CHAPTER

9

Internet Control Message Protocol Version 4 (ICMPv4)

As discussed in Chapter 7, the IPv4 provides unreliable and connectionless datagram delivery. It was designed this way to make efficient use of network resources. The IP protocol is a best-effort delivery service that delivers a datagram from its original source to its final destination. However, it has two deficiencies: lack of error control and lack of assistance mechanisms. ICMPv4 is designed to compensate for these deficiencies.

OBJECTIVES

The chapter has several objectives:

- ❑ To discuss the rationale for the existence of ICMP.
- ❑ To show how ICMP messages are divided into two categories: error-reporting and query messages.
- ❑ To discuss the purpose and format of error-reporting messages.
- ❑ To discuss the purpose and format of query messages.
- ❑ To show how the checksum is calculated for an ICMP message.
- ❑ To show how debugging tools using the ICMP protocol.
- ❑ To show how a simple software package that implements ICMP is organized.

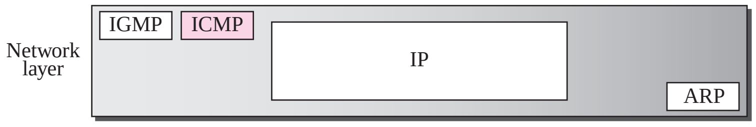
9.1 INTRODUCTION

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

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

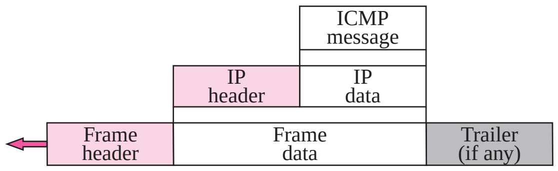
The **Internet Control Message Protocol (ICMP)** has been designed to compensate for the above two deficiencies. It is a companion to the IP protocol. Figure 9.1 shows the position of ICMP in relation to IP and other protocols in the network layer.

Figure 9.1 *Position of ICMP in the network layer*



ICMP itself is a network layer protocol. However, its messages are not passed directly to the data link layer as would be expected. Instead, the messages are first encapsulated inside IP datagrams before going to the lower layer (see Figure 9.2).

Figure 9.2 *ICMP encapsulation*



The value of the protocol field in the IP datagram is 1 to indicate that the IP data is an ICMP message.

9.2 MESSAGES

ICMP messages are divided into two broad categories: **error-reporting messages** and **query messages.** The error-reporting messages report problems that a router or a host (destination) may encounter when it processes an IP packet. The query messages, which occur in pairs, help a host or a network manager get specific information from a router or another host. For example, nodes can discover their neighbors. Also, hosts can discover and learn about routers on their network and routers can help a node redirect its messages. Table 9.1 lists the ICMP messages in each category.

Table 9.1 *ICMP messages*

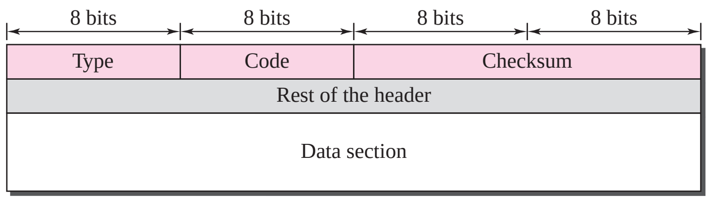
Category	*Type*	*Message*
Error-reporting messages	3	Destination unreachable
	4	Source quench
	11	Time exceeded
	12	Parameter problem
	5	Redirection
Query messages	8 or 0	Echo request or reply
	13 or 14	Timestamp request or reply

Message Format

An ICMP message has an 8-byte header and a variable-size data section. Although the general format of the header is different for each message type, the first 4 bytes are common to all. As Figure 9.3 shows, the first field, ICMP type, defines the type of the message. The code field specifies the reason for the particular message type. The last common field is the checksum field (to be discussed later in the chapter). The rest of the header is specific for each message type.

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

Figure 9.3 *General format of ICMP messages*



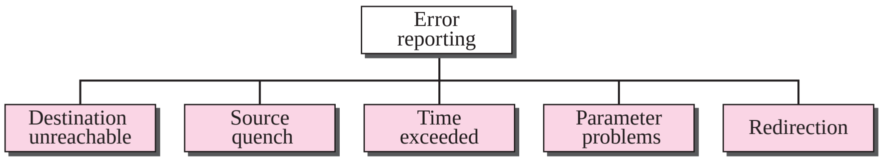
Error Reporting Messages

One of the main responsibilities of ICMP is to report errors. Although technology has produced increasingly reliable transmission media, errors still exist and must be handled. IP, as discussed in Chapter 7, is an unreliable protocol. This means that error checking and error control are not a concern of IP. ICMP was designed, in part, to compensate for this shortcoming. However, ICMP does not correct errors, it simply reports them. Error correction is left to the higher-level protocols. Error messages are always sent to the original source because the only information available in the datagram about the route is the source and destination IP addresses. ICMP uses the source IP address to send the error message to the source (originator) of the datagram.

ICMP always reports error messages to the original source.

Five types of errors are handled: destination unreachable, source quench, time exceeded, parameter problems, and redirection (see Figure 9.4).

Figure 9.4 *Error-reporting messages*



The following are important points about ICMP error messages:

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

Note that all error messages contain a data section that includes the IP header of the original datagram plus the first 8 bytes of data in that datagram. The original datagram header is added to give the original source, which receives the error message, information about the datagram itself. The 8 bytes of data are included because, as we will see in Chapters 14 and 15 on UDP and TCP protocols, the first 8 bytes provide information about the port numbers (UDP and TCP) and sequence number (TCP). This information is needed so the source can inform the protocols (TCP or UDP) about the error. ICMP forms an error packet, which is then encapsulated in an IP datagram (see Figure 9.5).

Destination Unreachable

When a router cannot route a datagram or a host cannot deliver a datagram, the datagram is discarded and the router or the host sends a **destination-unreachable message** back to the source host that initiated the datagram. Figure 9.6 shows the format of the

Figure 9.5 *Contents of data field for the error messages*

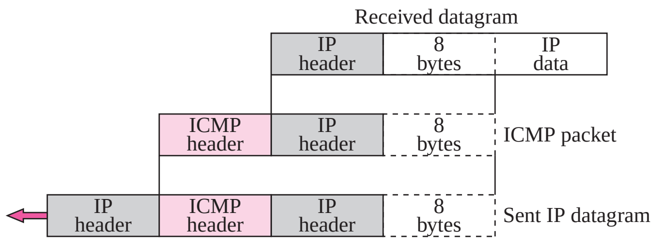
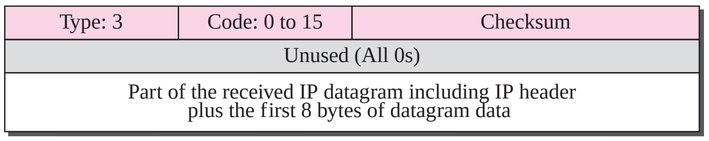


Figure 9.6 *Destination-unreachable format*


Type: 3
Code: 0 to 15
Checksum
Unused (All 0s)
Part of the received IP datagram including IP header plus the first 8 bytes of datagram data

destination-unreachable message. The code field for this type specifies the reason for discarding the datagram:

- **Code 0.** The network is unreachable, possibly due to hardware failure.
- **Code 1.** The host is unreachable. This can also be due to hardware failure.
- **Code 2.** The protocol is unreachable. An IP datagram can carry data belonging to higher-level protocols such as UDP, TCP, and OSPF. If the destination host receives a datagram that must be delivered, for example, to the TCP protocol, but the TCP protocol is not running at the moment, a code 2 message is sent.
- **Code 3.** The port is unreachable. The application program (process) that the datagram is destined for is not running at the moment.
- **Code 4.** Fragmentation is required, but the DF (do not fragment) field of the datagram has been set. In other words, the sender of the datagram has specified that the datagram not be fragmented, but routing is impossible without fragmentation.
- **Code 5.** Source routing cannot be accomplished. In other words, one or more routers defined in the source routing option cannot be visited.
- **Code 6.** The destination network is unknown. This is different from code 0. In code 0, the router knows that the destination network exists, but it is unreachable at the moment. For code 6, the router has no information about the destination network.
- **Code 7.** The destination host is unknown. This is different from code 1. In code 1, the router knows that the destination host exists, but it is unreachable at the moment. For code 7, the router is unaware of the existence of the destination host.

- **Code 8.** The source host is isolated.
- **Code 9.** Communication with the destination network is administratively prohibited.
- **Code 10.** Communication with the destination host is administratively prohibited.
- **Code 11.** The network is unreachable for the specified type of service. This is different from code 0. Here the router can route the datagram if the source had requested an available type of service.
- **Code 12.** The host is unreachable for the specified type of service. This is different from code 1. Here the router can route the datagram if the source had requested an available type of service.
- **Code 13.** The host is unreachable because the administrator has put a filter on it.
- **Code 14.** The host is unreachable because the host precedence is violated. The message is sent by a router to indicate that the requested precedence is not permitted for the destination.
- **Code 15.** The host is unreachable because its precedence was cut off. This message is generated when the network operators have imposed a minimum level of precedence for the operation of the network, but the datagram was sent with a precedence below this level.

Note that destination-unreachable messages can be created either by a router or the destination host. Code 2 and code 3 messages can only be created by the destination host; the messages of the remaining codes can only be created by routers.

Destination-unreachable messages with codes 2 or 3 can be created only by the destination host. Other destination-unreachable messages can be created only by routers.

Note that even if a router does not report a destination-unreachable message, it does not necessarily mean that the datagram has been delivered. For example, if a datagram is traveling through an Ethernet network, there is no way that a router knows that the datagram has been delivered to the destination host or the next router because Ethernet does not provide any acknowledgment mechanism.

A router cannot detect all problems that prevent the delivery of a packet.

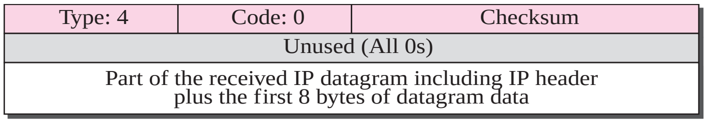
Source Quench

The IP protocol is a connectionless protocol. There is no communication between the source host, which produces the datagram, the routers, which forward it, and the destination host, which processes it. One of the ramifications of this absence of communication is the lack of *flow control* and *congestion control*.

There is no flow-control or congestion-control mechanism in the IP protocol.

The **source-quench message** in ICMP was designed to add a kind of flow control and congestion control to the IP. When a router or host discards a datagram due to congestion, it sends a source-quench message to the sender of the datagram. This message has two purposes. First, it informs the source that the datagram has been discarded. Second, it warns the source that there is congestion somewhere in the path and that the source should slow down (quench) the sending process. The source-quench format is shown in Figure 9.7.

Figure 9.7 *Source-quench format*


Type: 4
Code: 0
Checksum
Unused (All 0s)
Part of the received IP datagram including IP header
plus the first 8 bytes of datagram data

A source-quench message informs the source that a datagram has been discarded due to congestion in a router or the destination host. The source must slow down the sending of datagrams until the congestion is relieved.

There are some points that deserve more explanation. First, the router or destination host that has experienced the congestion sends one source-quench message for each discarded datagram to the source host. Second, there is no mechanism to tell the source that the congestion has been relieved and the source can resume sending datagrams at its previous rate. The source continues to lower the rate until no more source-quench messages are received. Third, the congestion can be created either by a one-to-one or many-to-one communication. In a one-to-one communication, a single high-speed host could create datagrams faster than a router or the destination host can handle. In this case, source-quench messages can be helpful. They tell the source to slow down. In a many-to-one communication, many sources create datagrams that must be handled by a router or the destination host. In this case, each source can be sending datagrams at different speeds, some of them at a low rate, others at a high rate. Here, the source-quench message may not be very useful. The router or the destination host has no clue which source is responsible for the congestion. It may drop a datagram from a very slow source instead of dropping the datagram from the source that has actually created the congestion.

One source-quench message is sent for each datagram that is discarded due to congestion.

Time Exceeded

The **time-exceeded message** is generated in two cases:

- First, as we saw in Chapter 6, routers use routing tables to find the next hop (next router) that must receive the packet. If there are errors in one or more routing tables, a packet can travel in a loop or a cycle, going from one router to the next or visiting a series of routers endlessly. As we saw in Chapter 7, each datagram contains a field called *time to live* that controls this situation. When a datagram visits a router, the value of this field is decremented by 1. When the time-to-live value reaches 0, after decrementing, the router discards the datagram. However, when the datagram is discarded, a time-exceeded message must be sent by the router to the original source.

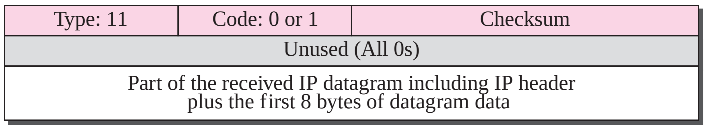
Whenever a router decrements a datagram with a time-to-live value to zero, it discards the datagram and sends a time-exceeded message to the original source.

- Second, a time-exceeded message is also generated when all fragments that make up a message do not arrive at the destination host within a certain time limit. When the first fragment arrives, the destination host starts a timer. If all the fragments have not arrived when the time expires, the destination discards all the fragments and sends a time-exceeded message to the original sender.

When the final destination does not receive all of the fragments in a set time, it discards the received fragments and sends a time-exceeded message to the original source.

Figure 9.8 shows the format of the time-exceeded message. Code 0 is used when the datagram is discarded by the router due to a time-to-live field value of zero. Code 1 is used when arrived fragments of a datagram are discarded because some fragments have not arrived within the time limit.

Figure 9.8 *Time-exceeded message format*


Type: 11
Code: 0 or 1
Checksum
Unused (All 0s)
Part of the received IP datagram including IP header plus the first 8 bytes of datagram data

In a time-exceeded message, code 0 is used only by routers to show that the value of the time-to-live field is zero. Code 1 is used only by the destination host to show that not all of the fragments have arrived within a set time.

Parameter Problem

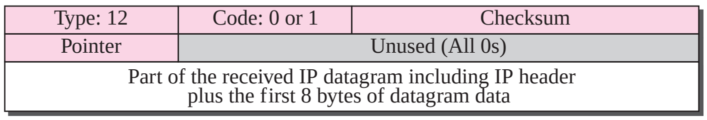
Any ambiguity in the header part of a datagram can create serious problems as the datagram travels through the Internet. If a router or the destination host discovers an ambiguous or missing value in any field of the datagram, it discards the datagram and sends a parameter-problem message back to the source.

A parameter-problem message can be created by a router or the destination host.

Figure 9.9 shows the format of the **parameter-problem message.** The code field in this case specifies the reason for discarding the datagram:

- **Code 0.** There is an error or ambiguity in one of the header fields. In this case, the value in the pointer field points to the byte with the problem. For example, if the value is zero, then the first byte is not a valid field.
- **Code 1.** The required part of an option is missing. In this case, the pointer is not used.

Figure 9.9 *Parameter-problem message format*

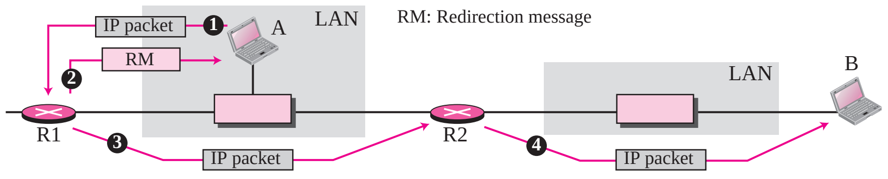

Type: 12
Code: 0 or 1
Checksum
Pointer
Unused (All 0s)
Part of the received IP datagram including IP header
plus the first 8 bytes of datagram data

Redirection

When a router needs to send a packet destined for another network, it must know the IP address of the next appropriate router. The same is true if the sender is a host. Both routers and hosts then must have a routing table to find the address of the router or the next router. Routers take part in the routing update process as we will see in Chapter 11 and are supposed to be updated constantly. Routing is dynamic.

However, for efficiency, hosts do not take part in the routing update process because there are many more hosts in an internet than routers. Updating the routing tables of hosts dynamically produces unacceptable traffic. The hosts usually use static routing. When a host comes up, its routing table has a limited number of entries. It usually knows only the IP address of one router, the default router. For this reason, the host may send a datagram, which is destined for another network, to the wrong router. In this case, the router that receives the datagram will forward the datagram to the correct router. However, to update the routing table of the host, it sends a redirection message to the host. This concept of redirection is shown in Figure 9.10. Host A wants to send a datagram to host B. Router R2 is obviously the most efficient routing choice, but host A did not choose router R2. The datagram goes to R1 instead. R1, after consulting its table, finds that the packet should have gone to R2. It sends the packet to R2 and, at the same time, sends a redirection message to host A. Host A's routing table can now be updated.

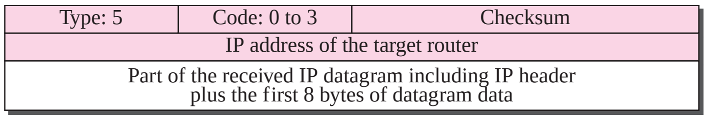
Figure 9.10 *Redirection concept*



A host usually starts with a small routing table that is gradually augmented and updated. One of the tools to accomplish this is the redirection message.

The format of the **redirection message** is shown in Figure 9.11. Note that the IP address of the appropriate target is given in the second row.

Figure 9.11 *Redirection message format*



Type: 5	Code: 0 to 3	Checksum
IP address of the target router		
Part of the received IP datagram including IP header plus the first 8 bytes of datagram data		

Although the redirection message is considered an error-reporting message, it is different from other error messages. The router does not discard the datagram in this case; it is sent to the appropriate router. The code field for the redirection message narrows down the redirection:

- ❑ **Code 0.** Redirection for a network-specific route.
- ❑ **Code 1.** Redirection for a host-specific route.
- ❑ **Code 2.** Redirection for a network-specific route based on a specified type of service.
- ❑ **Code 3.** Redirection for a host-specific route based on a specified type of service.

A redirection message is sent from a router to a host on the same local network.

Query Messages

In addition to error reporting, ICMP can also diagnose some network problems. This is accomplished through the query messages. A group of five different pairs of messages have been designed for this purpose, but three of these pairs are deprecated today, as we discuss later in the section. Only two pairs are used today: echo request and replay and timestamp request and replay. In this type of ICMP message, a node sends a message that is answered in a specific format by the destination node.

Echo Request and Reply

The **echo-request** and **echo-reply messages** are designed for diagnostic purposes. Network managers and users utilize this pair of messages to identify network problems. The combination of echo-request and echo-reply messages determines whether two systems (hosts or routers) can communicate with each other.

A host or router can send an echo-request message to another host or router. The host or router that receives an echo-request message creates an echo-reply message and returns it to the original sender.

An echo-request message can be sent by a host or router. An echo-reply message is sent by the host or router that receives an echo-request message.

The echo-request and echo-reply messages can be used to determine if there is communication at the IP level. Because ICMP messages are encapsulated in IP datagrams, the receipt of an echo-reply message by the machine that sent the echo request is proof that the IP protocols in the sender and receiver are communicating with each other using the IP datagram. Also, it is proof that the intermediate routers are receiving, processing, and forwarding IP datagrams.

Echo-request and echo-reply messages can be used by network managers to check the operation of the IP protocol.

The echo-request and echo-reply messages can also be used by a host to see if another host is reachable. At the user level, this is done by invoking the packet Internet groper (ping) command. Today, most systems provide a version of the *ping* command that can create a series (instead of just one) of echo-request and echo-reply messages, providing statistical information. We will see the use of this program at the end of the chapter.

Echo-request and echo-reply messages can test the reachability of a host. This is usually done by invoking the ping command.

Echo request, together with echo reply, can determine whether or not a node is functioning properly. The node to be tested is sent an echo-request message. The optional data field contains a message that must be repeated exactly by the responding node in its echo-reply message. Figure 9.12 shows the format of the echo-reply and echo-request message. The identifier and sequence number fields are not formally defined by the protocol and can be used arbitrarily by the sender. The identifier is often the same as the process ID.

Timestamp Request and Reply

Two machines (hosts or routers) can use the **timestamp-request** and **timestamp-reply messages** to determine the round-trip time needed for an IP datagram to travel between

Figure 9.12 *Echo-request and echo-reply messages*

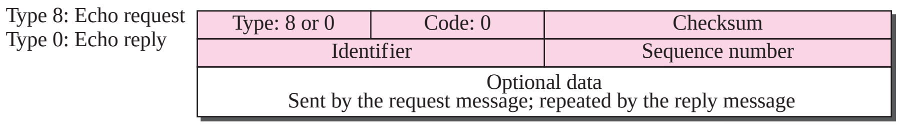
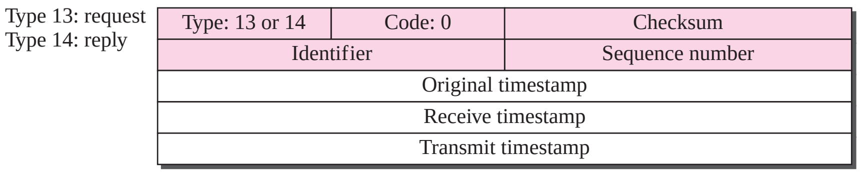


Figure 9.13 *Timestamp-request and timestamp-reply message format*



them. It can also be used to synchronize the clocks in two machines. The format of these two messages is shown in Figure 9.13.

The three timestamp fields are each 32 bits long. Each field can hold a number representing time measured in milliseconds from midnight in Universal Time (formerly called Greenwich Mean Time). (Note that 32 bits can represent a number between 0 and 4,294,967,295, but a timestamp in this case cannot exceed $86{,}400{,}000 = 24 \times 60 \times 60 \times 1000$.)

The source creates a timestamp-request message. The source fills the *original timestamp* field with the Universal Time shown by its clock at departure time. The other two timestamp fields are filled with zeros.

The destination creates the timestamp-reply message. The destination copies the original timestamp value from the request message into the same field in its reply message. It then fills the *receive timestamp* field with the Universal Time shown by its clock at the time the request was received. Finally, it fills the *transmit timestamp* field with the Universal Time shown by its clock at the time the reply message departs.

The timestamp-request and timestamp-reply messages can be used to compute the one-way or round-trip time required for a datagram to go from a source to a destination and then back again. The formulas are

sending time = receive timestamp − original timestamp
receiving time = returned time − transmit timestamp
round-trip time = sending time + receiving time

Note that the sending and receiving time calculations are accurate only if the two clocks in the source and destination machines are synchronized. However, the round-trip calculation is correct even if the two clocks are not synchronized because each clock contributes twice to the round-trip calculation, thus canceling any difference in synchronization.

Timestamp-request and timestamp-reply messages can be used to calculate the round-trip time between a source and a destination machine even if their clocks are not synchronized.

For example, given the following information:

original timestamp: 46	**receive timestamp: 59**
transmit timestamp: 60	**return time: 67**

We can calculate the round-trip time to be 20 milliseconds:

sending time = 59 − 46 = 13 milliseconds
receiving time = 67 − 60 = 7 milliseconds
round-trip time = 13 + 7 = 20 milliseconds

Given the actual one-way time, the timestamp-request and timestamp-reply messages can also be used to synchronize the clocks in two machines using the following formula:

Time difference = receive timestamp − (original timestamp field + one-way time duration)

The one-way time duration can be obtained either by dividing the round-trip time duration by two (if we are sure that the sending time is the same as the receiving time) or by other means. For example, we can tell that the two clocks in the previous example are 3 milliseconds out of synchronization because

Time difference = 59 − (46 + 10) = 3

The timestamp-request and timestamp-reply messages can be used to synchronize two clocks in two machines if the exact one-way time duration is known.

Deprecated Messages

Three pairs of messages are declared obsolete by IETF:

1. *Information request and replay* messages are not used today because their duties are done by Address Resolution Protocol (ARP) discussed in Chapter 8.
2. *Address mask request and reply* messages are not used today because their duties are done by Dynamic Host Configuration Protocol (DHCP), discussed in Chapter 8.
3. *Router solicitation and advertisment* messages are not used today because their duties are done by Dynamic Host Configuration Protocol (DHCP), discussed in Chapter 18.

Checksum

In Chapter 7, we learned the concept and idea of the checksum. In ICMP the checksum is calculated over the entire message (header and data).

Checksum Calculation

The sender follows these steps using one's complement arithmetic:

1. The checksum field is set to zero.
2. The sum of all the 16-bit words (header and data) is calculated.
3. The sum is complemented to get the checksum.
4. The checksum is stored in the checksum field.

Checksum Testing

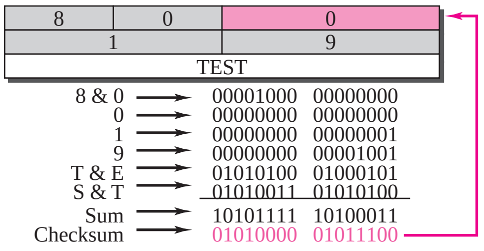
The receiver follows these steps using one's complement arithmetic:

1. The sum of all words (header and data) is calculated.
2. The sum is complemented.
3. If the result obtained in step 2 is 16 0s, the message is accepted; otherwise, it is rejected.

Example 9.1

Figure 9.14 shows an example of checksum calculation for a simple echo-request message (see Figure 9.12). We randomly chose the identifier to be 1 and the sequence number to be 9. The message is divided into 16-bit (2-byte) words. The words are added together and the sum is complemented. Now the sender can put this value in the checksum field.

Figure 9.14 *Example of checksum calculation*



9.3 DEBUGGING TOOLS

There are several tools that can be used in the Internet for debugging. We can find if a host or router is alive and running. We can trace the route of a packet. We introduce two tools that use ICMP for debugging: *ping* and *traceroute*.We will introduce more tools in future chapters after we have discussed the corresponding protocols.

Ping

We can use the **ping** program to find if a host is alive and responding. We used the *ping* program in Chapter 7 to simulate the record route option. We discuss *ping* in more detail to see how it uses ICMP packets.

The source host sends ICMP echo request messages (type: 8, code: 0); the destination, if alive, responds with ICMP echo reply messages. The *ping* program sets the identifier field in the echo request and reply message and starts the sequence number from 0; this number is incremented by one each time a new message is sent. Note that *ping* can calculate the round-trip time. It inserts the sending time in the data section of the message. When the packet arrives it subtracts the arrival time from the departure time to get the **round-trip time** (RTT).

Example 9.2

We use the *ping* program to test the server fhda.edu. The result is shown below:

```
$ ping fhda.edu
PING fhda.edu (153.18.8.1)    56 (84)  bytes of data.
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=0     ttl=62     time=1.91 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=1     ttl=62     time=2.04 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=2     ttl=62     time=1.90 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=3     ttl=62     time=1.97 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=4     ttl=62     time=1.93 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=5     ttl=62     time=2.00 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=6     ttl=62     time=1.94 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=7     ttl=62     time=1.94 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=8     ttl=62     time=1.97 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=9     ttl=62     time=1.89 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=10    ttl=62     time=1.98 ms

--- fhda.edu ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10103 ms
rtt min/avg/max = 1.899/1.955/2.041 ms
```

The *ping* program sends messages with sequence numbers starting from 0. For each probe it gives us the RTT time. The TTL (time to live) field in the IP datagram that encapsulates an ICMP message has been set to 62, which means the packet cannot travel more than 62 hops. At the beginning, *ping* defines the number of data bytes as 56 and the total number of bytes as 84. It is obvious that if we add 8 bytes of ICMP header and 20 bytes of IP header to 56, the result is 84. However, note that in each probe *ping* defines the number of bytes as 64. This is the total number of bytes in the ICMP packet (56 + 8). The *ping* program continues to send messages if we do not stop it using the interrupt key (ctrl + c, for example). After it is interrupted, it prints the statistics of the probes. It tells us the number of packets sent, the number of packets received, the total time, and the RTT minimum, maximum, and average. Some systems may print more information.

Example 9.3

For the second example, we want to know if the adelphia.net mail server is alive and running. The result is shown below: Note that in this case, we sent 14 packets, but only 13 have been returned. We may have interrupted the program before the last packet, with sequence number 13, was returned.

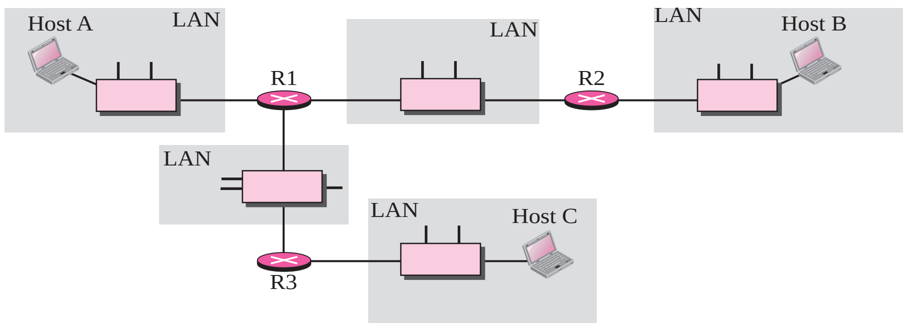
```
$ ping mail.adelphia.net
PING mail.adelphia.net (68.168.78.100) 56(84) bytes of data.
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=0   ttl=48   time=85.4 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=1   ttl=48   time=84.6 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=2   ttl=48   time=84.9 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=3   ttl=48   time=84.3 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=4   ttl=48   time=84.5 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=5   ttl=48   time=84.7 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=6   ttl=48   time=84.6 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=7   ttl=48   time=84.7 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=8   ttl=48   time=84.4 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=9   ttl=48   time=84.2 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=10  ttl=48   time=84.9 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=11  ttl=48   time=84.6 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=12  ttl=48   time=84.5 ms

--- mail.adelphia.net ping statistics ---
14 packets transmitted, 13 received, 7% packet loss, time 13129 ms
rtt min/avg/max/mdev = 84.207/84.694/85.469
```

Traceroute

The ***traceroute*** program in UNIX or ***tracert*** in Windows can be used to trace the route of a packet from the source to the destination. Let us show the idea of the **traceroute** program using Figure 9.15.

Figure 9.15 *The traceroute program operation*



We have seen an application of the *traceroute* program to simulate the loose source route and strict source route options of an IP datagram in the previous chapter. We use this program in conjunction with ICMP packets in this chapter. The program elegantly uses two ICMP messages, time exceeded and destination unreachable, to find the route of a packet. This is a program at the application level that uses the services of UDP (see Chapter 14).

Given the topology, we know that a packet from host A to host B travels through routers R1 and R2. However, most of the time, we are not aware of this topology. There could be several routes from A to B. The *traceroute* program uses the ICMP messages and the TTL (time to live) field in the IP packet to find the route.

1. The traceroute program uses the following steps to find the address of the router R1 and the round trip time between host A and router R1. The *traceroute* program repeats steps a to c three times to get a better average round-trip time. The first trip time may be much longer than the second or third because it takes time for the ARP program to find the physical address of router R1. For the second and third trip, ARP has the address in its cache.
   a. The traceroute application at host A sends a packet to destination B using UDP; the message is encapsulated in an IP packet with a TTL value of 1. The program notes the time the packet is sent.
   b. Router R1 receives the packet and decrements the value of TTL to 0. It then discards the packet (because TTL is 0). The router, however, sends a time-exceeded ICMP message (type: 11, code: 0) to show that the TTL value is 0 and the packet was discarded.
   c. The traceroute program receives the ICMP messages and uses the source address of the IP packet encapsulating ICMP to find the IP address of router R1. The program also makes note of the time the packet has arrived. The difference between this time and the time at step a is the round-trip time.
2. The traceroute program repeats the previous steps to find the address of router R2 and the round-trip time between host A and router R2. However, in this step, the value of TTL is set to 2. So router R1 forwards the message, while router R2 discards it and sends a time-exceeded ICMP message.
3. The traceroute program repeats the previous step to find the address of host B and the round-trip time between host A and host B. When host B receives the packet, it decrements the value of TTL, but it does not discard the message since it has reached its final destination. How can an ICMP message be sent back to host A? The traceroute program uses a different strategy here. The destination port of the UDP packet is set to one that is not supported by the UDP protocol. When host B receives the packet, it cannot find an application program to accept the delivery. It discards the packet and sends an ICMP destination-unreachable message (type: 3, code: 3) to host A. Note that this situation does not happen at router R1 or R2 because a router does not check the UDP header. The traceroute program records the destination address of the arrived IP datagram and makes note of the round-trip time. Receiving the destination-unreachable message with a code value 3 is an indication that the whole route has been found and there is no need to send more packets.

Example 9.4

We use the traceroute program to find the route from the computer voyager.deanza.edu to the server fhda.edu. The following shows the result.

```
$ traceroute fhda.edu
traceroute to fhda.edu      (153.18.8.1), 30 hops max, 38 byte packets
1 Dcore.fhda.edu           (153.18.31.25)     0.995 ms     0.899 ms     0.878 ms
2 Dbackup.fhda.edu         (153.18.251.4)     1.039 ms     1.064 ms     1.083 ms
3  tiptoe.fhda.edu          (153.18.8.1)      1.797 ms     1.642 ms     1.757 ms
```

The unnumbered line after the command shows that the destination is 153.18.8.1. The TTL value is 30 hops. The packet contains 38 bytes: 20 bytes of IP header, 8 bytes of UDP header, and 10 bytes of application data. The application data is used by traceroute to keep track of the packets. The first line shows the first router visited. The router is named Dcore.fhda.edu with IP address 153.18.31.254. The first round-trip time was 0.995 milliseconds, the second was 0.899 milliseconds, and the third was 0.878 milliseconds. The second line shows the second router visited. The router is named Dbackup.fhda.edu with IP address 153.18.251.4. The three round-trip times are also shown. The third line shows the destination host. We know that this is the destination host because there are no more lines. The destination host is the server fhda.edu, but it is named tiptoe.fhda.edu with the IP address 153.18.8.1. The three round-trip times are also shown.

Example 9.5

In this example, we trace a longer route, the route to xerox.com

```
$ traceroute xerox.com
traceroute to xerox.com (13.1.64.93), 30 hops max, 38 byte packets
1  Dcore.fhda.edu      (153.18.31.254)     0.622 ms     0.891 ms    0.875 ms
2  Ddmz.fhda.edu       (153.18.251.40)     2.132 ms     2.266 ms    2.094 ms
3  Cinic.fhda.edu      (153.18.253.126)    2.110 ms     2.145 ms    1.763 ms
4  cenic.net           (137.164.32.140)    3.069 ms     2.875 ms    2.930 ms
5  cenic.net           (137.164.22.31)     4.205 ms     4.870 ms    4.197 ms
6  cenic.net           (137.164.22.167)    4.250 ms     4.159 ms    4.078 ms
7  cogentco.com        (38.112.6.225)      5.062 ms     4.825 ms    5.020 ms
8  cogentco.com        (66.28.4.69)        6.070 ms     6.207 ms    5.653 ms
9  cogentco.com        (66.28.4.94)        6.070 ms     5.928 ms    5.499 ms
10  cogentco.com       (154.54.2.226)      6.545 ms     6.399 ms    6.535 ms
11  sbcglo             (151.164.89.241)    6.379 ms     6.370 ms    6.379 ms
12  sbcglo             (64.161.1.45)       6.908 ms     6.748 ms    7.359 ms
13  sbcglo             (64.161.1.29)       7.023 ms     7.040 ms    6.734 ms
14  snfc21.pbi.net     (151.164.191.49)    7.656 ms     7.129 ms    6.866 ms
15  sbcglobal.net      (151.164.243.58)    7.844 ms     7.545 ms    7.353 ms
16  pacbell.net        (209.232.138.114)   9.857 ms     9.535 ms    9.603 ms
17  209.233.48.223     (209.233.48.223)   10.634 ms    10.771 ms   10.592 ms
18  alpha.Xerox.COM    (13.1.64.93)       10.922 ms    11.048 ms   10.922 ms
```

Here there are 17 hops between source and destination. Note that some round-trip times look unusual. It could be that a router is too busy to process the packet immediately.

Example 9.6

An interesting point is that a host can send a traceroute packet to itself. This can be done by specifying the host as the destination. The packet goes to the loopback address as we expect.

```
$ traceroute voyager.deanza.edu
traceroute to voyager.deanza.edu   (127.0.0.1), 30 hops max, 38 byte packets
1  voyager          (127.0.0.1)          0.178 ms          0.086 ms          0.055 ms
```

Example 9.7

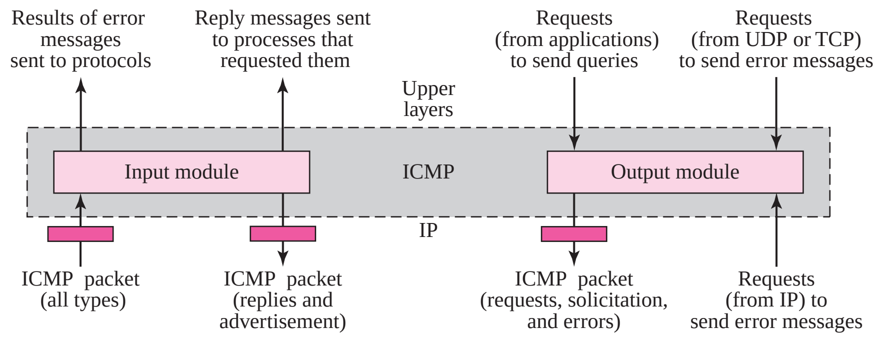
Finally, we use the traceroute program to find the route between fhda.edu and mhhe.com (McGraw-Hill server). We notice that we cannot find the whole route. When traceroute does not receive a response within 5 seconds, it prints an asterisk to signify a problem (not the case in this example), and then tries the next hop.

```
$ traceroute mhhe.com
traceroute to mhhe.com (198.45.24.104), 30 hops max, 38 byte packets
1   Dcore.fhda.edu        (153.18.31.254)      1.025 ms    0.892 ms    0.880 ms
2   Ddmz.fhda.edu         (153.18.251.40)      2.141 ms    2.159 ms    2.103 ms
3   Cinic.fhda.edu        (153.18.253.126)     2.159 ms    2.050 ms    1.992 ms
4   cenic.net             (137.164.32.140)     3.220 ms    2.929 ms    2.943 ms
5   cenic.net             (137.164.22.59)      3.217 ms    2.998 ms    2.755 ms
6   SanJose1.net          (209.247.159.109)   10.653 ms   10.639 ms   10.618 ms
7   SanJose2.net          (64.159.2.1)        10.804 ms   10.798 ms   10.634 ms
8   Denver1.Level3.net    (64.159.1.114)      43.404 ms   43.367 ms   43.414 ms
9   Denver2.Level3.net    (4.68.112.162)      43.533 ms   43.290 ms   43.347 ms
10  unknown               (64.156.40.134)     55.509 ms   55.462 ms   55.647 ms
11  mcleodusa1.net        (64.198.100.2)      60.961 ms   55.681 ms   55.461 ms
12  mcleodusa2.net        (64.198.101.202)    55.692 ms   55.617 ms   55.505 ms
13  mcleodusa3.net        (64.198.101.142)    56.059 ms   55.623 ms   56.333 ms
14  mcleodusa4.net        (209.253.101.178)  297.199 ms  192.790 ms  250.594 ms
15  eppg.com              (198.45.24.246)     71.213 ms   70.536 ms   70.663 ms
16  ...                   ...                 ...         ...         ...
```

9.4 ICMP PACKAGE

To give an idea of how ICMP can handle the sending and receiving of ICMP messages, we present our version of an ICMP package made of two modules: an input module and an output module. Figure 9.16 shows these two modules.

Figure 9.16 *ICMP package*


Results of error messages sent to protocols
Reply messages sent to processes that requested them
Upper layers
Requests (from applications) to send queries
Requests (from UDP or TCP) to send error messages
Input module
ICMP
Output module
IP
ICMP packet (all types)
ICMP packet (replies and advertisement)
ICMP packet (requests, solicitation, and errors)
Requests (from IP) to send error messages

Input Module

The input module handles all received ICMP messages. It is invoked when an ICMP packet is delivered to it from the IP layer. If the received packet is a request, the module creates a reply and sends it out. If the received packet is a redirection message, the module uses the information to update the routing table. If the received packet is an error message, the module informs the protocol about the situation that caused the error. The pseudocode is shown below:

Table 9.2 *Input Module*

```
ICMP_Input_module (ICMP_Packet)
{
   If (the type is a request)
   {
        Create a reply
        Send the reply
   }
   If (the type defines a redirection)
   {
       Modify the routing table
   }
   If (the type defines other error messages)
   {
       Inform the appropriate source protocol
   }
   Return
}
```

Output Module

The output module is responsible for creating request, solicitation, or error messages requested by a higher level or the IP protocol. The module receives a demand from IP, UDP, or TCP to send one of the ICMP error messages. If the demand is from IP, the output module must first check that the request is allowed. Remember, an ICMP message cannot be created for four situations: an IP packet carrying an ICMP error message, a fragmented IP packet, a multicast IP packet, or an IP packet having IP address 0.0.0.0 or 127.X.Y. Z. The output module may also receive a demand from an application program to send one of the ICMP request messages. The pseudocode is shown in Table 9.3.

Table 9.3 *Output Module*

```
ICMP_Output_Module (demand)
{
   If (the demand defines an error message)
   {
```

Table 9.3 *Output Module (continued)*

```
            If (demand comes from IP AND is forbidden)
            {
                 Return
            }
            If (demand is a valid redirection message)
            {
                 Return
            }
            Create an error message
   If (demand defines a request)
   {
        Create a request message
   }
   Send the message
   Return
}
```

9.5 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], and [Ste 94].

RFCs

Several RFCs discuss ICMP including RFC 792, RFC 950, RFC 956, RFC 957, RFC 1016, RFC 1122, RFC 1256, RFC 1305, and RFC 1987.

9.6 KEY TERMS

destination-unreachable message
echo-reply messages
error-reporting message
echo-request message
Internet Control Message Protocol (ICMP)
parameter-problem message
ping
query message
redirection message
round-trip time (RTT)
source-quench message
time-exceeded message
timestamp-reply message
timestamp-request message
traceroute