7

Ping Program

7.1 Introduction

The name "ping" is taken from the sonar operation to locate objects. The Ping program was written by Mike Muuss and it tests whether another host is reachable. The program sends an ICMP echo request message to a host, expecting an ICMP echo reply to be returned. (Figure 6.3 lists all the ICMP message types.)

Normally if you can't Ping a host, you won't be able to Telnet or FTP to that host. Conversely, if you can't Telnet to a host, Ping is often the starting point to determine what the problem is. Ping also measures the round-trip time to the host, giving us some indication of how "far away" that host is.

In this chapter we'll use Ping as a diagnostic tool and to further explore ICMP. Ping also gives us an opportunity to examine the IP record route and timestamp options. Chapter 11 of [Stevens 1990] provides the source code for the Ping program.

> Years ago we could make the unqualified statement that if we can't Ping a host, we can't Telnet or FTP to that host. With the increased awareness of security on the Internet, routers that provide access control lists, and firewall gateways, unqualified statements like this are no longer true. Reachability of a given host may depend not only on reachability at the IP layer, but also on what protocol is being used, and the port numbers involved. Ping may show a host as being unreachable, yet we might be able to Telnet to port 25 (the mail server).

7.2 Ping Program

We call the `ping` program that sends the echo requests the *client*, and the host being pinged the *server*. Most TCP/IP implementations support the Ping server directly in the kernel—the server is not a user process. (The two ICMP query services that we described in Chapter 6, the address mask and timestamp requests, are also handled directly by the kernel.)

Figure 7.1 shows the ICMP echo request and echo reply messages.

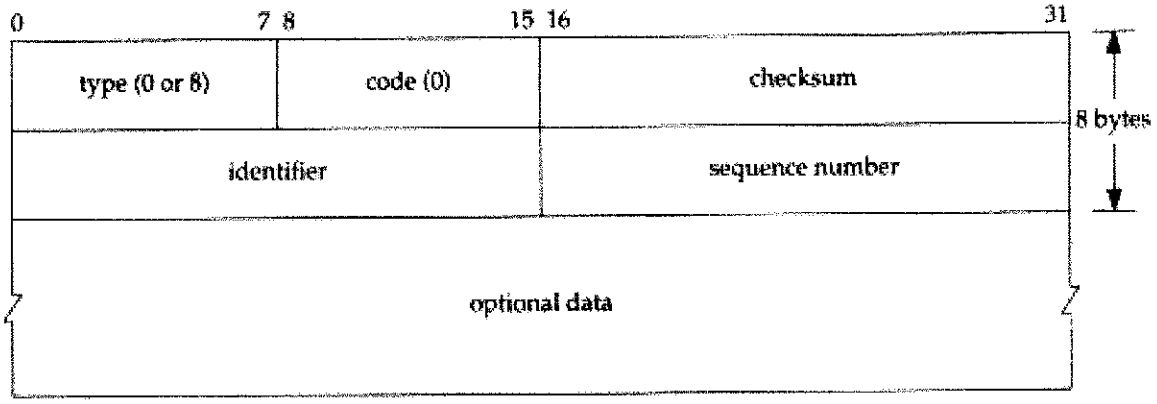

Figure 7.1 Format of ICMP message for echo request and echo reply.

As with other ICMP query messages, the server must echo the *identifier* and *sequence number* fields. Also, any optional data sent by the client must be echoed. These are presumably of interest to the client.

Unix implementations of `ping` set the *identifier* field in the ICMP message to the process ID of the sending process. This allows `ping` to identify the returned responses if there are multiple instances of `ping` running at the same time on the same host.

The *sequence number* starts at 0 and is incremented every time a new echo request is sent. `ping` prints the sequence number of each returned packet, allowing us to see if packets are missing, reordered, or duplicated. IP is a best effort datagram delivery service, so any of these three conditions can occur.

Historically the `ping` program has operated in a mode where it sends an echo request once a second, printing each echo reply that is returned. Newer implementations, however, require the `-s` option to operate this way. By default, these newer implementations send only a single echo request and output "*host* is alive" if an echo reply is received, or "no answer" if no reply is received within 20 seconds.

LAN Output

`ping` output on a LAN normally looks like the following:

```
bsdi % ping svr4
PING svr4 (140.252.13.34): 56 data bytes
64 bytes from 140.252.13.34: icmp_seq=0 ttl=255 time=0 ms
64 bytes from 140.252.13.34: icmp_seq=1 ttl=255 time=0 ms
64 bytes from 140.252.13.34: icmp_seq=2 ttl=255 time=0 ms
64 bytes from 140.252.13.34: icmp_seq=3 ttl=255 time=0 ms
64 bytes from 140.252.13.34: icmp_seq=4 ttl=255 time=0 ms
64 bytes from 140.252.13.34: icmp_seq=5 ttl=255 time=0 ms
64 bytes from 140.252.13.34: icmp_seq=6 ttl=255 time=0 ms
64 bytes from 140.252.13.34: icmp_seq=7 ttl=255 time=0 ms
^?                                        type interrupt key to stop
--- svr4 ping statistics ---
8 packets transmitted, 8 packets received, 0% packet loss
round-trip min/avg/max = 0/0/0 ms
```

When the ICMP echo reply is returned, the sequence number is printed, followed by the TTL, and the round-trip time is calculated. (TTL is the time-to-live field in the IP header. The current BSD `ping` program prints the received TTL each time an echo reply is received—some implementations don't do this. We examine the usage of the TTL in Chapter 8 with the `traceroute` program.)

As we can see from the output above, the echo replies were returned in the order sent (0, 1, 2, and so on).

`ping` is able to calculate the round-trip time by storing the time at which it sends the echo request in the data portion of the ICMP message. When the reply is returned it subtracts this value from the current time. Notice that on the sending system, `bsdi`, the round-trip times are all calculated as 0 ms. This is because of the low-resolution timer available to the program. The BSD/386 Version 0.9.4 system only provides a 10-ms timer. (We talk more about this in Appendix B.) We'll see later that when looking at the `tcpdump` output from this `ping` example on a system with a finer resolution clock (the Sun) the time difference between the ICMP echo request and its echo reply is just under 4 ms.

The first line of output contains the IP address of the destination host, even though we specified its name (`svr4`). This implies that the name has been converted to the IP address by a resolver. We examine resolvers and the DNS in Chapter 14. For now realize that if we type a `ping` command, and a few seconds pass before the first line of output with the IP address is printed, this is the time required for the DNS to determine the IP address corresponding to the hostname.

Figure 7.2 shows the `tcpdump` output for this example.

```
1  0.0                    bsdi > svr4: icmp: echo request
2  0.003733 (0.0037)      svr4 > bsdi: icmp: echo reply

3  0.998045 (0.9943)      bsdi > svr4: icmp: echo request
4  1.001747 (0.0037)      svr4 > bsdi: icmp: echo reply

5  1.997818 (0.9961)      bsdi > svr4: icmp: echo request
6  2.001542 (0.0037)      svr4 > bsdi: icmp: echo reply

7  2.997610 (0.9961)      bsdi > svr4: icmp: echo request
8  3.001311 (0.0037)      svr4 > bsdi: icmp: echo reply

9  3.997390 (0.9961)      bsdi > svr4: icmp: echo request
10 4.001115 (0.0037)      svr4 > bsdi: icmp: echo reply

11 4.997201 (0.9961)      bsdi > svr4: icmp: echo request
12 5.000904 (0.0037)      svr4 > bsdi: icmp: echo reply

13 5.996977 (0.9961)      bsdi > svr4: icmp: echo request
14 6.000708 (0.0037)      svr4 > bsdi: icmp: echo reply

15 6.996764 (0.9961)      bsdi > svr4: icmp: echo request
16 7.000479 (0.0037)      svr4 > bsdi: icmp: echo reply
```

Figure 7.2 `ping` output across a LAN.

The time between sending the echo request and receiving the echo reply is always 3.7 ms. We can also see that echo requests are sent approximately 1 second apart.

Often the first round-trip time is larger than the rest. This occurs if the destination's hardware address isn't in the ARP cache of the sender. As we saw in Chapter 4, sending an ARP request and getting the ARP reply can take a few milliseconds before the first echo request can be sent. The following example shows this:

```
sun % arp -a                                    make sure ARP cache is empty

sun % ping svr4
PING svr4: 56 data bytes
64 bytes from svr4 (140.252.13.34): icmp_seq=0. time=7. ms
64 bytes from svr4 (140.252.13.34): icmp_seq=1. time=4. ms
64 bytes from svr4 (140.252.13.34): icmp_seq=2. time=4. ms
64 bytes from svr4 (140.252.13.34): icmp_seq=3. time=4. ms
^?                                              type interrupt key to stop
----svr4 PING Statistics----
4 packets transmitted, 4 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 4/4/7
```

The additional 3 ms in the first RTT is probably for the ARP request and reply.

This example was run on the host sun, which provides a timer with microsecond resolution, but the ping program prints the round-trip times with only millisecond resolution. The earlier example, run under BSD/386 Version 0.9.4, printed the round-trip times as 0 ms, since the available timer provided only 10-ms accuracy. The following output is from BSD/386 Version 1.0, which provides a timer with microsecond resolution and a version of ping that prints the higher resolution.

```
bsdi % ping svr4
PING svr4 (140.252.13.34): 56 data bytes
64 bytes from 140.252.13.34: icmp_seq=0 ttl=255 time=9.304 ms
64 bytes from 140.252.13.34: icmp_seq=1 ttl=255 time=6.089 ms
64 bytes from 140.252.13.34: icmp_seq=2 ttl=255 time=6.079 ms
64 bytes from 140.252.13.34: icmp_seq=3 ttl=255 time=6.096 ms
^?                                              type interrupt key to stop
--- svr4 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 6.079/6.880/9.304 ms
```

WAN Output

On a wide area network the results can be quite different. The following example was captured on a weekday afternoon, a time when the Internet is normally busy:

```
gemini % ping vangogh.cs.berkeley.edu
PING vangogh.cs.berkeley.edu: 56 data bytes
64 bytes from (128.32.130.2): icmp_seq=0. time=660. ms
64 bytes from (128.32.130.2): icmp_seq=5. time=1780. ms
64 bytes from (128.32.130.2): icmp_seq=7. time=380. ms
64 bytes from (128.32.130.2): icmp_seq=8. time=420. ms
64 bytes from (128.32.130.2): icmp_seq=9. time=390. ms
64 bytes from (128.32.130.2): icmp_seq=14. time=110. ms
64 bytes from (128.32.130.2): icmp_seq=15. time=170. ms
64 bytes from (128.32.130.2): icmp_seq=16. time=100. ms
^?                                              type interrupt key to stop
```

```
----vangogh.CS.Berkeley.EDU PING Statistics----
17 packets transmitted, 8 packets received, 52% packet loss
round-trip (ms)  min/avg/max = 100/501/1780
```

Either the echo requests or the echo replies for sequence numbers 1, 2, 3, 4, 6, 10, 11, 12, and 13 were lost somewhere. Note also the large variance in the round-trip times. (This high packet loss rate of 52% is an anomaly. This is not normal for the Internet, even on a weekday afternoon.)

It is also possible across WANs to see packets duplicated (the same sequence number printed two or more times), and to see packets reordered (sequence number $N + 1$ printed before sequence number N).

Hardwired SLIP Links

Let's look at the round-trip times encountered over SLIP links, since they often run at slow asynchronous speeds, such as 9600 bits/sec or less. Recall our serial line throughput calculations in Section 2.10. For this example we'll set the speed of the hardwired SLIP link between hosts `bsdi` and `slip` to 1200 bits/sec.

We can estimate the round-trip time as follows. First, notice from the example Ping output shown earlier that by default it sends 56 bytes of data in the ICMP message. With a 20-byte IP header and an 8-byte ICMP header this gives a total IP datagram size of 84 bytes. (We can verify this by running `tcpdump -e` and seeing the Ethernet frame sizes.) Also, from Section 2.4 we know that at least two additional bytes are added: the END byte at the beginning and end of the datagram. It's also possible for additional bytes to be added by the SLIP framing, but that depends on the value of each byte in the datagram. At 1200 bits/sec with 8 bits per byte, 1 start bit, and 1 stop bit, the rate is 120 bytes per second, or 8.33 ms per byte. Our estimate is then $(86 \times 8.33 \times 2)$, or 1433 ms. (The multiplier of 2 is because we are calculating the round-trip time.)

The following output verifies our calculation:

```
svr4 % ping -s slip
PING slip: 56 data bytes
64 bytes from slip (140.252.13.65): icmp_seq=0. time=1480. ms
64 bytes from slip (140.252.13.65): icmp_seq=1. time=1480. ms
64 bytes from slip (140.252.13.65): icmp_seq=2. time=1480. ms
64 bytes from slip (140.252.13.65): icmp_seq=3. time=1480. ms
^?
----slip PING Statistics----
5 packets transmitted, 4 packets received, 20% packet loss
round-trip (ms)  min/avg/max = 1480/1480/1480
```

(The `-s` option is required for SVR4 to send one request every second.) The round-trip time is almost 1.5 seconds but the program is still sending out each ICMP echo request at 1-second intervals. This means there are two outstanding echo requests (sent at time 0 and time 1) before the first reply comes back (at time 1.480). That's also why the summary line says one packet has been lost. It really hasn't been lost, it's probably still on its way back.

We'll return to this slow SLIP link in Chapter 8 when we examine the `traceroute` program.

Dialup SLIP Links

The conditions change with a dialup SLIP link since we now have modems on each end of the link. The modems being used between the systems sun and netb provide what is called V.32 modulation (9600 bits/sec), V.42 error control (also called LAP-M), and V.42bis data compression. This means that our simple calculations, which were fairly accurate for a hardwired link where we knew all the parameters, become less accurate.

Numerous factors are at work. The modems introduce some latency. The size of the packets may decrease with the data compression, but the size may then increase to a multiple of the packet size used by the error control protocol. Also the receiving modem can't release received data bytes until the cyclic redundancy character (the checksum) has been verified. Finally, we're dealing with a computer's asynchronous serial interface on each end, and many operating systems read these interfaces only at fixed intervals, or when a certain number of characters have been received.

As an example, we ping the host gemini from the host sun:

```
sun % ping gemini
PING gemini: 56 data bytes
64 bytes from gemini (140.252.1.11): icmp_seq=0. time=373. ms
64 bytes from gemini (140.252.1.11): icmp_seq=1. time=360. ms
64 bytes from gemini (140.252.1.11): icmp_seq=2. time=340. ms
64 bytes from gemini (140.252.1.11): icmp_seq=3. time=320. ms
64 bytes from gemini (140.252.1.11): icmp_seq=4. time=330. ms
64 bytes from gemini (140.252.1.11): icmp_seq=5. time=310. ms
64 bytes from gemini (140.252.1.11): icmp_seq=6. time=290. ms
64 bytes from gemini (140.252.1.11): icmp_seq=7. time=300. ms
64 bytes from gemini (140.252.1.11): icmp_seq=8. time=280. ms
64 bytes from gemini (140.252.1.11): icmp_seq=9. time=290. ms
64 bytes from gemini (140.252.1.11): icmp_seq=10. time=300. ms
64 bytes from gemini (140.252.1.11): icmp_seq=11. time=280. ms
----gemini PING Statistics----
12 packets transmitted, 12 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 280/314/373
```

Note that the first RTT is not a multiple of 10 ms, but every other line is. If we run this numerous times, we see this property every time. (This is not caused by the resolution of the clock on the host sun, because we know that its clock provides millisecond resolution from the tests we run in Appendix B.)

Also note that the first RTT is larger than the next, and they keep decreasing, and then they range between 280 and 300 ms. If we let it run for a minute or two, the RTTs stay in this range, never going below 260 ms. If we calculate the expected RTT at 9600 bits/sec (Exercise 7.2) we get 180 ms, so our observed values are about 1.5 times the expected value.

If we run ping for 60 seconds and look at the average RTT it calculates, we find that with V.42 and V.42bis our average is 277 ms. (This is better than the average printed for our preceding example, because we ran it longer to amortize the longer RTTs at the beginning.) If we turn off just the V.42bis data compression our average is 330 ms. If we turn off the V.42 error control (which also turns off the V.42bis data compression) our average is 300 ms. These modem parameters do affect the RTTs, and using the error control and data compression appears to be the best.

7.3 IP Record Route Option

The `ping` program gives us an opportunity to look at the IP record route (RR) option. Most versions of `ping` provide the `-R` option that enables the record route feature. It causes `ping` to set the IP RR option in the outgoing IP datagram (which contains the ICMP echo request message). This causes every router that handles the datagram to add its IP address to a list in the options field. When the datagram reaches the final destination, the list of IP addresses should be copied into the outgoing ICMP echo reply, and all the routers on the return path also add their IP addresses to the list. When `ping` receives the echo reply it prints the list of IP addresses.

As simple as this sounds, there are pitfalls. Generation of the RR option by the source host, processing of the RR option by the intermediate routers, and reflection of the incoming RR list in an ICMP echo request into the outgoing ICMP echo reply are all optional features. Fortunately, most systems today do support these optional features, but some systems don't reflect the IP list.

The biggest problem, however, is the limited room in the IP header for the list of IP addresses. We saw in Figure 3.1 (p. 34) that the *header length* in the IP header is a 4-bit field, limiting the entire IP header to 15 32-bit words (60 bytes). Since the fixed size of the IP header is 20 bytes, and the RR option uses 3 bytes for overhead (which we describe below), this leaves 37 bytes (60 – 20 – 3) for the list, allowing up to nine IP addresses. In the early days of the ARPANET, nine IP addresses seemed like a lot, but since this is a round-trip list (in the case of the `-R` option for `ping`), it's of limited use today. (In Chapter 8 we'll look at the Traceroute tool for determining the route followed by a datagram.) Despite these shortcomings, the record route option works and provides an opportunity to look in detail at the handling of IP options.

Figure 7.3 shows the general format of the RR option in the IP datagram.

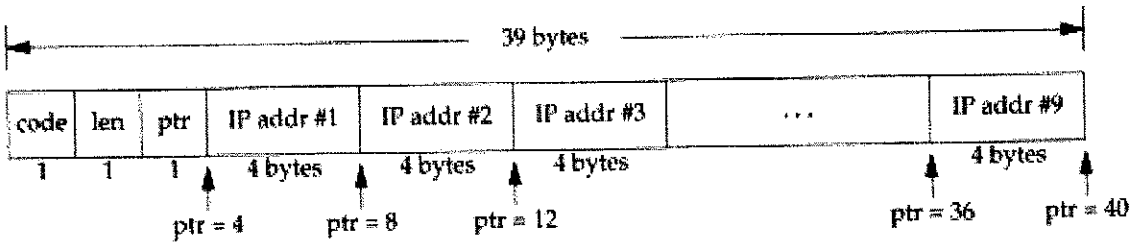

Figure 7.3 General format of record route option in IP header.

Code is a 1-byte field specifying the type of IP option. For the RR option its value is 7. *Len* is the total number of bytes of the RR option, which in this case is 39. (Although it's possible to specify an RR option with less than the maximum size, `ping` always provides a 39-byte option field, to record up to nine IP addresses. Given the limited room in the IP header for options, it doesn't make sense to specify a size less than the maximum.)

Ptr is called the pointer field. It is a 1-based index into the 39-byte option of where to store the next IP address. Its minimum value is 4, which is the pointer to the first IP address. As each IP address is recorded into the list, the value of *ptr* becomes 8, 12, 16, up to 36. After the ninth address is recorded *ptr* becomes 40, indicating the list is full.

When a router (which by definition is multihomed) records its IP address in the list, which IP address is recorded? It could be the address of the incoming interface or the outgoing interface. RFC 791 [Postel 1981a] specifies that the router records the outgoing IP address. We'll see that when the originating host (the host running `ping`) receives the ICMP echo reply with the RR option enabled, it also records its incoming IP address in the list.

Normal Example

Let's run an example of the RR option with the `ping` program. We'll run `ping` on the host `svr4` to the host `slip`. One intermediate router (`bsdi`) will handle the datagram. The following output is from `svr4`:

```
svr4 % ping -R slip
PING slip (140.252.13.65): 56 data bytes
64 bytes from 140.252.13.65: icmp_seq=0 ttl=254 time=280 ms
RR:     bsdi (140.252.13.66)
        slip (140.252.13.65)
        bsdi (140.252.13.35)
        svr4 (140.252.13.34)
64 bytes from 140.252.13.65: icmp_seq=1 ttl=254 time=280 ms (same route)
64 bytes from 140.252.13.65: icmp_seq=2 ttl=254 time=270 ms (same route)
^?
--- slip ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 270/276/280 ms
```

Figure 7.4 shows the four hops that the packets take (two in each direction), and which hop adds which IP address to the RR list.

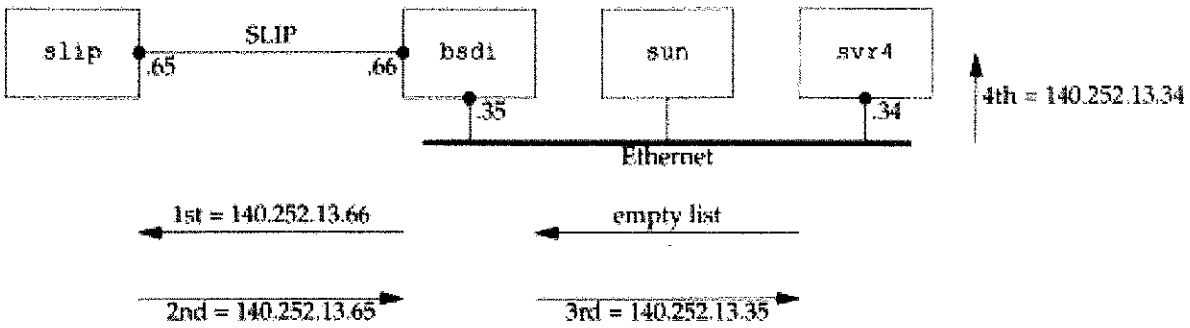

Figure 7.4 `ping` with record route option.

The router `bsdi` adds a different IP address to the list in each direction. It always adds the IP address of the outgoing interface. We can also see that when the ICMP echo reply reaches the originating system (`svr4`) it adds the IP address of the incoming interface to the list.

We can also watch this exchange of packets from the host sun, running tcpdump with its -v option (to see the IP options). Figure 7.5 shows the output.

```
1  0.0                    svr4 > slip: icmp: echo request (ttl 32, id 35835,
                          optlen=40 RR(39)= RR(#0.0.0.0/0.0.0.0/0.0.0.0/
                          0.0.0.0/ 0.0.0.0/0.0.0.0/0.0.0.0/0.0.0.0/0.0.0.0) EOL)

2  0.267746 (0.2677)      slip > svr4: icmp: echo reply (ttl 254, id 1976,
                          optlen=40 RR(39)= RR(140.252.13.66/140.252.13.65/
                          140.252.13.35/#0.0.0.0/0.0.0.0/0.0.0.0/0.0.0.0/
                          0.0.0.0/0.0.0.0) EOL)
```

Figure 7.5 tcpdump output of record route option.

The output optlen=40 indicates there are 40 bytes of option space in the IP header. (Recall that the length of the IP header must be a multiple of 4 bytes.) RR(39) means the record route option is present, and its length field is 39. The list of nine IP addresses is then shown, with a pound sign (#) indicating which IP address is pointed to by the *ptr* field in the RR option header. Since we are watching these packets on the host sun (see Figure 7.4) we only see the ICMP echo request with the empty list, and the ICMP echo reply with three addresses in the list. We have deleted the remaining lines in the tcpdump output, since they are nearly identical to what we show in Figure 7.5.

The notation EOL at the end of the record route information indicates the IP option "end of list" value appeared. The EOL option has a value of 0. What's happening is that 39 bytes of RR data are in the 40 bytes of option space in the IP header. Since the option space is set to 0 before the datagram is sent, this final byte of 0 that follows the 39 bytes of RR data is interpreted as an EOL. That is what we want to have happen. If there are multiple options in the option field of the IP header, and pad bytes are needed before the next option starts, the other special character NOP ("no operation"), with a value of 1, can be used.

In Figure 7.5, SVR4 sets the TTL field of the echo request to 32, and BSD/386 sets it to 255. (It prints as 254 since the router bsdi has already decremented it by one.) Newer systems are setting the TTL of ICMP messages to the maximum (255).

It turns out that of the three TCP/IP implementations used by the author, both BSD/386 and SVR4 support the record route option. That is, they correctly update the RR list when forwarding a datagram, and they correctly reflect the RR list from an incoming ICMP echo request to the outgoing ICMP echo reply. SunOS 4.1.3, however, updates the RR list when forwarding a datagram, but does not reflect the RR list. Solaris 2.x corrects this problem.

Abnormal Output

The following example was seen by the author and provides a starting point for our description of the ICMP redirect message in Chapter 9. We ping the host aix on the 140.252.1 subnet (accessible through the dialup SLIP connection on the host sun) with the record route option. We get the following output, when run on the host slip:

```
slip % ping -R aix
PING aix (140.252.1.92): 56 data bytes
64 bytes from 140.252.1.92: icmp_seq=0 ttl=251 time=650 ms
RR:     bsdi (140.252.13.35)
        sun (140.252.1.29)
        netb (140.252.1.183)
        aix (140.252.1.92)
        gateway (140.252.1.4)          why is this router used?
        netb (140.252.1.183)
        sun (140.252.13.33)
        bsdi (140.252.13.66)
        slip (140.252.13.65)
64 bytes from aix: icmp_seq=1 ttl=251 time=610 ms (same route)
64 bytes from aix: icmp_seq=2 ttl=251 time=600 ms (same route)
^?
--- aix ping statistics ---
4 packets transmitted, 3 packets received, 25% packet loss
round-trip min/avg/max = 600/620/650 ms
```

We could have run this example from the host `bsdi`. We chose to run it from `slip` to see all nine IP addresses in the RR list used.

The puzzle in this output is why the outgoing datagram (the ICMP echo request) went directly from `netb` to `aix`, but the return (the ICMP echo reply) went from `aix`, through the router `gateway`, before going to `netb`. What we're seeing here is a feature of IP routing that we describe below. Figure 7.6 shows the path of the datagrams.

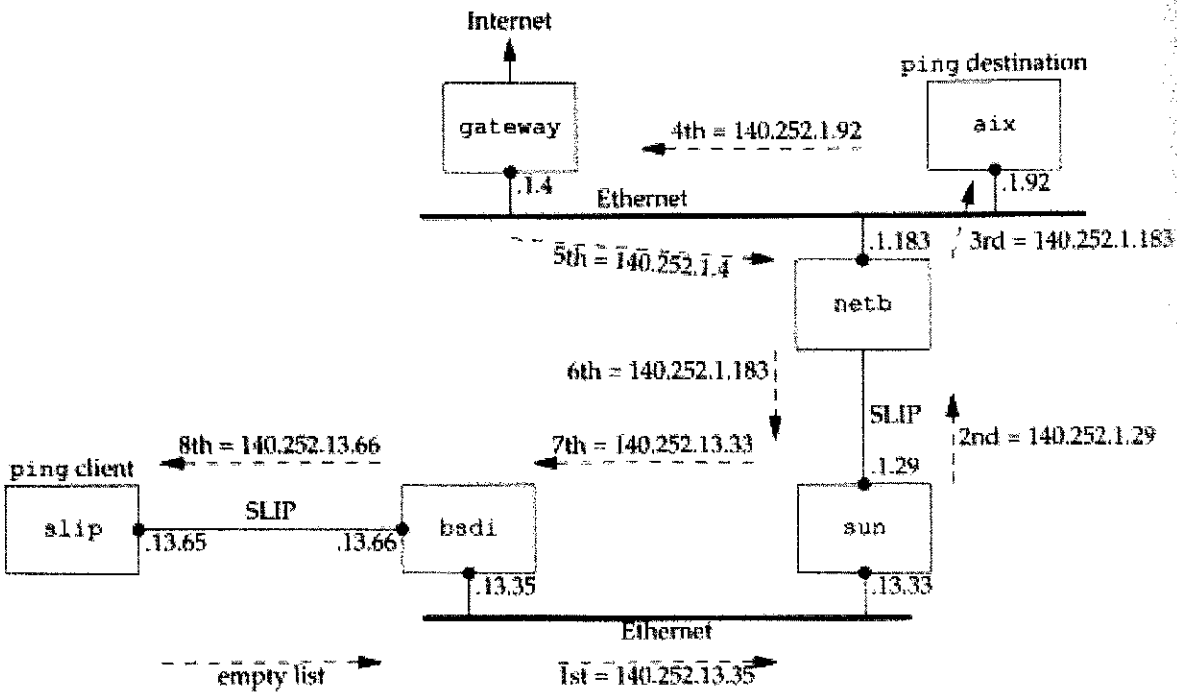

Figure 7.6 `ping` with record route, showing IP routing feature.

The problem is that `aix` does not know to send IP datagrams destined for the subnet 140.252.13 to `netb`. Instead, `aix` has a default entry in its routing table that tells it to

send all datagrams to the router `gateway` if it doesn't have a particular route for the destination. The router `gateway` has more routing knowledge than any of the hosts on the 140.252.1 subnet. (There are more than 150 hosts on this Ethernet and instead of running a routing daemon on every one, each has a "default" entry that points to the router `gateway`.)

An unanswered question here is why doesn't `gateway` send an ICMP redirect (Section 9.5) to `aix` to update its routing table? For some reason (perhaps that the datagram generating the redirect is an ICMP echo request message) the redirect is not generated. But if we use Telnet and connect to the daytime server on `aix`, the ICMP redirect is generated, and the routing table on `aix` is updated. If we then execute `ping` with the record route option enabled, the route shows that the datagrams go from `netb` to `aix` and back to `netb`, without the extra hop to the router `gateway`. We'll look at these ICMP redirects in more detail in Section 9.5.

7.4 IP Timestamp Option

The IP timestamp option is similar to the record route option. Figure 7.7 shows the format of the IP timestamp option (compare with Figure 7.3).

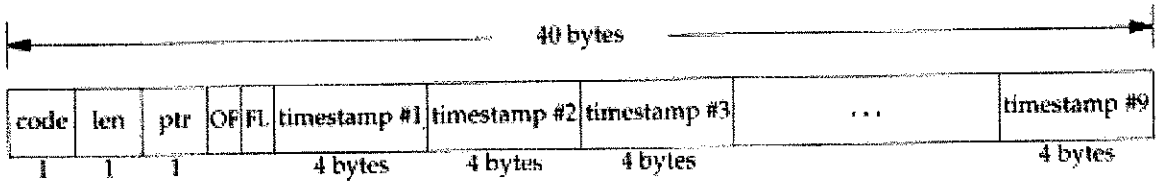

Figure 7.7 General format of timestamp option in IP header.

The *code* field is `0x44` for the timestamp option. The two fields *len* and *ptr* are the same as for the record route option: the total length of the option (normally 36 or 40) and a pointer to the next available entry (5, 9, 13, etc.).

The next two fields are 4-bit values: *OF* is the *overflow* field and *FL* is a *flags* field. The operation of the timestamp option is driven by the *flags* field, as shown in Figure 7.8.

flags	Description
0	Record only timestamps. This is what we show in Figure 7.7.
1	Each router records its IP address and its timestamp. There is room for only four of these pairs in the options list.
3	The sender initializes the options list with up to four pairs of IP addresses and a 0 timestamp. A router records its timestamp only if the next IP address in the list matches the router's.

Figure 7.8 Meaning of the *flags* value for timestamp option.

If a router can't add a timestamp because there's no room left, it just increments the *overflow* field.

The preferred value for the timestamps is the number of milliseconds past midnight, UTC, similar to the ICMP timestamp request and reply (Section 6.4). If this format is not available to a router, it can insert whatever time representation that it uses, but must then turn on the high-order bit of the timestamp to indicate the nonstandard value.

Given the limitations that we encountered with the record route option, things get worse with the timestamp option. If we record both IP addresses and timestamps (a *flags* of 1), we can store only four of these pairs. Recording only timestamps is next to useless because we have no indication regarding which timestamp corresponds to which router (unless we have a fixed topology that never changes). A *flags* of 3 is better, as we can then select which routers insert their timestamp. A more fundamental problem is that you probably have no control over how accurate the timestamp is at any given router. This makes it fruitless to try to measure hop times between routers using this IP option. We'll see that the traceroute program (Chapter 8) provides a better way of measuring hop times between routers.

7.5 Summary

The `ping` program is the basic connectivity test between two systems running TCP/IP. It uses the ICMP echo request and echo reply messages and does not use a transport layer (TCP or UDP). The Ping server is normally part of the kernel's ICMP implementation.

We looked at the normal `ping` output for a LAN, WAN, and SLIP links (dialup and hardwired), and performed some serial line throughput calculations for a dedicated SLIP link. `ping` also let us examine and use the IP record route option. We used this IP option to see how default routes are often used, and will return to this topic in Chapter 9. We also looked at the IP timestamp option, but it is of limited practical use.

Exercises

7.1 Draw a time line for the `ping` output for the SLIP link in Section 7.2.

7.2 Calculate the RTT if the SLIP link between `bsdi` and `slip` is set to 9600 bits/sec. Assume the default of 56 bytes of data.

7.3 The current BSD `ping` program allows us to specify a pattern for the data portion of the ICMP message. (The first 8 bytes of the data portion are not filled with the pattern, since the time at which the packet is sent is stored here.) If we specify a pattern of `0xc0`, recalculate the answer to the previous exercise. (*Hint*: Reread Section 2.4.)

7.4 Does the use of compressed SLIP (CSLIP, Section 2.5) affect the `ping` times that we observed in Section 7.2?

7.5 Examine Figure 2.4 (p. 28). Do you expect any difference between a ping of the loopback address, versus a ping of the host's Ethernet address?