4

ARP: Address Resolution Protocol

4.1 Introduction

The problem that we deal with in this chapter is that IP addresses only make sense to the TCP/IP protocol suite. A data link such as an Ethernet or a token ring has its own addressing scheme (often 48-bit addresses) to which any network layer using the data link must conform. A network such as an Ethernet can be used by different network layers at the same time. For example, a collection of hosts using TCP/IP and another collection of hosts using some PC network software can share the same physical cable.

When an Ethernet frame is sent from one host on a LAN to another, it is the 48-bit Ethernet address that determines for which interface the frame is destined. The device driver software never looks at the destination IP address in the IP datagram.

Address resolution provides a mapping between the two different forms of addresses: 32-bit IP addresses and whatever type of address the data link uses. RFC 826 [Plummer 1982] is the specification of ARP.

Figure 4.1 shows the two protocols we talk about in this chapter and the next: ARP (address resolution protocol) and RARP (reverse address resolution protocol).

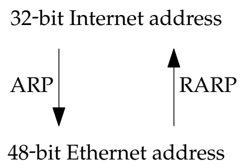

Figure 4.1 Address resolution protocols: ARP and RARP.

ARP provides a dynamic mapping from an IP address to the corresponding hardware address. We use the term *dynamic* since it happens automatically and is normally not a concern of either the application user or the system administrator.

RARP is used by systems without a disk drive (normally diskless workstations or X terminals) but requires manual configuration by the system administrator. We describe it in Chapter 5.

4.2 An Example

Whenever we type a command of the form

```
% ftp bsdi
```

the following steps take place. These numbered steps are shown in Figure 4.2.

1. The application, the FTP client, calls the function `gethostbyname`(3) to convert the hostname (`bsdi`) into its 32-bit IP address. This function is called a *resolver* in the DNS (Domain Name System), which we describe in Chapter 14. This conversion is done using the DNS, or on smaller networks, a static hosts file (`/etc/hosts`).

2. The FTP client asks its TCP to establish a connection with that IP address.

3. TCP sends a connection request segment to the remote host by sending an IP datagram to its IP address. (We'll see the details of how this is done in Chapter 18.)

4. If the destination host is on a locally attached network (e.g., Ethernet, token ring, or the other end of a point-to-point link), the IP datagram can be sent directly to that host. If the destination host is on a remote network, the IP routing function determines the Internet address of a locally attached next-hop router to send the IP datagram to. In either case the IP datagram is sent to a host or router on a locally attached network.

5. Assuming an Ethernet, the sending host must convert the 32-bit IP address into a 48-bit Ethernet address. A translation is required from the *logical* Internet address to its corresponding *physical* hardware address. This is the function of ARP.

   ARP is intended for broadcast networks where many hosts or routers are connected to a single network.

6. ARP sends an Ethernet frame called an *ARP request* to every host on the network. This is called a *broadcast*. We show the broadcast in Figure 4.2 with dashed lines. The ARP request contains the IP address of the destination host (whose name is `bsdi`) and is the request "if you are the owner of this IP address, please respond to me with your hardware address."

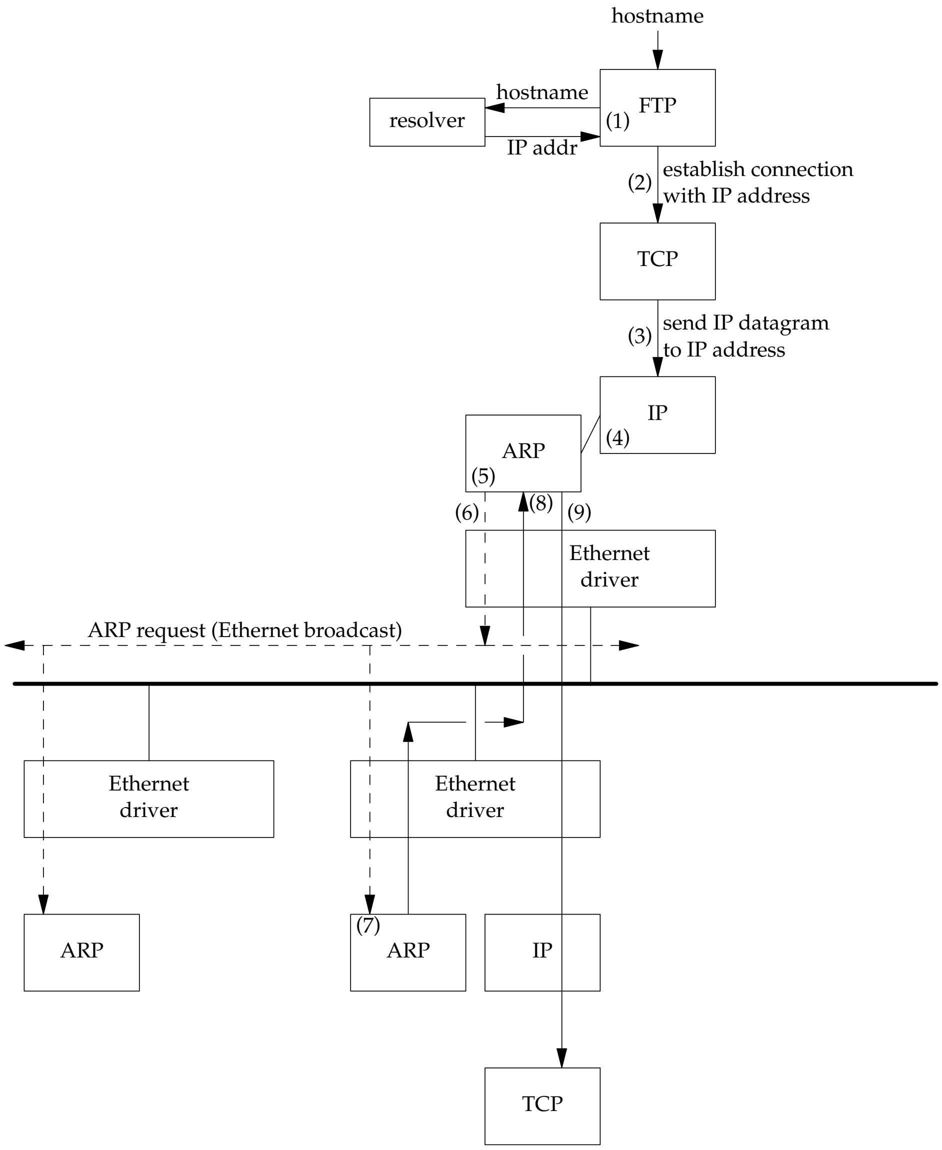

Figure 4.2 Operation of ARP when user types "`ftp` hostname".

7. The destination host's ARP layer receives this broadcast, recognizes that the sender is asking for its IP address, and replies with an *ARP reply*. This reply contains the IP address and the corresponding hardware address.
8. The ARP reply is received and the IP datagram that forced the ARP request–reply to be exchanged can now be sent.
9. The IP datagram is sent to the destination host.

The fundamental concept behind ARP is that the network interface has a hardware address (a 48-bit value for an Ethernet or token ring interface). Frames exchanged at the hardware level must be addressed to the correct interface. But TCP/IP works with its own addresses: 32-bit IP addresses. Knowing a host's IP address doesn't let the kernel send a frame to that host. The kernel (i.e., the Ethernet driver) must know the destination's hardware address to send it data. The function of ARP is to provide a dynamic mapping between 32-bit IP addresses and the hardware addresses used by various network technologies.

Point-to-point links don't use ARP. When these links are configured (normally at bootstrap time) the kernel must be told of the IP address at each end of the link. Hardware addresses such as Ethernet addresses are not involved.

4.3 ARP Cache

Essential to the efficient operation of ARP is the maintenance of an *ARP cache* on each host. This cache maintains the recent mappings from Internet addresses to hardware addresses. The normal expiration time of an entry in the cache is 20 minutes from the time the entry was created.

We can examine the ARP cache with the `arp`(8) command. The `-a` option displays all entries in the cache:

```
bsdi % arp -a
sun (140.252.13.33) at 8:0:20:3:f6:42
svr4 (140.252.13.34) at 0:0:c0:c2:9b:26
```

The 48-bit Ethernet addresses are displayed as six hexadecimal numbers separated by colons. We discuss additional features of the `arp` command in Section 4.8.

4.4 ARP Packet Format

Figure 4.3 shows the format of an ARP request and an ARP reply packet, when used on an Ethernet to resolve an IP address. (ARP is general enough to be used on other networks and can resolve addresses other than IP addresses. The first four fields following the frame type field specify the types and sizes of the final four fields.)

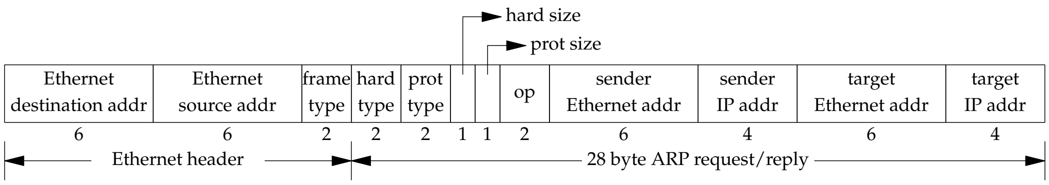

Figure 4.3 Format of ARP request or reply packet when used on an Ethernet.

The first two fields in the Ethernet header are the source and destination Ethernet addresses. The special Ethernet destination address of all one bits means the broadcast address. All Ethernet interfaces on the cable receive these frames.

The 2-byte Ethernet *frame type* specifies the type of data that follows. For an ARP request or an ARP reply, this field is `0x0806`.

The adjectives *hardware* and *protocol* are used to describe the fields in the ARP packets. For example, an ARP request asks for the protocol address (an IP address in this case) corresponding to a hardware address (an Ethernet address in this case).

The *hard type* field specifies the type of hardware address. Its value is 1 for an Ethernet. *Prot type* specifies the type of protocol address being mapped. Its value is `0x0800` for IP addresses. This is purposely the same value as the type field of an Ethernet frame containing an IP datagram. (See Figure 2.1, p. 23.)

The next two 1-byte fields, *hard size* and *prot size*, specify the sizes in bytes of the hardware addresses and the protocol addresses. For an ARP request or reply for an IP address on an Ethernet they are 6 and 4, respectively.

The *op* field specifies whether the operation is an ARP request (a value of 1), ARP reply (2), RARP request (3), or RARP reply (4). (We talk about RARP in Chapter 5.) This field is required since the *frame type* field is the same for an ARP request and an ARP reply.

The next four fields that follow are the sender's hardware address (an Ethernet address in this example), the sender's protocol address (an IP address), the target hardware address, and the target protocol address. Notice there is some duplication of information: the sender's hardware address is available both in the Ethernet header and in the ARP request.

For an ARP request all the fields are filled in except the target hardware address. When a system receives an ARP request directed to it, it fills in its hardware address, swaps the two sender addresses with the two target addresses, sets the *op* field to 2, and sends the reply.

4.5 ARP Examples

In this section we'll use the `tcpdump` command to see what really happens with ARP when we execute normal TCP utilities such as Telnet. Appendix A contains additional details on the `tcpdump` program.

Normal Example

To see the operation of ARP we'll execute the `telnet` command, connecting to the discard server.

```
bsdi % arp -a                          verify ARP cache is empty
bsdi % telnet svr4 discard             connect to the discard server
Trying 140.252.13.34...
Connected to svr4.
Escape character is '^]'.
^]                                     type Control, right bracket to get Telnet client prompt
telnet> quit                           and terminate
Connection closed.
```

While this is happening we run the `tcpdump` command on another system (`sun`) with the `-e` option. This displays the hardware addresses (which in our examples are 48-bit Ethernet addresses).

```
1  0.0                 0:0:c0:6f:2d:40 ff:ff:ff:ff:ff:ff arp 60:
                       arp who-has svr4 tell bsdi
2  0.002174 (0.0022)   0:0:c0:c2:9b:26 0:0:c0:6f:2d:40 arp 60:
                       arp reply svr4 is-at 0:0:c0:c2:9b:26

3  0.002831 (0.0007)   0:0:c0:6f:2d:40 0:0:c0:c2:9b:26 ip 60:
                       bsdi.1030 > svr4.discard: S 596459521:596459521(0)
                       win 4096 <mss 1024> [tos 0x10]
4  0.007834 (0.0050)   0:0:c0:c2:9b:26 0:0:c0:6f:2d:40 ip 60:
                       svr4.discard > bsdi.1030: S 3562228225:3562228225(0)
                       ack 596459522 win 4096 <mss 1024>
5  0.009615 (0.0018)   0:0:c0:6f:2d:40 0:0:c0:c2:9b:26 ip 60:
                       bsdi.1030 > svr4.discard: . ack 1 win 4096 [tos 0x10]
```

Figure 4.4 ARP request and ARP reply generated by TCP connection request.

Figure A.3 in Appendix A contains the raw output from `tcpdump` used for Figure 4.4. Since this is the first example of `tcpdump` output in the text, you should review that appendix to see how we've beautified the output.

We have deleted the final four lines of the `tcpdump` output that correspond to the termination of the connection (which we cover in Chapter 18), since they're not relevant to the discussion here.

In line 1 the hardware address of the source (`bsdi`) is `0:0:c0:6f:2d:40`. The destination hardware address is `ff:ff:ff:ff:ff:ff`, which is the Ethernet broadcast address. Every Ethernet interface on the cable will receive the frame and process it, as shown in Figure 4.2.

The next output field on line 1, `arp`, means the *frame type* field is `0x0806`, specifying either an ARP request or an ARP reply.

The value 60 printed after the words `arp` and `ip` on each of the five lines is the length of the Ethernet frame. Since the size of an ARP request and ARP reply is 42 bytes (28 bytes for the ARP message, 14 bytes for the Ethernet header), each frame has been padded to the Ethernet minimum: 60 bytes.

Referring to Figure 1.7, this minimum of 60 bytes starts with and includes the 14-byte Ethernet header, but does not include the 4-byte Ethernet trailer. Some books state the minimum as 64 bytes, which includes the Ethernet trailer. We purposely did not include the 14-byte Ethernet header in the minimum of 46 bytes shown in Figure 1.7, since the corresponding maximum (1500 bytes) is what's referred to as the MTU—maximum transmission unit (Figure 2.5). We use the MTU often, because it limits the size of an IP datagram, but are normally not concerned with the minimum. Most device drivers or interface cards automatically pad an Ethernet frame to the minimum size. The IP datagrams on lines 3, 4, and 5 (containing the TCP segments) are all smaller than the minimum, and have also been padded to 60 bytes.

The next field on line 1, `arp who-has`, identifies the frame as an ARP request with the IP address of `svr4` as the target IP address and the IP address of `bsdi` as the sender

IP address. tcpdump prints the hostnames corresponding to the IP address by default. (We'll use the -n option in Section 4.7 to see the actual IP addresses in an ARP request.)

From line 2 we see that while the ARP request is broadcast, the destination address of the ARP reply is bsdi (0:0:c0:6f:2d:40). The ARP reply is sent directly to the requesting host; it is not broadcast.

tcpdump prints arp reply for this frame, along with the hostname and hardware address of the responder.

Line 3 is the first TCP segment requesting that a connection be established. Its destination hardware address is the destination host (svr4). We'll cover the details of this segment in Chapter 18.

The number printed after the line number on each line is the time (in seconds) when the packet was received by tcpdump. Each line other than the first also contains the time difference (in seconds) from the previous line, in parentheses. We can see in this figure that the time between sending the ARP request and receiving the ARP reply is 2.2 ms. The first TCP segment is sent 0.7 ms after this. The overhead involved in using ARP for dynamic address resolution in this example is less than 3 ms.

A final point from the tcpdump output is that we don't see an ARP request from svr4 before it sends its first TCP segment (line 4). While it's possible that svr4 already had an entry for bsdi in its ARP cache, normally when a system receives an ARP request, in addition to sending the ARP reply it also saves the requestor's hardware address and IP address in its own ARP cache. This is on the logical assumption that if the requestor is about to send it an IP datagram, the receiver of the datagram will probably send a reply.

ARP Request to a Nonexistent Host

What happens if the host being queried for is down or nonexistent? To see this we specify a nonexistent Internet address—the network ID and subnet ID are that of the local Ethernet, but there is no host with the specified host ID. From Figure 3.10 we see the host IDs 36 through 62 are nonexistent (the host ID of 63 is the broadcast address). We'll use the host ID 36 in this example.

```
                                                telnet to an address this time, not a hostname
bsdi % date ; telnet 140.252.13.36 ; date
Sat Jan 30 06:46:33 MST 1993
Trying 140.252.13.36...
telnet: Unable to connect to remote host: Connection timed out
Sat Jan 30 06:47:49 MST 1993           76 seconds after previous date output

bsdi % arp -a                          check the ARP cache
? (140.252.13.36) at (incomplete)
```

Figure 4.5 shows the tcpdump output.

```
1   0.0                  arp who-has 140.252.13.36 tell bsdi
2   5.509069 ( 5.5091)   arp who-has 140.252.13.36 tell bsdi
3  29.509745 (24.0007)   arp who-has 140.252.13.36 tell bsdi
```

Figure 4.5 ARP requests to a nonexistent host.

This time we didn't specify the `-e` option since we already know that the ARP requests are broadcast.

What's interesting here is to see the frequency of the ARP requests: 5.5 seconds after the first request, then again 24 seconds later. (We examine TCP's timeout and retransmission algorithms in more detail in Chapter 21.) The total time shown in the `tcpdump` output is 29.5 seconds. But the output from the `date` commands before and after the `telnet` command shows that the connection request from the Telnet client appears to have given up after about 75 seconds. Indeed, we'll see later that most BSD implementations set a limit of 75 seconds for a TCP connection request to complete.

In Chapter 18 when we see the sequence of TCP segments that is sent to establish the connection, we'll see that these ARP requests correspond one-to-one with the initial TCP SYN (synchronize) segment that TCP is trying to send.

Note that on the wire we never see the TCP segments. All we can see are the ARP requests. Until an ARP reply comes back, the TCP segments can't be sent, since the destination hardware address isn't known. If we ran `tcpdump` in a filtering mode, looking only for TCP data, there would have been no output at all.

ARP Cache Timeout

A timeout is normally provided for entries in the ARP cache. (In Section 4.8 we'll see that the `arp` command allows an entry to be placed into the cache by the administrator that will never time out.) Berkeley-derived implementations normally have a timeout of 20 minutes for a completed entry and 3 minutes for an incomplete entry. (We saw an incomplete entry in our previous example where we forced an ARP to a nonexistent host on the Ethernet.) These implementations normally restart the 20-minute timeout for an entry each time the entry is used.

> The Host Requirements RFC says that this timeout should occur even if the entry is in use, but most Berkeley-derived implementations do not do this—they restart the timeout each time the entry is referenced.

4.6 Proxy ARP

Proxy ARP lets a router answer ARP requests on one of its networks for a host on another of its networks. This fools the sender of the ARP request into thinking that the router is the destination host, when in fact the destination host is "on the other side" of the router. The router is acting as a proxy agent for the destination host, relaying packets to it from other hosts.

An example is the best way to describe proxy ARP. In Figure 3.10 we showed that the system `sun` was connected to two Ethernets. But we also noted that this wasn't really true, if you compare that figure with the one on the inside front cover. There is in fact a router between `sun` and the subnet 140.252.1, and this router performs proxy ARP to make it appear as though `sun` is actually on the subnet 140.252.1. Figure 4.6 shows the arrangement, with a Telebit NetBlazer, named `netb`, between the subnet and the host `sun`.

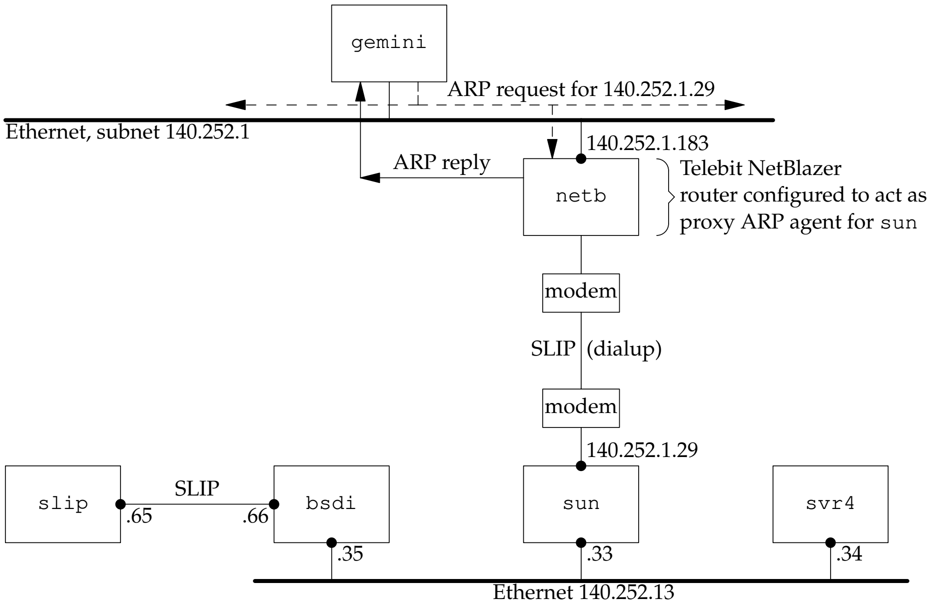

Figure 4.6 Example of proxy ARP.

When some other host on the subnet 140.252.1 (say, `gemini`) has an IP datagram to send to `sun` at address 140.252.1.29, `gemini` compares the network ID (140.252) and subnet ID (1) and since they are equal, issues an ARP request on the top Ethernet in Figure 4.6 for IP address 140.252.1.29. The router `netb` recognizes this IP address as one belonging to one of its dialup hosts, and responds with the hardware address of its Ethernet interface on the cable 140.252.1. The host `gemini` sends the IP datagram to `netb` across the Ethernet, and `netb` forwards the datagram to `sun` across the dialup SLIP link. This makes it transparent to all the hosts on the 140.252.1 subnet that host `sun` is really configured "behind" the router `netb`.

If we execute the `arp` command on the host `gemini`, after communicating with the host `sun`, we see that both IP addresses on the 140.252.1 subnet, `netb` and `sun`, map to the same hardware address. This is often a clue that proxy ARP is being used.

```
gemini % arp -a
                                        many lines for other hosts on the 140.252.1 subnet
netb (140.252.1.183) at 0:80:ad:3:6a:80
sun (140.252.1.29) at 0:80:ad:3:6a:80
```

Another detail in Figure 4.6 that we need to explain is the apparent lack of an IP address at the bottom of the router `netb` (the SLIP link). That is, why don't both ends of the dialup SLIP link have an IP address, as do both ends of the hardwired SLIP link between `bsdi` and `slip`? We noted in Section 3.8 that the destination address of the dialup SLIP link, as shown by the `ifconfig` command, was 140.252.1.183. The NetBlazer doesn't require an IP address for its end of each dialup SLIP link. (Doing so would use up more IP addresses.) Instead, it determines which dialup host is sending it packets by which serial interface the packet arrives on, so there's no need for each dialup host to use a unique IP address for its link to the router. All the dialup hosts use 140.252.1.183 as the destination address for their SLIP link.

Proxy ARP handles the delivery of datagrams to the router `sun`, but how are the other hosts on the subnet 140.252.13 handled? Routing must be used to direct datagrams to the other hosts. Specifically, routing table entries must be made somewhere on the 140.252 network that point all datagrams destined to either the subnet 140.252.13, or the specific hosts on that subnet, to the router `netb`. This router then knows how to get the datagrams to their final destination, by sending them through the router `sun`.

Proxy ARP is also called *promiscuous ARP* or the *ARP hack*. These names are from another use of proxy ARP: to hide two physical networks from each other, with a router between the two. In this case both physical networks can use the same network ID as long as the router in the middle is configured as a proxy ARP agent to respond to ARP requests on one network for a host on the other network. This technique has been used in the past to "hide" a group of hosts with older implementations of TCP/IP on a separate physical cable. Two common reasons for separating these older hosts are their inability to handle subnetting and their use of the older broadcasting address (a host ID of all zero bits, instead of the current standard of a host ID with all one bits).

4.7 Gratuitous ARP

Another feature of ARP that we can watch is called *gratuitous ARP*. It occurs when a host sends an ARP request looking for its own IP address. This is usually done when the interface is configured at bootstrap time.

In our internet, if we bootstrap the host `bsdi` and run `tcpdump` on the host `sun`, we see the packet shown in Figure 4.7.

```
1  0.0                 0:0:c0:6f:2d:40 ff:ff:ff:ff:ff:ff arp 60:
                       arp who-has 140.252.13.35 tell 140.252.13.35
```

Figure 4.7 Example of gratuitous ARP.

(We specified the `-n` flag for `tcpdump` to print numeric dotted-decimal addresses, instead of hostnames.) In terms of the fields in the ARP request, the sender's protocol address and the target's protocol address are identical: 140.252.13.35 for host `bsdi`. Also, the source address in the Ethernet header, `0:0:c0:6f:2d:40` as shown by `tcpdump`, equals the sender's hardware address (from Figure 4.4).

Gratuitous ARP provides two features.

1. It lets a host determine if another host is already configured with the same IP address. The host `bsdi` is not expecting a reply to this request. But if a reply is received, the error message "duplicate IP address sent from Ethernet address: a:b:c:d:e:f" is logged on the console. This is a warning to the system administrator that one of the systems is misconfigured.

2. If the host sending the gratuitous ARP has just changed its hardware address (perhaps the host was shut down, the interface card replaced, and then the host was rebooted), this packet causes any other host on the cable that has an entry in its cache for the old hardware address to update its ARP cache entry accordingly. A little known fact of the ARP protocol [Plummer 1982] is that if a host receives an ARP request for an IP address that is already in the receiver's cache, then that cache entry is updated with the sender's hardware address (e.g., Ethernet address) from the ARP request. This is done for any ARP request received by the host. (Recall that ARP requests are broadcast, so this is done by all hosts on the network each time an ARP request is sent.)

   [Bhide, Elnozahy, and Morgan 1991] describe an application that can use this feature of ARP to allow a backup file server to take over from a failed server by issuing a gratuitous ARP request with the backup's hardware address and the failed server's IP address. This causes all packets destined for the failed server to be sent to the backup instead, without the client applications being aware that the original server has failed.

   Unfortunately the authors then decided against this approach, since it depends on the correct implementation of ARP on all types of clients. They obviously encountered client implementations that did not implement ARP according to its specification.

   Monitoring all the systems on the author's subnet shows that SunOS 4.1.3 and 4.4BSD both issue gratuitous ARPs when bootstrapping, but SVR4 does not.

4.8 `arp` Command

We've used this command with the `-a` flag to display all the entries in the ARP cache. Other options are provided.

The superuser can specify the `-d` option to delete an entry from the ARP cache. (This was used before running a few of the examples, to let us see the ARP exchange.)

Entries can also be added using the `-s` option. It requires a *hostname* and an Ethernet *address*: the IP address corresponding to the *hostname*, and the Ethernet *address* are added to the cache. This entry is made permanent (i.e., it won't time out from the cache) unless the keyword `temp` appears at the end of the command line.

The keyword `pub` at the end of a command line with the `-s` option causes the system to act as an ARP agent for that host. The system will answer ARP requests for the IP address corresponding to the *hostname*, replying with the specified Ethernet *address*. If the advertised *address* is the system's own, then this system is acting as a proxy ARP agent for the specified *hostname*.

4.9 Summary

ARP is a basic protocol in almost every TCP/IP implementation, but it normally does its work without the application or the system administrator being aware. The ARP cache is fundamental to its operation, and we've used the arp command to examine and manipulate the cache. Each entry in the cache has a timer that is used to remove both incomplete and completed entries. The arp command displays and modifies entries in the ARP cache.

We followed through the normal operation of ARP along with specialized versions: proxy ARP (when a router answers ARP requests for hosts accessible on another of the router's interfaces) and gratuitous ARP (sending an ARP request for your own IP address, normally when bootstrapping).

Exercises

4.1 In the commands we typed to generate the output shown in Figure 4.4 (p. 58), what would happen if, after verifying that the local ARP cache was empty, we type the command

```
bsdi % rsh svr4 arp -a
```

to verify that the ARP cache is also empty on the destination host? (This command causes the arp -a command to be executed on the host svr4.)

4.2 Describe a test to determine if a given host handles a received gratuitous ARP request correctly.

4.3 Step 7 in Section 4.2 can take a while (milliseconds) because a packet is sent and ARP then waits for the response. How do you think ARP handles multiple datagrams that arrive from IP for the same destination address during this period?

4.4 At the end of Section 4.5 we mentioned that the Host Requirements RFC and Berkeley-derived implementations differ in their handling of the timeout of an active ARP entry. What happens if we're on a Berkeley-derived client and keep trying to contact a server host that's been taken down to replace its Ethernet board? Does this change if the server issues a gratuitous ARP when it bootstraps?