6

ICMP: Internet Control Message Protocol

6.1 Introduction

ICMP is often considered part of the IP layer. It communicates error messages and other conditions that require attention. ICMP messages are usually acted on by either the IP layer or the higher layer protocol (TCP or UDP). Some ICMP messages cause errors to be returned to user processes.

ICMP messages are transmitted within IP datagrams, as shown in Figure 6.1.

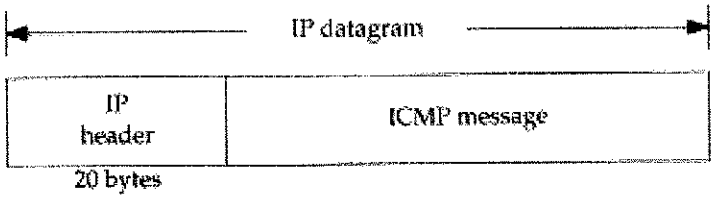

Figure 6.1 ICMP messages encapsulated within an IP datagram.

RFC 792 [Postel 1981b] contains the official specification of ICMP.

Figure 6.2 shows the format of an ICMP message. The first 4 bytes have the same format for all messages, but the remainder differs from one message to the next. We'll show the exact format of each message when we describe it.

There are 15 different values for the *type* field, which identify the particular ICMP message. Some types of ICMP messages then use different values of the *code* field to further specify the condition.

The *checksum* field covers the entire ICMP message. The algorithm used is the same as we described for the IP header checksum in Section 3.2. The ICMP checksum is required.

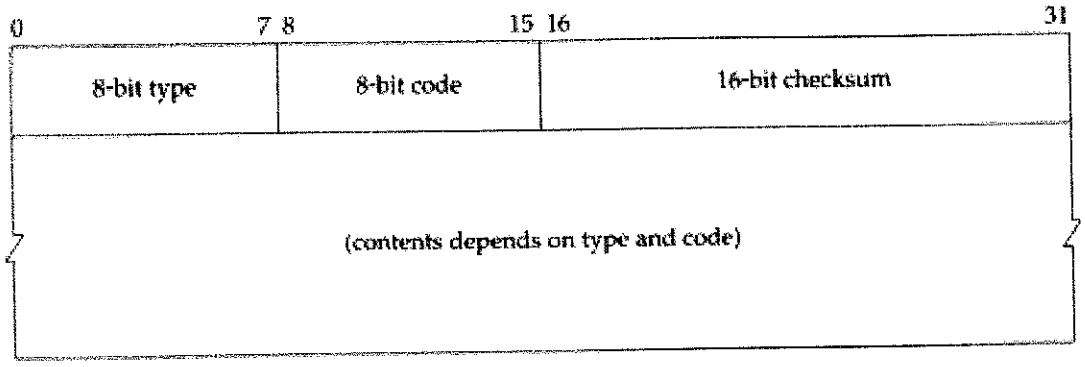

Figure 6.2 ICMP message.

In this chapter we talk about ICMP messages in general and a few in detail: address mask request and reply, timestamp request and reply, and port unreachable. We discuss the echo request and reply messages in detail with the Ping program in Chapter 7, and we discuss the ICMP messages dealing with IP routing in Chapter 9.

6.2 ICMP Message Types

Figure 6.3 lists the different ICMP message types, as determined by the *type* field and *code* field in the ICMP message.

The final two columns in this figure specify whether the ICMP message is a query message or an error message. We need to make this distinction because ICMP error messages are sometimes handled specially. For example, an ICMP error message is never generated in response to an ICMP error message. (If this were not the rule, we could end up with scenarios where an error generates an error, which generates an error, and so on, indefinitely.)

When an ICMP error message is sent, the message always contains the IP header and the first 8 bytes of the IP datagram that caused the ICMP error to be generated. This lets the receiving ICMP module associate the message with one particular protocol (TCP or UDP from the protocol field in the IP header) and one particular user process (from the TCP or UDP port numbers that are in the TCP or UDP header contained in the first 8 bytes of the IP datagram). We'll show an example of this in Section 6.5.

An ICMP error message is never generated in response to

1. An ICMP error message. (An ICMP error message may, however, be generated in response to an ICMP query message.)
2. A datagram destined to an IP broadcast address (Figure 3.9) or an IP multicast address (a class D address, Figure 1.5).
3. A datagram sent as a link-layer broadcast.
4. A fragment other than the first. (We describe fragmentation in Section 11.5.)

type	*code*	Description	Query	Error
0	0	echo reply (Ping reply, Chapter 7)	•	
3		destination unreachable:		
	0	network unreachable (Section 9.3)		•
	1	host unreachable (Section 9.3)		•
	2	protocol unreachable		•
	3	port unreachable (Section 6.5)		•
	4	fragmentation needed but don't-fragment bit set (Section 11.6)		•
	5	source route failed (Section 8.5)		•
	6	destination network unknown		•
	7	destination host unknown		•
	8	source host isolated (obsolete)		•
	9	destination network administratively prohibited		•
	10	destination host administratively prohibited		•
	11	network unreachable for TOS (Section 9.3)		•
	12	host unreachable for TOS (Section 9.3)		•
	13	communication administratively prohibited by filtering		•
	14	host precedence violation		•
	15	precedence cutoff in effect		•
4	0	source quench (elementary flow control, Section 11.11)		•
5		redirect (Section 9.5):		
	0	redirect for network		•
	1	redirect for host		•
	2	redirect for type-of-service and network		•
	3	redirect for type-of-service and host		•
8	0	echo request (Ping request, Chapter 7)	•	
9	0	router advertisement (Section 9.6)	•	
10	0	router solicitation (Section 9.6)	•	
11		time exceeded:		
	0	time-to-live equals 0 during transit (Traceroute, Chapter 8)		•
	1	time-to-live equals 0 during reassembly (Section 11.5)		•
12		parameter problem:		
	0	IP header bad (catchall error)		•
	1	required option missing		•
13	0	timestamp request (Section 6.4)	•	
14	0	timestamp reply (Section 6.4)	•	
15	0	information request (obsolete)	•	
16	0	information reply (obsolete)	•	
17	0	address mask request (Section 6.3)	•	
18	0	address mask reply (Section 6.3)	•	

Figure 6.3 ICMP message types.

5. A datagram whose source address does not define a single host. This means the source address cannot be a zero address, a loopback address, a broadcast address, or a multicast address.

These rules are meant to prevent the *broadcast storms* that have occurred in the past when ICMP errors were sent in response to broadcast packets.

6.3 ICMP Address Mask Request and Reply

The ICMP address mask request is intended for a diskless system to obtain its subnet mask (Section 3.5) at bootstrap time. The requesting system broadcasts its ICMP request. (This is similar to a diskless system using RARP to obtain its IP address at bootstrap time.) An alternative method for a diskless system to obtain its subnet mask is the BOOTP protocol, which we describe in Chapter 16. Figure 6.4 shows the format of the ICMP address mask request and reply messages.

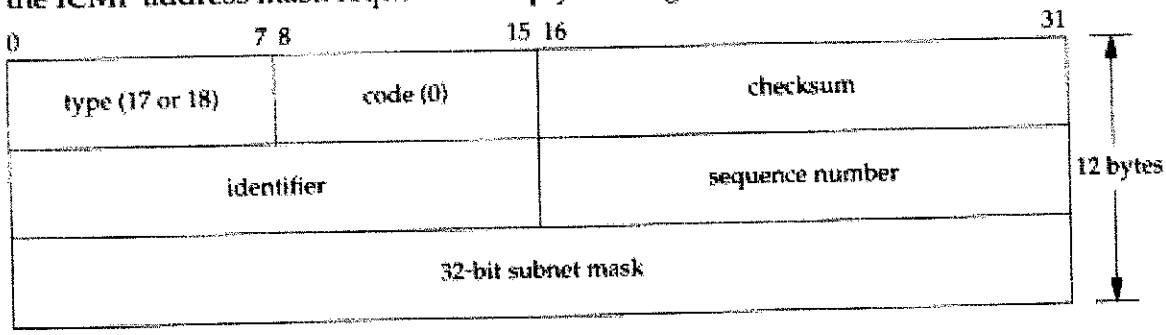

Figure 6.4 ICMP address mask request and reply messages.

The identifier and sequence number fields in the ICMP message can be set to anything the sender chooses, and these values are returned in the reply. This allows the sender to match replies with requests.

We can write a simple program (named `icmpaddrmask`) that issues an ICMP address mask request and prints all replies. Since normal usage is to send the request to the broadcast address, that's what we'll do. The destination address (140.252.13.63) is the broadcast address for the subnet 140.252.13.32 (Figure 3.12).

```
sun % icmpaddrmask 140.252.13.63
received mask = ffffffe0, from 140.252.13.33      from ourself
received mask = ffffffe0, from 140.252.13.35      from bsdi
received mask = ffff0000, from 140.252.13.34      from svr4
```

The first thing we note in this output is that the returned value from `svr4` is wrong. It appears that SVR4 is returning the general class B address mask, assuming no subnets, even though the interface on `svr4` has been configured with the correct subnet mask:

```
svr4 % ifconfig emd0
emd0: flags=23<UP,BROADCAST,NOTRAILERS>
        inet 140.252.13.34 netmask ffffffe0 broadcast 140.252.13.63
```

There is a bug in the SVR4 handling of the ICMP address mask request.

We'll watch this exchange on the host `bsdi` using `tcpdump`. The output is shown in Figure 6.5. We specify the `-e` option to see the hardware addresses.

```
1  0.0                  8:0:20:3:f6:42 ff:ff:ff:ff:ff:ff ip 60:
                        sun > 140.252.13.63: icmp: address mask request

2  0.00 (0.00)          0:0:c0:6f:2d:40 ff:ff:ff:ff:ff:ff ip 46:
                        bsdi > sun: icmp: address mask is 0xfffffe0

3  0.01 (0.01)          0:0:c0:c2:9b:26 8:0:20:3:f6:42 ip 60:
                        svr4 > sun: icmp: address mask is 0xffff0000
```

Figure 6.5 ICMP address mask request sent to broadcast address.

Note that the sending host, `sun`, receives an ICMP reply (the output line with the comment *from ourself* shown earlier), even though nothing is seen on the wire. This is a general characteristic of broadcasting: the sending host receives a copy of the broadcast packet through some internal loopback mechanism. Since by definition the term "broadcast" means *all* the hosts on the local network, it should include the sending host. (Referring to Figure 2.4 [p. 28] what is happening is that when the Ethernet driver recognizes that the destination address is the broadcast address, the packet is sent onto the network *and* a copy is made and passed to the loopback interface.)

Next, `bsdi` broadcasts the reply, while `svr4` sends the reply only to the requestor. Normally the reply should be unicast unless the source IP address of the request is 0.0.0.0, which it isn't in this example. Therefore, sending the reply to the broadcast address is a BSD/386 bug.

> The Host Requirements RFC says that a system must not send an address mask reply unless it is an authoritative agent for address masks. (To be an authoritative agent it must be specifically configured to send these replies. See Appendix E.) As we can see from this example, however, most host implementations send a reply if they get a request. Some hosts even send the wrong reply!

The final point is shown by the following example. We send an address mask request to our own IP address and to the loopback address:

```
sun % icmpaddrmask sun
received mask = ff000000, from 140.252.13.33

sun % icmpaddrmask localhost
received mask = ff000000, from 127.0.0.1
```

In both cases the returned address mask corresponds to the loopback address, the class A address 127.0.0.1. Again, referring to Figure 2.4 we see that IP datagrams sent to the host's own IP address (140.252.13.33 in this example) are actually sent to the loopback interface. The ICMP address mask reply must correspond to the subnet mask of the interface on which the request was received (since a multihomed host can have different subnet masks for each interface), and in both cases the request is received from the loopback interface.

6.4 ICMP Timestamp Request and Reply

The ICMP timestamp request allows a system to query another for the current time. The recommended value to be returned is the number of milliseconds since midnight, Coordinated Universal Time (UTC). (Older manuals refer to UTC as Greenwich Mean Time.) The nice feature of this ICMP message is that it provides millisecond resolution, whereas some other methods for obtaining the time from another host (such as the `rdate` command provided by some Unix systems) provide a resolution of seconds. The drawback is that only the time since midnight is returned—the caller must know the date from some other means.

Figure 6.6 shows the format of the ICMP timestamp request and reply messages.

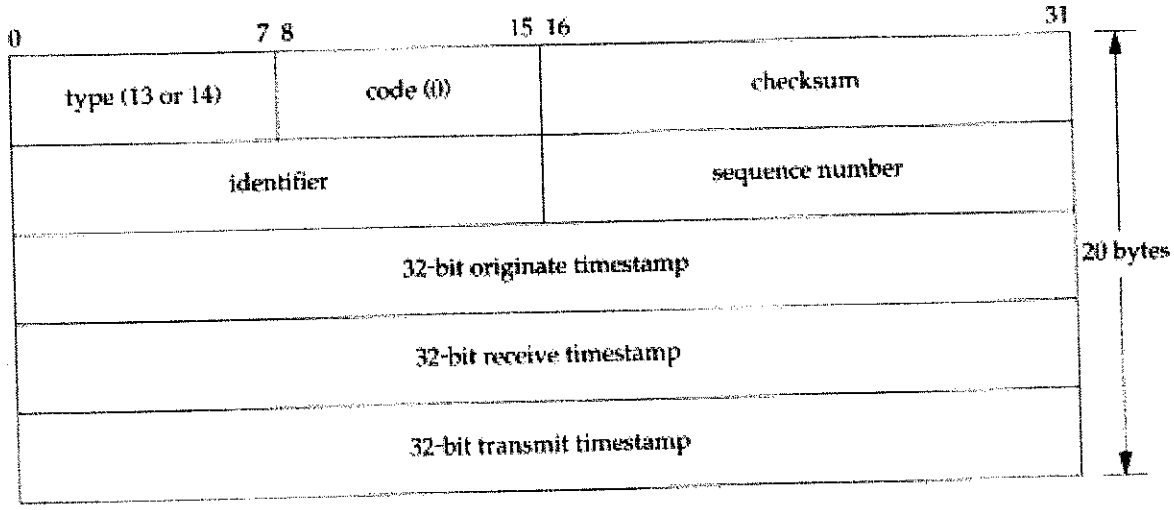

Figure 6.6 ICMP timestamp request and reply messages.

The requestor fills in the *originate* timestamp and sends the request. The replying system fills in the *receive* timestamp when it receives the request, and the *transmit* timestamp when it sends the reply. In actuality, however, most implementations set the latter two fields to the same value. (The reason for providing the three fields is to let the sender compute the time for the request to be sent, and separately compute the time for the reply to be sent.)

Examples

We can write a simple program (named `icmptime`) that sends an ICMP timestamp request to a host and prints the returned reply. We try it first on our small internet:

```
sun % icmptime bsdi
orig = 83573330, recv = 83573330, xmit = 83573330, rtt = 2 ms
difference = -6 ms

sun % icmptime bsdi
orig = 83577980, recv = 83577980, xmit = 83577980, rtt = 2 ms
difference = -7 ms
```

The program prints the three timestamps in the ICMP message: the originate (`orig`), receive (`recv`), and transmit (`xmit`) timestamps. As we can see in this and the following examples, all the hosts set the receive and transmit timestamps to the same value.

We also calculate the round-trip time (`rtt`), which is the time the reply is received minus the time the request was sent. The `difference` is the received timestamp minus the originate timestamp. Figure 6.7 shows the relationship between these values.

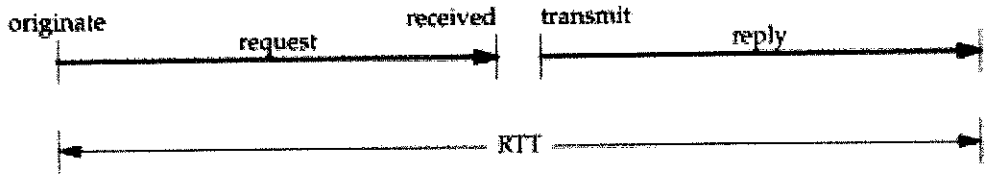

Figure 6.7 Relationship between values printed by our `icmptime` program.

If we believe the RTT and assume that one-half of the RTT is for the request, and the other half for the reply, then the sender's clock needs to be adjusted by `difference` minus one-half the RTT, to have the same time as the host being queried. In the preceding example, the clock on `bsdi` was 7 and 8 ms behind the clock on `sun`.

Since the timestamp values are the number of milliseconds past midnight, UTC, they should always be less than 86,400,000 ($24 \times 60 \times 60 \times 1000$). These examples were run just before 4:00 P.M. in a time zone that is 7 hours behind UTC, so the values being greater than 82,800,000 (2300 hours) makes sense.

If we run this program several times to the host `bsdi` we see that the final digit in the receive and transmit timestamp is always 0. This is because the software release (Version 0.9.4) only provides a 10-ms clock. (We describe this in Appendix B.)

If we run the program twice to the host `svr4` we see that the low-order three digits of the SVR4 timestamp are always 0:

```
sun % icmptime svr4
orig = 83588210, recv = 83588000, xmit = 83588000, rtt = 4 ms
difference = -210 ms

sun % icmptime svr4
orig = 83591547, recv = 83591000, xmit = 83591000, rtt = 4 ms
difference = -547 ms
```

For some reason SVR4 doesn't provide any millisecond resolution using the ICMP timestamp. This imprecision makes the calculated differences useless for subsecond adjustments.

If we try two other hosts on the 140.252.1 subnet, the results show that one clock differs from `sun`'s by 3.7 seconds, and the other by nearly 75 seconds:

```
sun % icmptime gemini
orig = 83601883, recv = 83598140, xmit = 83598140, rtt = 247 ms
difference = -3743 ms

sun % icmptime aix
orig = 83606768, recv = 83532183, xmit = 83532183, rtt = 253 ms
difference = -74585 ms
```

Another interesting example is to the router `gateway` (a Cisco router). It shows that when a system returns a nonstandard timestamp value (something other than milliseconds past midnight, UTC), it is supposed to turn on the high-order bit of the 32-bit timestamp. Our program detects this, and prints the receive and transmit timestamps in angle brackets (after turning off the high-order bit). Also, we can't calculate the difference between the originate and receive timestamps, since they're not the same units.

```
sun % icmptime gateway
orig = 83620811, recv = <4871036>, xmit = <4871036>, rtt = 220 ms

sun % icmptime gateway
orig = 83641007, recv = <4891232>, xmit = <4891232>, rtt = 213 ms
```

If we run our program to this host a few times it becomes obvious that the values do contain millisecond resolution and do count the number of milliseconds past some starting point, but the starting point is not midnight, UTC. (It could be a counter that's incremented every millisecond since the router was bootstrapped, for example.)

As a final example we'll compare `sun`'s clock with a system whose clock is known to be accurate—an NTP stratum 1 server. (We say more about NTP, the Network Time Protocol, below.)

```
sun % icmptime clock.llnl.gov
orig = 83662791, recv = 83662919, xmit = 83662919, rtt = 359 ms
difference = 128 ms

sun % icmptime clock.llnl.gov
orig = 83670425, recv = 83670559, xmit = 83670559, rtt = 345 ms
difference = 134 ms
```

If we calculate the difference minus one-half the RTT, this output indicates that the clock on `sun` is between 38.5 and 51.5 ms fast.

Alternatives

There are other ways to obtain the time and date.

1. We described the daytime service and time service in Section 1.12. The former returns the current time and date in a human readable form, a line of ASCII characters. We can test this service using the `telnet` command:

```
sun % telnet bsdi daytime
Trying 140.252.13.35 ...
Connected to bsdi.
Escape character is '^]'.
Wed Feb  3 16:38:33 1993
Connection closed by foreign host.
```

first three lines output are from the Telnet client
here's the daytime service output
this is also from the Telnet client

The time server, on the other hand, returns a 32-bit binary value with the number of seconds since midnight January 1, 1900, UTC. While this provides the date, the time value is in units of a second. (The `rdate` command that we mentioned earlier uses the TCP time service.)

2. Serious timekeepers use the Network Time Protocol (NTP) described in RFC 1305 [Mills 1992]. This protocol uses sophisticated techniques to maintain the clocks for a group of systems on a LAN or WAN to within millisecond accuracy. Anyone interested in precise timekeeping on computers should read this RFC.
3. The Open Software Foundation's (OSF) Distributed Computing Environment (DCE) defines a Distributed Time Service (DTS) that also provides clock synchronization between computers. [Rosenberg, Kenney, and Fisher 1992] provide additional details on this service.
4. Berkeley Unix systems provide the daemon `timed`(8) to synchronize the clocks of systems on a local area network. Unlike NTP and DTS, `timed` does not work across wide area networks.

6.5 ICMP Port Unreachable Error

The last two sections looked at ICMP query messages—the address mask and timestamp queries and replies. We'll now examine an ICMP error message, the port unreachable message, a subcode of the ICMP destination unreachable message, to see the additional information returned in an ICMP error message. We'll watch this using UDP (Chapter 11).

One rule of UDP is that if it receives a UDP datagram and the destination port does not correspond to a port that some process has in use, UDP responds with an ICMP port unreachable. We can force a port unreachable using the TFTP client. (We describe TFTP in Chapter 15.)

The well-known UDP port for the TFTP server to be reading from is 69. But most TFTP client programs allow us to specify a different port using the `connect` command. We use this to specify a port of 8888:

```
bsdi % tftp
tftp> connect svr4 8888          specify the hostname and port number
tftp> get temp.foo               try to fetch a file
Transfer timed out.              about 25 seconds later
tftp> quit
```

The `connect` command saves the name of the host to contact and the port number on that host, for when we later issue the `get` command. After typing the `get` command a UDP datagram is sent to port 8888 on host `svr4`. Figure 6.8 shows the `tcpdump` output for the exchange of packets that takes place.

Before the UDP datagram can be sent to `svr4` an ARP request is sent to determine its hardware address (line 1). The ARP reply (line 2) is returned and then the UDP datagram is sent (line 3). (We have left the ARP request–reply in this `tcpdump` output to remind us that this exchange may be required before the first IP datagram is sent from one host to the other. In future output we'll delete this exchange if it's not relevant to the topic being discussed.)

```
1   0.0                 arp who-has svr4 tell bsdi
2   0.002050 (0.0020)   arp reply svr4 is-at 0:0:c0:c2:9b:26

3   0.002723 (0.0007)   bsdi.2924 > svr4.8888: udp 20
4   0.006399 (0.0037)   svr4 > bsdi: icmp: svr4 udp port 8888 unreachable

5   5.000776 (4.9944)   bsdi.2924 > svr4.8888: udp 20
6   5.004304 (0.0035)   svr4 > bsdi: icmp: svr4 udp port 8888 unreachable

7   10.000887 (4.9966)  bsdi.2924 > svr4.8888: udp 20
8   10.004416 (0.0035)  svr4 > bsdi: icmp: svr4 udp port 8888 unreachable

9   15.001014 (4.9966)  bsdi.2924 > svr4.8888: udp 20
10  15.004574 (0.0036)  svr4 > bsdi: icmp: svr4 udp port 8888 unreachable

11  20.001177 (4.9966)  bsdi.2924 > svr4.8888: udp 20
12  20.004759 (0.0036)  svr4 > bsdi: icmp: svr4 udp port 8888 unreachable
```

Figure 6.8 ICMP port unreachable generated by TFTP request.

An ICMP port unreachable is immediately returned (line 4). But the TFTP client appears to ignore the ICMP message, sending another UDP datagram about 5 seconds later (line 5). This continues three more times before the client gives up.

Notice that the ICMP messages are exchanged between hosts, without a port number designation, while each 20-byte UDP datagram is from a specific port (2924) and to a specific port (8888).

The number 20 at the end of each UDP line is the length of the data in the UDP datagram. In this example 20 is the sum of the TFTP's 2-byte opcode, the 9-byte null terminated name `temp.foo`, and the 9-byte null terminated string `netascii`. (See Figure 15.1 for the details of the TFTP packet layout.)

If we run this same example using the `-e` option of `tcpdump` we see the exact length of each ICMP port unreachable message that's returned to the sender. This length is 70 bytes, and is allocated as shown in Figure 6.9.

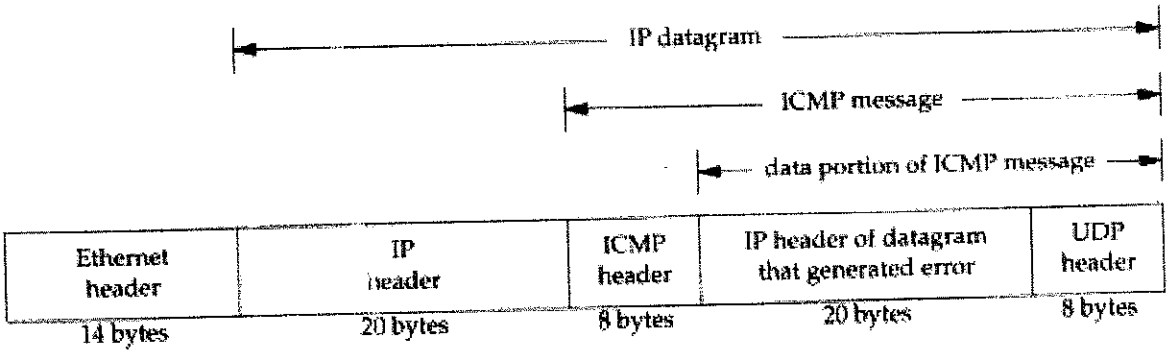

Figure 6.9 ICMP message returned for our "UDP port unreachable" example.

One rule of ICMP is that the ICMP error messages (see the final column of Figure 6.3, p. 71) must include the IP header (including any options) of the datagram that generated the error along with at least the first 8 bytes that followed this IP header. In our example, the first 8 bytes following the IP header contain the UDP header (Figure 11.2).

The important fact is that contained in the UDP header are the source and destination port numbers. It is this destination port number (8888) that caused the ICMP port unreachable to be generated. The source port number (2924) can be used by the system receiving the ICMP error to associate the error with a particular user process (the TFTP client in this example).

One reason the IP header of the datagram that caused the error is sent back is because in this IP header is the protocol field that lets ICMP know how to interpret the 8 bytes that follow (the UDP header in this example). When we look at the TCP header (Figure 17.2) we'll see that the source and destination port numbers are contained in the first 8 bytes of the TCP header.

The general format of the ICMP unreachable messages is shown in Figure 6.10.

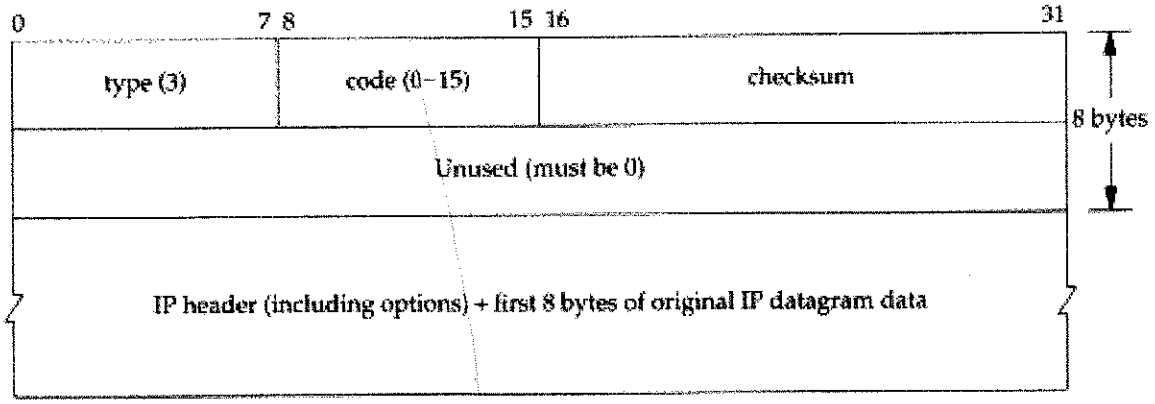

Figure 6.10 ICMP unreachable message.

In Figure 6.3 we noted that there are 16 different ICMP unreachable messages, *codes* 0 through 15. The ICMP port unreachable is *code* 3. Also, although Figure 6.10 indicates that the second 32-bit word in the ICMP message must be 0, the Path MTU Discovery mechanism (Section 2.9) allows a router to place the MTU of the outgoing interface in the low-order 16 bits of this 32-bit value, when *code* equals 4 ("fragmentation needed but the don't fragment bit is set"). We show an example of this error in Section 11.6.

> Although the rules of ICMP allow a system to return more than the first 8 bytes of the data portion of the IP datagram that caused the ICMP error, most Berkeley-derived implementations return exactly 8 bytes. The Solaris 2.2 `ip_icmp_return_data_bytes` option returns the first 64 bytes of data by default (Section E.4).

tcpdump Time Line

Throughout the text we'll also display the tcpdump output in a time line diagram as shown in Figure 6.11.

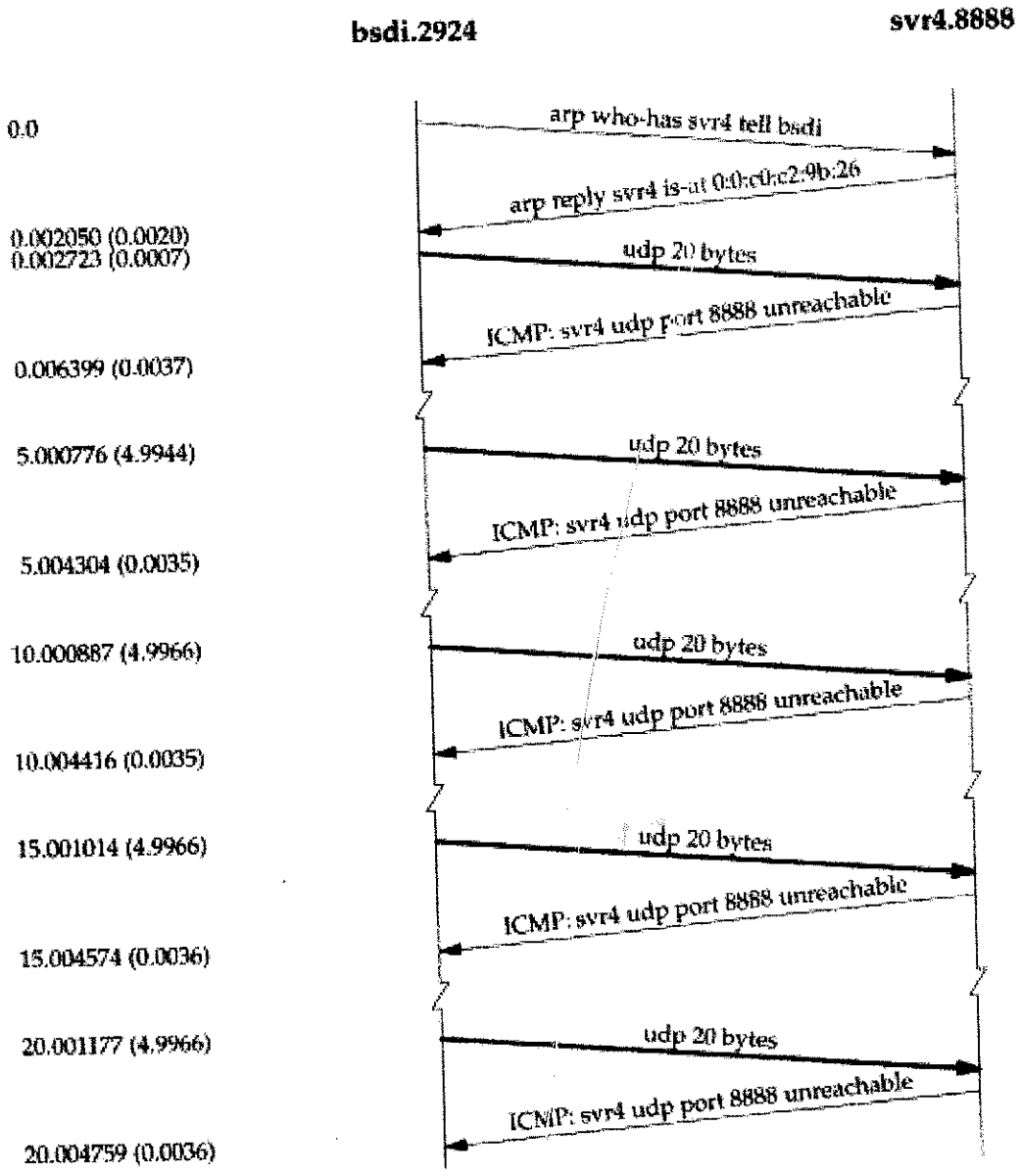

Figure 6.11 Time line of TFTP request to an invalid port.

Time increases down the page and the labels on the far left of the figure are the same time values as in our `tcpdump` output (Figure 6.8). The labels at the top are the hostnames and port numbers for each side of the time line. Be aware that the y-axis down the page is *not* exactly proportional to the time value. When there is a significant time lag, as between each 5-second retransmission in this example, we'll designate that with a squiggle on both sides of the time line. When UDP or TCP data is being transmitted, we show that packet with a thicker line.

Why does the TFTP client keep retransmitting its request when the ICMP messages are being returned? An element of network programming is occurring in which BSD systems don't notify user processes using UDP of ICMP messages that are received for that socket unless the process has issued a `connect` on that socket. The standard BSD TFTP client does not issue the `connect`, so it never receives the ICMP error notification.

Another point to notice here is the poor retransmission timeout algorithm used by this TFTP client. It just assumes that 5 seconds is adequate and retransmits every 5 seconds, for a total of 25 seconds. We'll see later that TCP has a much better algorithm.

> This old-fashioned timeout and retransmission algorithm used by the TFTP client is forbidden by the Host Requirements RFC. Nevertheless, all three systems on the author's subnet, and Solaris 2.2 still use it. AIX 3.2.2 applies an exponential backoff to its timeout, sending packets at 0, 5, 15, and 35 seconds, which is the recommended way. We talk much more about timeouts in Chapter 21.

Finally note that the ICMP messages are returned about 3.5 ms after the UDP datagram is sent, which we'll see in Chapter 7 is similar to the round-trip times for Ping replies.

6.6 4.4BSD Processing of ICMP Messages

Since ICMP covers such a wide range of conditions, from fatal errors to informational messages, each ICMP message is handled differently, even within a given implementation. Figure 6.12 is a redo of Figure 6.3, showing the handling performed by 4.4BSD for each of the possible ICMP messages.

If the final column specifies the kernel, that ICMP message is handled by the kernel. If the final column specifies "user process", then that message is passed to all user processes that have registered with the kernel to read received ICMP messages. If there are none of these user processes, the message is silently discarded. (These user processes also receive a copy of all the other ICMP messages, even those handled by the kernel, but only after the kernel has processed the message.) Some messages are completely ignored. Finally, if the final column is a string in quotes, that is the Unix error message corresponding to that condition. Some of these errors, such as TCP's handling of a source quench, we'll cover in later chapters.

type	*code*	Description	Handled by
0	0	echo reply	user process
3		destination unreachable:	
	0	network unreachable	"No route to host"
	1	host unreachable	"No route to host"
	2	protocol unreachable	"Connection refused"
	3	port unreachable	"Connection refused"
	4	fragmentation needed but DF bit set	"Message too long"
	5	source route failed	"No route to host"
	6	destination network unknown	"No route to host"
	7	destination host unknown	"No route to host"
	8	source host isolated (obsolete)	"No route to host"
	9	dest. network administratively prohibited	"No route to host"
	10	dest. host administratively prohibited	"No route to host"
	11	network unreachable for TOS	"No route to host"
	12	host unreachable for TOS	"No route to host"
	13	communication administratively prohibited	(ignored)
	14	host precedence violation	(ignored)
	15	precedence cutoff in effect	(ignored)
4	0	source quench	kernel for TCP, ignored by UDP
5		redirect:	
	0	redirect for network	kernel updates routing table
	1	redirect for host	kernel updates routing table
	2	redirect for type-of-service and network	kernel updates routing table
	3	redirect for type-of-service and host	kernel updates routing table
8	0	echo request	kernel generates reply
9	0	router advertisement	user process
10	0	router solicitation	user process
11		time exceeded:	
	0	TTL equals 0 during transit	user process
	1	TTL equals 0 during reassembly	user process
12		parameter problem:	
	0	IP header bad (catchall error)	"Protocol not available"
	1	required option missing	"Protocol not available"
13	0	timestamp request	kernel generates reply
14	0	timestamp reply	user process
15	0	information request (obsolete)	(ignored)
16	0	information reply (obsolete)	user process
17	0	address mask request	kernel generates reply
18	0	address mask reply	user process

Figure 6.12 Handling of the ICMP message types by 4.4BSD.

6.7 Summary

This chapter has been a look at the Internet Control Message Protocol, a required part of every implementation. Figure 6.3 lists all the ICMP message types, most of which we'll discuss later in the text.

We looked at the ICMP address mask request and reply and the timestamp request and reply in detail. These are typical of the request–reply messages. Both have an identifier and sequence number in the ICMP message. The sending application stores a unique value in the identifier field, to distinguish between replies for itself and replies for other processes. The sequence number field lets the client match replies with requests.

We also saw the ICMP port unreachable error, a common ICMP error. This let us examine the information returned in an ICMP error: the IP header and the next 8 bytes of the IP datagram that caused the error. This information is required by the receiver of the ICMP error, to know more about the cause of the error. Both TCP and UDP store the source and destination port numbers in the first 8 bytes of their headers for this reason.

Finally, we presented our first time line of `tcpdump` output, a presentation format we'll use in later chapters.

Exercises

6.1 At the end of Section 6.2 we listed five special conditions under which an ICMP error message is not sent. What would happen if these five conditions weren't followed and we sent a broadcast UDP datagram to an unlikely port on the local cable?

6.2 Read the Host Requirements RFC [Braden 1989a] to see if the generation of an ICMP port unreachable is a "must," "should," or "may." What section and page is this found on?

6.3 Read RFC 1349 [Almquist 1992] to see how the IP type-of-service field (Figure 3.2) should be set by ICMP.

6.4 If your system provides the `netstat` command, use it to see what types of ICMP messages are received and sent.