority packet to overtake lower-priority packets within the router, and allows the higher-priority packets better access to constrained resources such as buffers. The list of precedence values is shown in Table 2.8, in which priority seven is the highest priority. A router can use the value of the Type of Service field to help it select the route based on the service requirements. Values of the ToS field are maintained by IANA: the current values are listed in Table 2.9.

Type of Service is now largely deprecated in favor of Differentiated Services (DiffServ), described in Chapter 6. This deprecation and reinterpretation of the ToS field is possible because nearly all datagrams were always sent using precedence zero and ToS zero.

2.4.5 Address Resolution Protocol (ARP)

Suppose a router receives an IP packet. The router looks up the destination IP address carried by the packet and determines the next hop to which to forward the packet—perhaps the destination is even directly attached to the router. This tells the router out of which interface it should send the packet.

If the link from the router is point-to-point, everything is simple and the router can wrap the packet in the data-link layer protocol and send it. However, if the link is a multidrop link such as Ethernet (that is, it is a shared-medium link with multiple attached nodes) the router must also determine the data-link layer address (such as the MAC address) to forward the packet to the right node. If we could simply make the IP address of a node equal to its MAC address there would be no issue, but MAC addresses are typically 6 bytes, which cannot be carried in the 4-byte IP address field. Besides, in IP we like to be able to assign multiple addresses to a single node.

An option would be to broadcast the packet on the link and let the receiving nodes determine from the IP address of the packet whether it is for them. This might be acceptable when all of the attached nodes are hosts, but if any node was a router it would immediately try to forward the packet itself. What is really required is a way of resolving the data-link layer address from the next hop IP address.

The simple solution is to configure the router with a mapping table. This certainly works, but adds configuration overhead and is inflexible, and the whole point of a network such as a multidrop Ethernet is that you can simply plug in new nodes and run with the minimum of fuss. If you had to visit every node on the network and add a configuration item it would be a nuisance.

The Address Resolution Protocol (ARP) defined in RFC 826 solves this problem for us by allowing nodes to announce their presence on a network and also to query MAC addresses based on given IP addresses. When a node is plugged into an Ethernet it announces its presence by advertising the IP address of its

Table 2.10 ARP Operation Codes

Operation Code	*Meaning*
1	ARP Request. Please supply the IP address corresponding to the requested target MAC address.
2	ARP Reply. Here is a mapping of target MAC address to target IP address.
3	RARP Request. Please supply my IP address given my MAC address.
4	RARP Reply. Here is your IP address given your MAC address.
8	InARP Request. Please supply the MAC address corresponding to the target IP address.
9	InARP Reply. Here is a mapping of target IP address to target MAC address.

attachment to the Ethernet together with its MAC address in a process known as *gratuitous ARP*. This advertisement message is broadcast at the Ethernet level using the MAC address 0xFFFFFFFFFFFF so that all other nodes on the network receive it and can add the information to their mapping tables or *ARP caches*. When a node sends an IP packet to this host it can look up the address of the host in its cache and determine the correct MAC address to use.

The format of all ARP messages is shown in Figure 2.13. The ARP message is illustrated encapsulated in an Ethernet frame shown in gray, which indicates that the payload protocol is ARP (0x0806). The ARP message indicates the hardware type (Ethernet) and the protocol type (IP). Next it gives the lengths of the hardware address (6 bytes for a MAC address) and the protocol address (4 bytes for IPv4). Including these address length fields makes ARP applicable to any data-link layer and any network protocol. The next field in the ARP message identifies the ARP operation—in this case it is set to 2 to indicate an ARP Reply. Table 2.10 lists the full set of ARP operation codes.

The last four fields in the ARP message carry the ARP information. The source MAC and IP addresses are always present and give the information about the sender of the message. The target MAC and IP addresses are set according to the Operation code. If the operation is an ARP Request the target MAC address is zero, because a request is being made to resolve the target IP address into a MAC address. But when the operation is an ARP Reply both the target MAC and IP addresses are present.

A node that receives an ARP Reply message extracts the address mapping information and stores it in its *ARP cache*. It uses the ARP cache information for all subsequent IP messages that it sends. But this process causes a raft of issues, largely associated with what happens when an entry in the cache is wrong or absent.

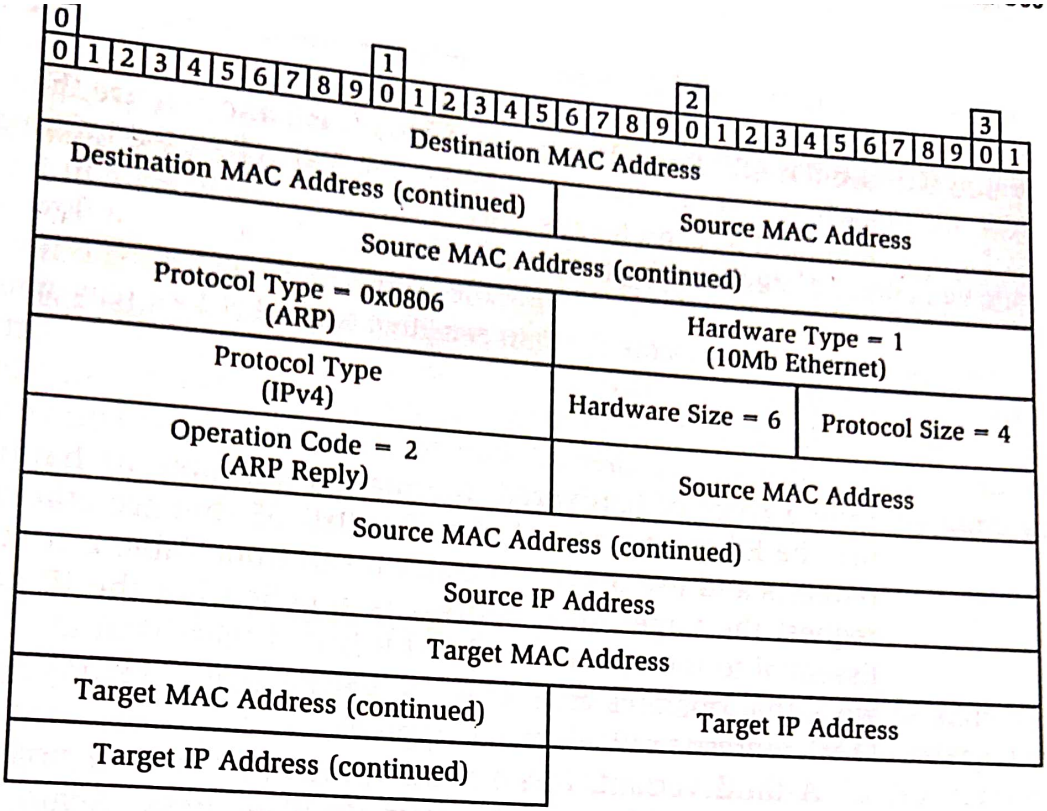

Figure 2.13 The format of an ARP message encapsulated in an Ethernet frame.

The cache might not contain the mapping for a particular host simply because the local host booted more recently than the remote host and so has not heard an ARP reply from the remote host. Alternatively, the cache may be of limited size so that entries are discarded based on least use or greatest age. Further, it may be considered wise to time out cache entries regardless of how much use they are getting so that stale and potentially inaccurate information does not persist. In any of these cases, the local host cannot send its IP packet because it doesn't have a mapping of the next hop address.

A solution is to have all nodes periodically retransmit their IP to MAC address mapping. This would mean that a node only had to wait a well-known period of time before it had an up-to-date entry in its ARP cache. But we would need to keep this time period quite small, perhaps in the order of 10 seconds, and that would imply an unacceptable amount of traffic on an Ethernet with 100 stations. Besides, if the reason the entry is missing from the cache is that the cache is smaller than the number of nodes on the Ethernet, this approach would not work.

What is needed instead is the ability for a node to query the network to determine the address mapping. To do this, a node that cannot map the next hop IP address of a packet to a MAC address discards the IP packet and sends an ARP Request (Operation code one) instead. The ARP Request is broadcast (as are all ARP messages) and contains the target IP address for resolution—the target MAC address is set to zero. All nodes on the local network receive the

broadcast ARP Request, but only the one that matches the target MAC address responds—it builds an ARP reply and broadcasts it on the Ethernet. All nodes see the ARP Reply because it is broadcast, and can use the Reply to update their caches. The originator of the Request will collect the information from the Reply and hold it ready for the next IP packet it has to send in the same direction.

Note that this process can cause a "false start" in a flow of IP packets as a few packets are discarded along the path while addresses are resolved. Applications or transport protocols that are sensitive to packet loss may notice this behavior and throttle back the rate at which they send data, believing the network to be unreliable.

Diskless workstations may use *Reverse ARP* (RARP) to discover their own IP address when they are booted. Typically a node knows its own MAC address from a piece of hardwired information built into its hardware. RARP requests use the Ethernet protocol identifier 0x0835, but are otherwise identical to ARP requests and use operation codes taken from Table 2.10. Obviously, in a RARP request the target MAC address is supplied but the IP address is set to zero. Essential to the operation of RARP is that there is at least one node on the network that operates as a RARP server keeping a record of manually configured MAC address to IP address mappings.

A third variant, *Inverse ARP* (InARP), allows a node that knows another node's MAC address to discover its IP address. Again, message formats are identical and operation codes are listed in Table 2.10. InARP is particularly useful in data-link networks such as Frame Relay and ATM where permanent virtual circuits may already exist between known data-link end points and where there is a need to discover the IP address that lives at the other end of the link.

The extended use of the ARP operation code is a good example of how a well-designed protocol can be extended in a simple way. If the original protocol definition had assigned just one bit to distinguish the request and reply messages, it would have been much harder to fit RARP and InARP into the same message formats.

Most operating systems give the user access to the ARP cache and allow the user to manually add entries and to query the network. Figure 2.14 shows the options to a popular ARP application.

```
c:\> arp
ARP -a [inet_addr] [-N if_addr]
ARP -d inet_addr [if_addr]
ARP -s inet_addr eth_addr [if_addr]
-a Displays current ARP cache. If inet_addr is specified, only
   the specified entry is displayed. If -N if_addr is used the
   ARP entries for the network interface are displayed.
-d Deletes the ARP cache entry specified by inet_addr.
-s Sets an entry in the ARP cache modifying any existing entry.
```

Figure 2.14 The options to a popular ARP application.

Although ARP request messages are broadcast, they must be responded to only by the node that matches the target address. This is important because if a response is generated by every node that knows the answer to the query there will be a large burst of responses. Obviously, this rule is bent for RARP, in which it is the RARP server that responds.

One last variation on ARP is *proxy* ARP, where ARP requests are answered by a server on behalf of the queried node. This is particularly useful in bridged and dial-up networks. See Chapter 15 for more discussion of proxy ARP.

Note finally that ARP can be used to help detect the problem of two nodes on a network having the same IP address. If a node receives an ARP response that carries its own IP address but a MAC address that is not its own, it should log messages to error logs and the user's screen, and should generally jump up and down and wave its hands in the air.

2.4.6 Dynamic Address Assignment

Reverse ARP offers a node the opportunity to discover its own IP address given its MAC address. This useful feature is extended by the Bootstrap Protocol (BOOTP) defined in RFC 951 and RFC 1542. This allows a diskless terminal not only to discover its own IP address, but to download an executable image. Such a device can be built with a small program burned into a chip containing BOOTP and the Trivial File Transfer Protocol (TFTP) described in Chapter 12. Upon booting, the node broadcasts the message shown in Figure 2.15 with most fields set to zero and the operation code set to 1 to show that this is a request. If the node has been configured (perhaps in a piece of flash RAM called the BootPROM) with the BOOTP server's address, the message is sent direct and is not broadcast. The BOOTP server fills in the message and returns it to the requester, which is able to initiate file transfer and so load its real executable image. The returned message is usually broadcast as well since the client does not yet know its own IP address and so cannot receive unicast messages, but if the client fills its address into the BOOTP Request and that address matches the one supplied by the server in the BOOTP Response, the response may be unicast.

Note that the information fields are fixed length, so a length field is needed to indicate how many bytes of the hardware address are relevant. There is an option for the client to fill in its IP address if it believes it knows the address and if the node is really just soliciting the boot information.

BOOTP messages are sent encapsulated in IP and UDP (see Chapter 7), so BOOTP is really an application protocol.

The boot-time function of BOOTP is enhanced by the Dynamic Host Configuration Protocol (DHCP) defined in RFC 2131. DHCP is backwards compatible with BOOTP but adds further facilities for remote configuration of the network capabilities of a booting node. DHCP uses the same message structure as BOOTP, but the Reserved field is allocated for use as flags and the Vendor-Specific field is used for passing configuration options. This is illustrated in Figure 2.16.

Options zero and one are used to delimit the sequence of options in the header. These are special options since they don't use the TLV encoding properly and are present as single-byte option types with neither length nor variable. Option zero is, perhaps, unnecessary given the presence of the header length field, but is used in any case as the last option in the list. Option one is used to provide padding so that the header is built up to a 4-byte boundary.

2.5.1 Route Control and Recording

Option seven can be used to track the path datagrams follow through the network. This will tell the destination how the packet got there, but it won't tell the sender a thing since the datagram is a one-way message. The sender may place the TLV shown in Figure 2.21 into the IP options field. The sender must leave enough space (set to zero) for the nodes along the path to insert their addresses since they cannot increase the size of the IP header (this might otherwise involve memory copying and so forth).

Each node that receives a datagram with the *Record Route* option present needs to add its IP address to the list in the TLV. The Pointer field is used to tell the node where it should place its address within the option, so initially the pointer field is set to 4—the fourth byte is the first available byte in which to record a hop. As each hop address is added, the pointer field is increased by four to point to the next vacant entry. When the option is received and the pointer value plus four exceeds the option length, the receiving node knows that there is not enough space to record its own address and it simply forwards the datagram without adding its address.

Options three and nine are used to allow the source node to have some control over the path taken by a datagram, thereby overriding the decisions that would otherwise be made by routers within the network. There may be many motivations for this—some are tied to the discussion of fragmentation and may allow a source node to direct datagrams through networks that will not need to fragment the data. Other motivations are related to traffic engineering, described in Chapter 8.

The *Source Route* is a series of IP addresses that identify the routers that must be visited by the datagram on its way through the network. Two alternatives

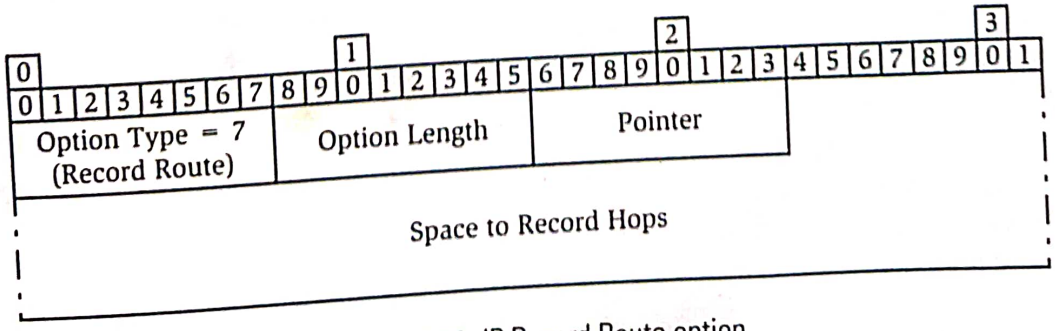

Figure 2.21 IP Record Route option.

exist: A *strict route* is a list of routers that must be visited one at a time, in the specified order and without using any intervening routers. The Record Route generated as a result of using a strict route would show exactly the same series of addresses. The second alternative is a *loose route*, which lists routers that must be visited in order, but allows other routers to be used on the way. The Record Route in this case would be a superset of the source route.

The strict and loose route options use the same format as the Record Route shown in Figure 2.21. That is, the route is expressed as a list of 32-bit IP addresses. The pointer field is used to indicate the address that is currently being processed—that is, it indicates the target of the current hop. When a datagram is sent out using a Source Route, the datagram is addressed not to the ultimate destination, but to the first address in the Source Route. When the datagram is received at the next router it compares the destination address, the next address pointed to in the route, and its own address. If the route is a strict route, all three must be equal. If it is not, an ICMP error is generated (see Section 2.6). If the route is a loose route and the current node is not the current destination of the datagram, the datagram is forwarded. When the datagram is received by a node that is the current destination, it copies the next address from the route to the destination field in the IP header, increments the pointer, and forwards the datagram. The last entry in the Source Route is the real destination of the datagram. Figure 2.22 shows this process at work.

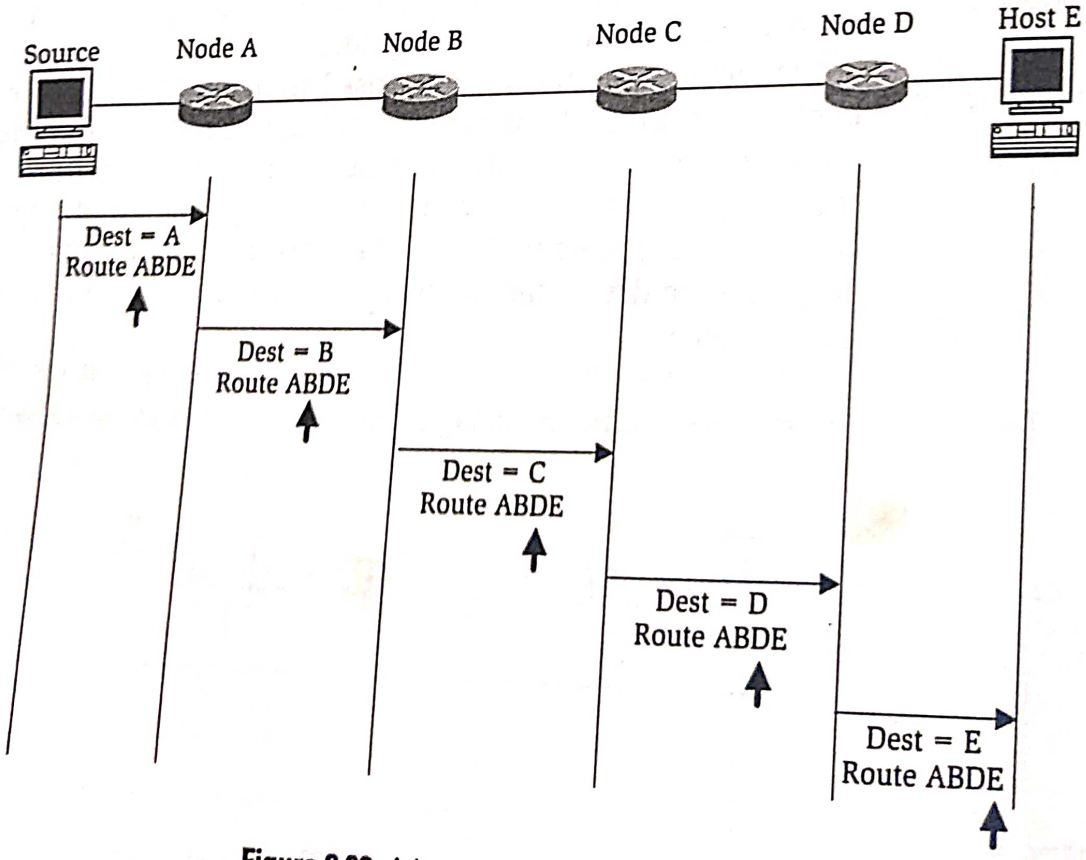

Figure 2.22 A loose source route in use.

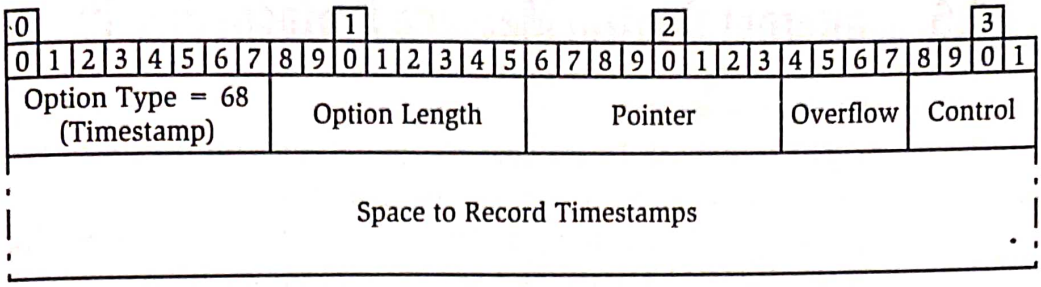

Figure 2.23 IP Timestamp option.

It should be clear that a strict route effectively records the route without using a Record Route option. This is good because it would be hard to fit both into the limited space available for IP options. Note that there is an issue with using loose route to record the path of the datagram because it may take in additional nodes along the way. This is somewhat ambiguously described in RFC 791, but it is clear that there is no place to insert the Record Route to track each node along the path.

There are several concerns with the Source Route options, including the fact that an intruder may contrive to get datagrams that are sent between nodes in one network to be routed through another network. This would allow the intruder to get access to the datagrams. In any case, many network operators don't like the idea of a user having control of how their traffic is carried through the network—that is their responsibility and the job of the routing protocols. In consequence, support for source routing is often disabled.

Given the size limitations of the IP header, these three options are not considered very useful. At best, the Record Route option can gather nine addresses. Although it may once have been considered remarkable to have three or four routers between source and destination, there are now often many more than nine. Further, many IP implementations have some trouble managing source routes correctly and do not always manage to forward the datagrams properly.

The Timestamp option is similar to the Record Route option. As shown in Figure 2.23, it includes two additional 4-bit flags (so the first value of the pointer field is five) to control the behavior of the option. The Control Flag field has three settings—zero means that each node visited by the datagram should insert a timestamp, one means that each node should insert its address and a timestamp, three means that the option already includes a list of IP addresses and space for the specified nodes to fill in their timestamps. The other 4-bit field is the Overflow field. This is used to count the number of nodes that are unable to supply a timestamp because there is no more space in the option.

Even with the Overflow field the Timestamp option runs up against the same space problems as Record Route. The option to record timestamps selectively (option three) may be used to mitigate this to some extent, but can't be used unless the source knows which nodes the datagram will visit.

2.6 Internet Control Message Protocol (ICMP)

The Internet Control Message Protocol (ICMP) is a funny animal. It is used to report errors in IP datagrams, which can be useful to the sender and to transit nodes because they can discover and isolate problems within the network and possibly select different paths down which to send datagrams. At the same time, ICMP supports two very useful applications, *ping* and *traceroute*, that are used to discover the reachability of remote IP addresses and to inspect the route that datagrams follow to get to their destinations. Further, ICMP contains the facility to "discover" routers within the network, which is useful for a host to discover its first hop to the outside world.

These features mean that ICMP is sometimes described as a routing protocol, but because it predates the fully fledged routing protocols and it may be used in a very selective way, it is described here rather than in Chapter 5.

To quote from RFC 1122, the standard that defines what a host attached to the Internet must be capable of doing, "ICMP is a control protocol that is considered to be an integral part of IP." This says that any node that professes to support IP must also support ICMP.

2.6.1 Messages and Formats

ICMP messages are carried as the payload of IP datagrams with the Next Protocol field set to 1 to indicate that the data contains an ICMP message, as shown in Figure 2.24. Each ICMP message begins with three standard fields: the Message Type indicates which ICMP message is present, and the Message Code qualifies this for meaning specific to the type of message. Table 2.14 lists the ICMP message types.

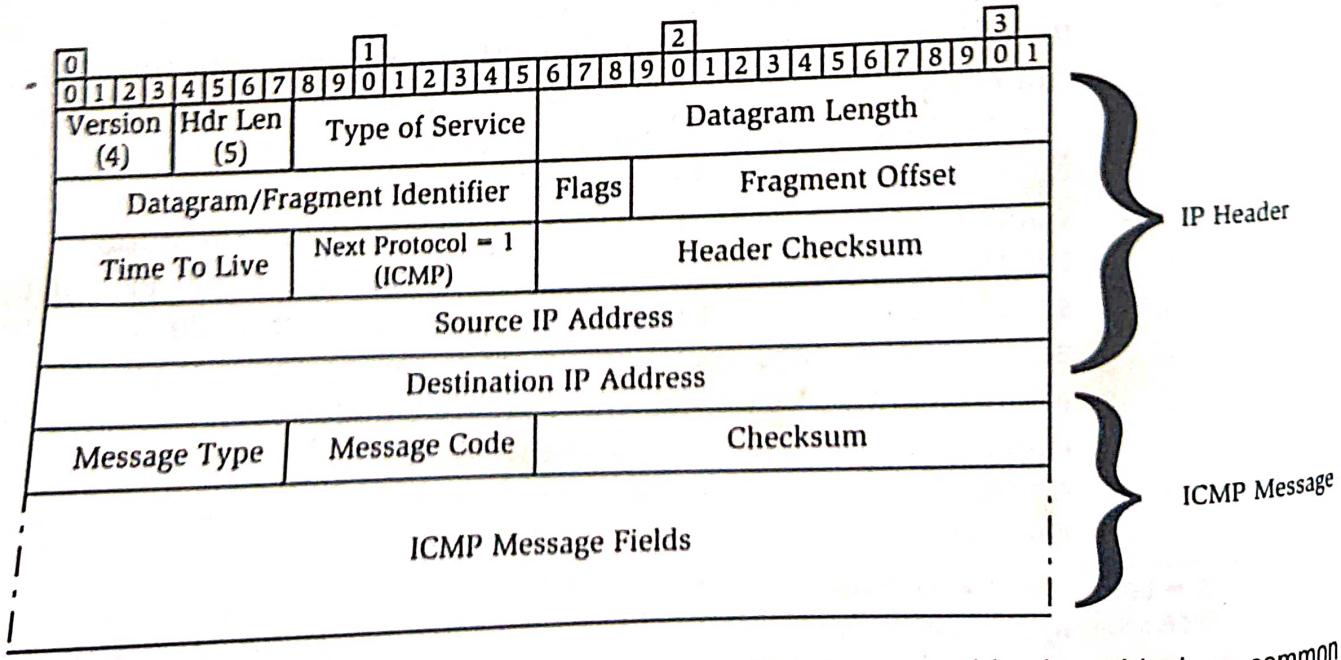

Figure 2.24 An ICMP message is encapsulated in an IP datagram and begins with three common fields and then continues according to the message type.

Table 2.14 The ICMP Messages

Message Type	*Message*
0	Echo Reply. Sent in direct response to an ICMP Echo Request message.
3	Destination Unreachable. An error message sent when a node cannot forward any IP datagram toward its destination.
4	Source Quench. Sent by a destination node to slow down the rate at which a source node sends IP datagrams.
5	Redirect. Used to tell a source node that there is a better first hop for it to use when trying to send IP datagrams to a given destination.
8	Echo. Sent by a node to probe the network for reachability to a particular destination.
9	Router Advertisement. Used by a router to tell hosts in its network that it exists and is ready for service.
10	Router Solicitation. Used by a host to discover which routers are available for use.
11	Time Exceeded. An error message generated by a router when it cannot forward an IP datagram because the TTL has expired.
12	Parameter Problem. An error sent by any node that discovers a problem with an IP datagram it has received.
13	Timestamp Request. Used to probe the network for the transmission and processing latency of messages to a given destination.
14	Timestamp Reply. Used in direct response to a Timestamp Request message.
15	Information Request. Used by a host to discover the subnet to which it is attached.
16	Information Reply. Used in direct response to an Information Request message.
17	Address Mask Request. Used by a host to discover the subnet mask for the network to which it is attached.
18	Address Mask Reply. Used in direct response to an Address Mask Request message.

The third common ICMP message field is a checksum. This is calculated across the whole ICMP message, including the three common fields but not including the IP header, using the algorithm described in Section 2.2.3. After the checksum are found fields specific to the type of message. These are described later in this section.

2.6.2 Error Reporting and Diagnosis

ICMP can be used to report errors with the delivery or forwarding of IP datagrams. The intention is to report only nontransient errors. Transient errors (that is, those