5

RARP: Reverse Address Resolution Protocol

5.1 Introduction

When a system with a local disk is bootstrapped it normally obtains its IP address from a configuration file that's read from a disk file. But a system without a disk, such as an X terminal or a diskless workstation, needs some other way to obtain its IP address.

Each system on a network has a unique hardware address, assigned by the manufacturer of the network interface. The principle of RARP is for the diskless system to read its unique hardware address from the interface card and send an RARP request (a broadcast frame on the network) asking for someone to reply with the diskless system's IP address (in an RARP reply).

While the concept is simple, the implementation is often harder than ARP for reasons described later in this chapter. The official specification of RARP is RFC 903 [Finlayson et al. 1984].

5.2 RARP Packet Format

The format of an RARP packet is almost identical to an ARP packet (Figure 4.3, p. 56). The only differences are that the *frame type* is 0x8035 for an RARP request or reply, and the *op* field has a value of 3 for an RARP request and 4 for an RARP reply.

As with ARP, the RARP request is broadcast and the RARP reply is normally unicast.

5.3 RARP Examples

In our internet we can force the host sun to bootstrap from the network, instead of its local disk. If we run an RARP server and tcpdump on the host bsdi we get the output shown in Figure 5.1. We use the -e flag to have tcpdump print the hardware addresses:

```
1  0.0                  8:0:20:3:f6:42 ff:ff:ff:ff:ff:ff rarp 60:
                        rarp who-is 8:0:20:3:f6:42 tell 8:0:20:3:f6:42

2  0.13 (0.13)          0:0:c0:6f:2d:40 8:0:20:3:f6:42 rarp 42:
                        rarp reply 8:0:20:3:f6:42 at sun

3  0.14 (0.01)          8:0:20:3:f6:42 0:0:c0:6f:2d:40 ip 65:
                        sun.26999 > bsdi.tftp: 23 RRQ "8CFC0D21.SUN4C"
```

Figure 5.1 RARP request and reply.

The RARP request is broadcast (line 1) and the RARP reply on line 2 is unicast. The output on line 2, at sun, means the RARP reply contains the IP address for the host sun (140.252.13.33).

On line 3 we see that once sun receives its IP address, it issues a TFTP read-request (RRQ) for the file 8CFC0D21.SUN4C. (TFTP is the Trivial File Transfer Protocol. We describe it in more detail in Chapter 15.) The eight hexadecimal digits in the filename are the hex representation of the IP address 140.252.13.33 for the host sun. This is the IP address that was returned in the RARP reply. The remainder of the filename, SUN4C, indicates the type of system being bootstrapped.

tcpdump says that line 3 is an IP datagram of length 65, and not a UDP datagram (which it really is), because we are running tcpdump with the -e flag, to see the hardware-level addresses. Another point to notice in Figure 5.1 is that the length of the Ethernet frame on line 2 appears to be shorter than the minimum (which we said was 60 bytes in Section 4.5.) The reason is that we are running tcpdump on the system that is sending this Ethernet frame (bsdi). The application, rarpd, writes 42 bytes to the BSD Packet Filter device (14 bytes for the Ethernet header and 28 bytes for the RARP reply) and this is what tcpdump receives a copy of. But the Ethernet device driver pads this short frame to the minimum size for transmission (60). Had we been running tcpdump on another system, the length would have been 60.

We can see in this example that when this diskless system receives its IP address in an RARP reply, it issues a TFTP request to read a bootstrap image. At this point we won't go into additional detail about how diskless systems bootstrap themselves. (Chapter 16 describes the bootstrap sequence of a diskless X terminal using RARP, BOOTP, and TFTP.)

Figure 5.2 shows the resulting packets if there is no RARP server on the network. The destination address of each packet is the Ethernet broadcast address. The Ethernet address following who-is is the target hardware address, and the Ethernet address following tell is the sender's hardware address.

Note the frequency of the retransmissions. The first retransmission occurs after 6.55 seconds and then increases to 42.80 seconds, then goes down to 5.34 seconds, then 6.55, and then works its way back to 42.79 seconds. This continues indefinitely. If we

```
1     0.0              8:0:20:3:f6:42 ff:ff:ff:ff:ff:ff rarp 60:
                       rarp who-is 8:0:20:3:f6:42 tell 8:0:20:3:f6:42
2     6.55 ( 6.55)     8:0:20:3:f6:42 ff:ff:ff:ff:ff:ff rarp 60:
                       rarp who-is 8:0:20:3:f6:42 tell 8:0:20:3:f6:42
3    15.52 ( 8.97)     8:0:20:3:f6:42 ff:ff:ff:ff:ff:ff rarp 60:
                       rarp who-is 8:0:20:3:f6:42 tell 8:0:20:3:f6:42
4    29.32 (13.80)     8:0:20:3:f6:42 ff:ff:ff:ff:ff:ff rarp 60:
                       rarp who-is 8:0:20:3:f6:42 tell 8:0:20:3:f6:42
5    52.78 (23.46)     8:0:20:3:f6:42 ff:ff:ff:ff:ff:ff rarp 60:
                       rarp who-is 8:0:20:3:f6:42 tell 8:0:20:3:f6:42
6    95.58 (42.80)     8:0:20:3:f6:42 ff:ff:ff:ff:ff:ff rarp 60:
                       rarp who-is 8:0:20:3:f6:42 tell 8:0:20:3:f6:42
7   100.92 ( 5.34)     8:0:20:3:f6:42 ff:ff:ff:ff:ff:ff rarp 60:
                       rarp who-is 8:0:20:3:f6:42 tell 8:0:20:3:f6:42
8   107.47 ( 6.55)     8:0:20:3:f6:42 ff:ff:ff:ff:ff:ff rarp 60:
                       rarp who-is 8:0:20:3:f6:42 tell 8:0:20:3:f6:42
9   116.44 ( 8.97)     8:0:20:3:f6:42 ff:ff:ff:ff:ff:ff rarp 60:
                       rarp who-is 8:0:20:3:f6:42 tell 8:0:20:3:f6:42
10  130.24 (13.80)     8:0:20:3:f6:42 ff:ff:ff:ff:ff:ff rarp 60:
                       rarp who-is 8:0:20:3:f6:42 tell 8:0:20:3:f6:42
11  153.70 (23.46)     8:0:20:3:f6:42 ff:ff:ff:ff:ff:ff rarp 60:
                       rarp who-is 8:0:20:3:f6:42 tell 8:0:20:3:f6:42
12  196.49 (42.79)     8:0:20:3:f6:42 ff:ff:ff:ff:ff:ff rarp 60:
                       rarp who-is 8:0:20:3:f6:42 tell 8:0:20:3:f6:42
```

Figure 5.2 RARP requests with no RARP server on the network.

calculate the differences between each timeout interval we see a doubling effect: from 5.34 to 6.55 is 1.21 seconds, from 6.55 to 8.97 is 2.42 seconds, from 8.97 to 13.80 is 4.83 seconds, and so on. When the timeout interval reaches some limit (greater than 42.80 seconds) it's reset to 5.34 seconds.

Increasing the timeout value like this is a better approach than using the same value each time. In Figure 6.8 we'll see one wrong way to perform timeout and retransmission, and in Chapter 21 we'll see TCP's method.

5.4 RARP Server Design

While the concept of RARP is simple, the design of an RARP server is system dependent and complex. Conversely, providing an ARP server is simple, and is normally part of the TCP/IP implementation in the kernel. Since the kernel knows its IP addresses and hardware addresses, when it receives an ARP request for one of its IP addresses, it just replies with the corresponding hardware address.

RARP Servers as User Processes

The complication with an RARP server is that the server normally provides the mapping from a hardware address to an IP address for many hosts (all the diskless systems on the network). This mapping is contained in a disk file (normally `/etc/ethers` on

Unix systems). Since kernels normally don't read and parse disk files, the function of an RARP server is provided as a user process, not as part of the kernel's TCP/IP implementation.

To further complicate matters, RARP requests are transmitted as Ethernet frames with a specific Ethernet frame type field (0x8035 from Figure 2.1.) This means an RARP server must have some way of sending and receiving Ethernet frames of this type. In Appendix A we describe how the BSD Packet Filter, Sun's Network Interface Tap, and the SVR4 Data Link Provider Interface can be used to receive these frames. Since the sending and receiving of these frames is system dependent, the implementation of an RARP server is tied to the system.

Multiple RARP Servers per Network

Another complication is that RARP requests are sent as hardware-level broadcasts, as shown in Figure 5.2. This means they are not forwarded by routers. To allow diskless systems to bootstrap even when the RARP server host is down, multiple RARP servers are normally provided on a single network (e.g., a single cable).

As the number of servers increases (to provide redundancy), the network traffic increases, since every server sends an RARP reply for every RARP request. The diskless system that sent the RARP request normally uses the first RARP reply that it receives. (We never had this problem with ARP, because only a single host sends an ARP reply.) Furthermore, there is a chance that each RARP server can try to respond at about the same time, increasing the probability of collisions on an Ethernet.

5.5 Summary

RARP is used by many diskless systems to obtain their IP address when bootstrapped. The RARP packet format is nearly identical to the ARP packet. An RARP request is broadcast, identifying the sender's hardware address, asking for anyone to respond with the sender's IP address. The reply is normally unicast.

Problems with RARP include its use of a link-layer broadcast, preventing most routers from forwarding an RARP request, and the minimal information returned: just the system's IP address. In Chapter 16 we'll see that BOOTP returns more information for the diskless system that is bootstrapping: its IP address, the name of a host to bootstrap from, and so on.

While the RARP concept is simple, the implementation of an RARP server is system dependent. Hence not all TCP/IP implementations provide an RARP server.

Exercises

5.1 Is a separate *frame type* field required for RARP? Could the same value be used for ARP and RARP 0x0806?

5.2 With multiple RARP servers on a network, how can they prevent their responses from colliding with each on the network?