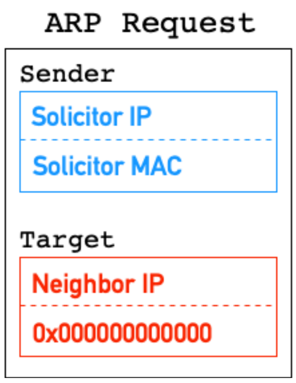

Figure 3-6005. Conceptual structure of ARP Request packet's header.

ARP PDU field number	Semantics of encoding	Example	Width in bits
1	Sender's hardware type	Ethernet, DECNet, etc	16
2	Sender stack's requesting protocol	IPv4	16
3	Byte length of hardware addresses	6 for Ethernet MAC addresses	8
4	Byte length of the sender's stack requesting protocol	4 for IPv4 addresses	8
5	Operation	Request, Response	16
6	MAC address of sender	Ethernet MAC addr	Specified in field # 1, above
7	Sender stack's protocol address	IPv4 addr	Specified in field # 2, above
8	MAC address of target	Ethernet MAC addr	Specified in field # 3, above
9	Target stack's protocol address	IPv4 addr	Specified in field # 4, above

Table 3-6010. Detailed fields of a generic ARP packet from RFC 826.

Express practice. Capturing a basic ARP transaction.

Capturing a basic ARP transaction is straightforward by using tcpdump; solicitor host at IP 192.168.86.2 seeks the IP-to-MAC mapping to 192.168.86.11, herein formalized as *map(192.168.86.11)*. Interface enp4s0 has been configured with IP address 192.168.86.11. The first packet printed out is the ARP Request. Observe that tcpdump prints a hexadecimal dump of the packet's bytes; furthermore, it prints a text with a more readable interpretation of the latter bytes. Most times this text suffices. If more specific detail is needed, then the hex dump must be interpreted as we do in the express practice ensuing this one. The aforementioned text provided by the tcpdump arp inspector follows:

```
Request who-has 192.168.86.11 tell 192.168.86.2
```

The ARP Reply elicited at the neighbor carries map(192.168.86.11) along with the solicitor's map as is specified in ARP:

```
        Reply 192.168.86.11 is-at e0:d5:5e:dd:ed:2a
```

The Request is sent to the broadcast address, whereas the Reply is sent back to the unicast solicitor's MAC address.

```
$ tcpdump -i enp4s0 -vv -X -n -e arp
tcpdump: listening on enp4s0, link-type EN10MB (Ethernet), capture size
262144 bytes
10:24:12.460337  18:d6:c7:02:8b:59  >  ff:ff:ff:ff:ff:ff,  ethertype  ARP
(0x0806),  length  42:  Ethernet  (len  6),  IPv4  (len  4),  Request  who-has
192.168.86.11 tell 192.168.86.2, length 28
        0x0000:  0001 0800 0604 0001 18d6 c702 8b59 c0a8  .............Y..
        0x0010:  5602 0000 0000 0000 c0a8 560b            V.........V.

10:24:12.460493  e0:d5:5e:dd:ed:2a  >  18:d6:c7:02:8b:59,  ethertype  ARP
(0x0806), length 60: Ethernet (len 6), IPv4 (len 4), Reply 192.168.86.11 is-
at e0:d5:5e:dd:ed:2a, length 46
        0x0000:  0001 0800 0604 0002 e0d5 5edd ed2a c0a8  ..........^..*..
        0x0010:  560b 18d6 c702 8b59 c0a8 5602 0000 0000  V......Y..V.....
        0x0020:  0000 0000 0000 0000 0000 0000 0000
```

Express practice. Analyzing the tcpdump hexdump to a basic ARP transaction.

The IP block assigned to the local network used in the present practice is 192.168.1.0/24. Its only default router is at 192.168.1.1. The host at 192.168.1.89 (H_a) needs to communicate with the host at 192.168.1.171 (H_b), consequently, it needs to find out the MAC address of the interface used by H_b. Ha resolves the MAC of H_b by initiating a new ARP resolution transaction that will when H_a receives the response furnished by H_b.

As usual, we'll run a convenient tcpdump trace at host H_a that will allow us to watch the outgoing ARP request as well as the ARP response provided by H_b. The tcpdump filter used will allow icmp also. That is such that we can better establish the *potentially-causal* relationship between the issuing of the ping command and the local stack's IP requesting ARP to resolve the remote IP into its interface's MAC. First, we start tcpdump on one terminal:

```
$ tcpdump -i eno1 -n -ex -XX -vvv arp or icmp
tcpdump: listening on eno1, link-type EN10MB (Ethernet), snapshot length
262144 bytes
```

On another terminal, ping is executed. The ping application formats a new ICMP Echo Request message that is submitted to IP over the local stack of H_a.

```
$ tcpdump -c 1 192.168.1.171
```

IP delegates the request to its ancillary ARP protocol for solving the MAC address of the interface at H_b. Consequently, ARP proceeds to sending an ARP Request encapsulated into an Ethernet frame[4]:

```
14:12:48.891571 e0:d5:5e:d8:86:a1 > ff:ff:ff:ff:ff:ff, ethertype ARP
(0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Request who-has
192.168.1.171 tell 192.168.1.89, length 28
        0x0000:  ffff ffff ffff e0d5 5ed8 86a1 0806 0001  ........^.......
        0x0010:  0800 0604 0001 e0d5 5ed8 86a1 c0a8 0159  ........^......Y
        0x0020:  0000 0000 0000 c0a8 01ab                 ..........
```

Ethernet Header:

- Source MAC. The sender's interface MAC address (`e0:d5:5e:d8:86:a1`)
- Destination MAC. Broadcast (`ff:ff:ff:ff:ff:ff`)
- Ethertype. ARP protocol which Ethertype is `0x0806`

ARP Payload interpreted according to the ARP header summary included above (Table 3-6010):

- Datalink type: Ethernet
- Network protocol: IPv4
- Byte length of HW addresses: 6
- Byte length of Network addresses: 4
- Opcode: Request
- Sender identification
  - HW = `0xe0d5 5e27 2f72`
  - IP = `0xc0a80159 which expressed in DDN is 192.168.1.89`
- Target identification
  - HW = `0x0000 0000 0000`
  - IP = `0xc0a801ab which expressed in DDN is 192.168.1.171`

Host H_b receives the ARP Request and reacts by sending back an ARP Reply carrying the requested IP-to-MAC mapping. The ARP Reply packet, encapsulated into an Ethernet frame is included here:

```
14:12:48.892004 34:64:a9:39:ca:c8 > e0:d5:5e:d8:86:a1, ethertype ARP
(0x0806), length 60: Ethernet (len 6), IPv4 (len 4), Reply 192.168.1.171
is-at 34:64:a9:39:ca:c8, length 46
        0x0000:  e0d5 5ed8 86a1 3464 a939 cac8 0806 0001  ..^...4d.9......
        0x0010:  0800 0604 0002 3464 a939 cac8 c0a8 01ab  ......4d.9......
        0x0020:  e0d5 5ed8 86a1 c0a8 0159 0000 0000 0000  ..^......Y......
        0x0030:  0000 0000 0000 0000 0000 0000            ............
```

An interpretation similar to the one given above for the ARP Request is included here (Check with table 3-6010 for the ARP header).

Ethernet Header:

- Source MAC. The neighbor's interface MAC address (`34:64:a9:39:ca:c8`); this time, the neighbor node is the sending node.
- Destination MAC. Unicast address to the Solicitor node (`e0:d5:5e:d8:86:a1`)

[4] The Ethernet frame header is printed out in blue on the trace hexdump that follows

- Ethertype. ARP protocol which Ethertype is `0x0806`
    - Datalink type: Ethernet
    - Network protocol: IPv4
    - Byte length of HW addresses: 6
    - Byte length of Network addresses: 4
    - Opcode: Reply (Bytes 6 and 7 of ARP packet: `0002`)
    - Neighbor identification
        - HW = `0x3464 a939 cac8 c0a8`
        - IP = `0xc0a801ab which expressed in DDN is 192.168.1.171`
    - Solicitor identification
        - HW = `0x0000 0000 0000`
        - IP = `0xc0a80159 which expressed in DDN is 192.168.1.89`

The Datalink/Physical protocol can now transmit the Ethernet frame, which finishes the application-layer request made by the ping application. At this moment, tcpdump can printout that request. The reply will be received and printed out as well. See both frames here, below. The Ethernet header is depicted in blue, the IP packet in melon and the ICMP header and payload in black:

```
14:12:48.892018 e0:d5:5e:d8:86:a1 > 34:64:a9:39:ca:c8, ethertype IPv4
(0x0800), length 98: (tos 0x0, ttl 64, id 15022, offset 0, flags [DF],
proto ICMP (1), length 84)
    192.168.1.89 > 192.168.1.171: ICMP echo request, id 8188, seq 1, length 64
        0x0000:  3464 a939 cac8 e0d5 5ed8 86a1 0800 4500  4d.9....^.....E.
        0x0010:  0054 3aae 4000 4001 7ba6 c0a8 0159 c0a8  .T:.@.@.{....Y..
        0x0020:  01ab 0800 d603 1ffc 0001 402d 5e64 0000  ..........@-^d..
        0x0030:  0000 979a 0d00 0000 0000 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
```

```
14:12:48.892463 34:64:a9:39:ca:c8 > e0:d5:5e:d8:86:a1, ethertype IPv4
(0x0800), length 98: (tos 0x0, ttl 64, id 35775, offset 0, flags [none],
proto ICMP (1), length 84)
    192.168.1.171 > 192.168.1.89: ICMP echo reply, id 8188, seq 1, length 64
        0x0000:  e0d5 5ed8 86a1 3464 a939 cac8 0800 4500  ..^...4d.9....E.
        0x0010:  0054 8bbf 0000 4001 6a95 c0a8 01ab c0a8  .T....@.j.......
        0x0020:  0159 0000 de03 1ffc 0001 402d 5e64 0000  .Y........@-^d..
        0x0030:  0000 979a 0d00 0000 0000 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
```

Typical implementation of ARP

The ARP for IP/Ethernet described above is not the only protocol responsible for finding local network addresses provided their IP addresses. Besides ARP for IPv4, other ARPs exist such as IPv6, Netware, X.25, DECnet and others. All these protocols, in Linux, conform a subsystem known as *the neighboring subsystem.* This subsystem offers common data structures that can be used by the various protocols