Project 3 Report
ECE 5600

Nathan Tipton
A01207112
Partner: Erik Sargent

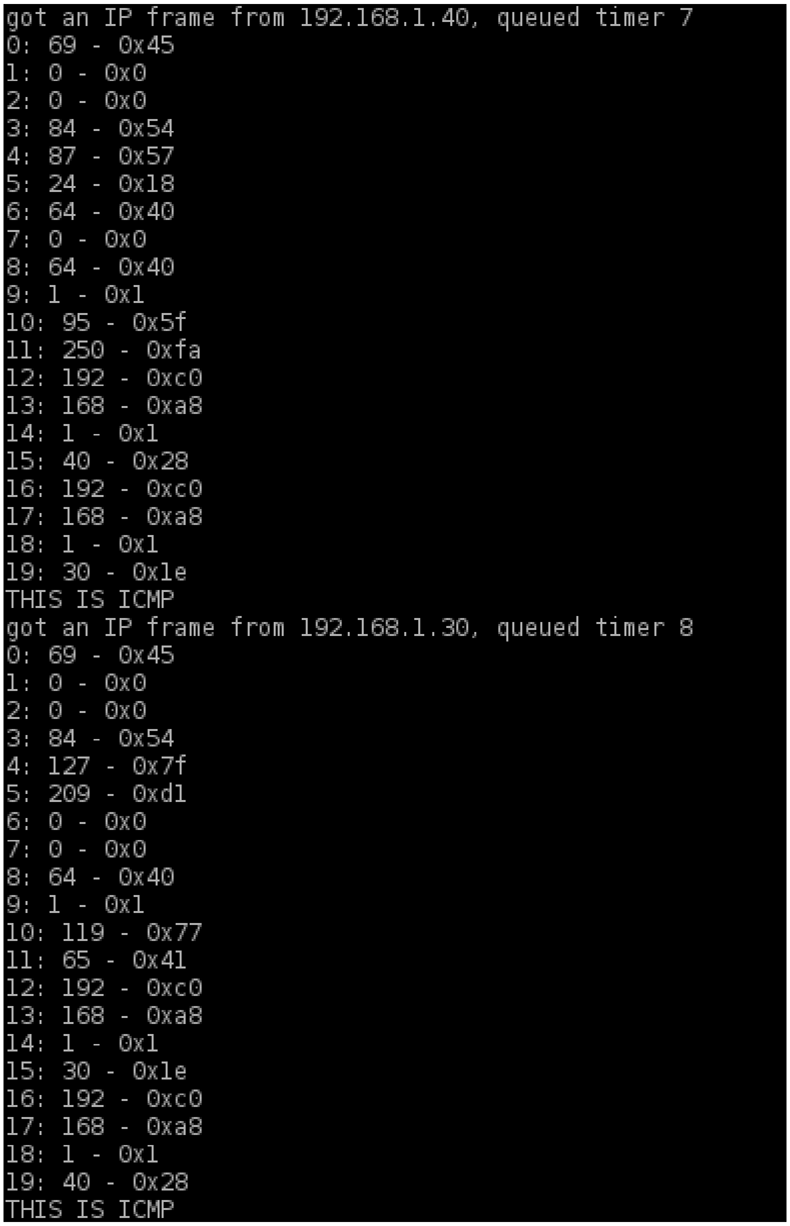
Nov 13, 2017

1 Objective

The purpose of this project is to familiarize ourselves with the structure of an IPv4. We will also understand and implement a function to send data packets using IP frame. We develop understanding of ICMP message types and implement ICMP echo. Using Wireshark we will analyze captured frames.

2 Results

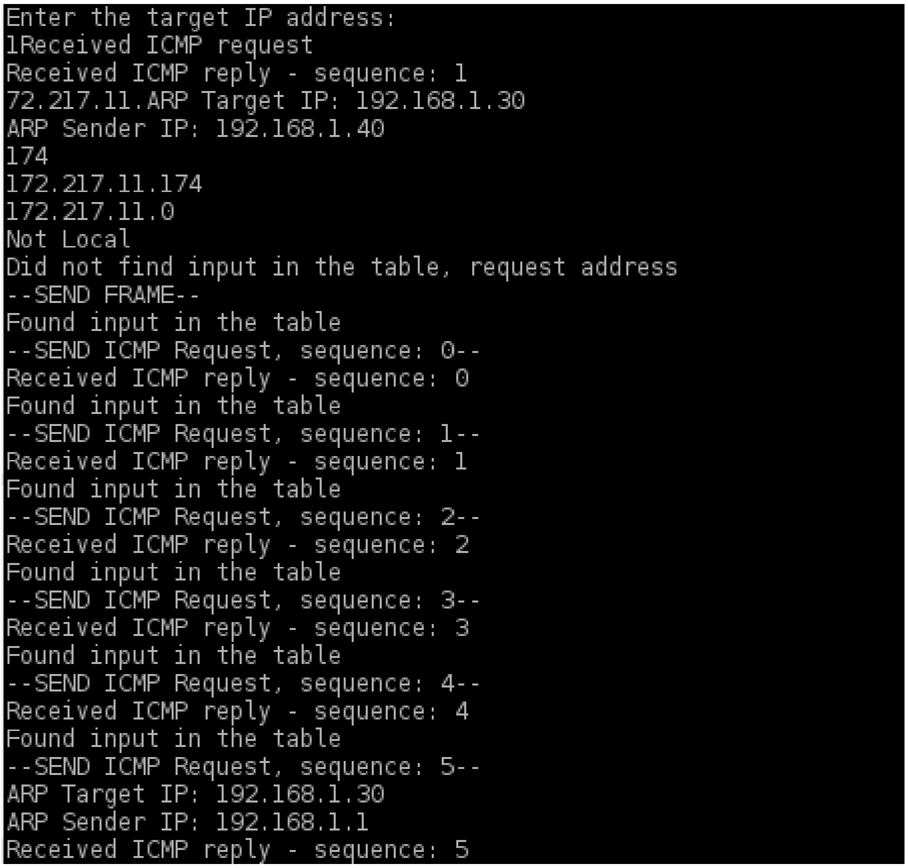
Frames on the network are captured by our program and stored in a frame buffer. Our program determines if frame is IP or ARP and sorts accordingly. When an IP is received the protocol byte is checked to see if it is ICMP. If so it is checked again to see if it is a request or reply.



```
got an IP frame from 192.168.1.40, queued timer 7
0: 69 - 0x45
1: 0 - 0x0
2: 0 - 0x0
3: 84 - 0x54
4: 87 - 0x57
5: 24 - 0x18
6: 64 - 0x40
7: 0 - 0x0
8: 64 - 0x40
9: 1 - 0x1
10: 95 - 0x5f
11: 250 - 0xfa
12: 192 - 0xc0
13: 168 - 0xa8
14: 1 - 0x1
15: 40 - 0x28
16: 192 - 0xc0
17: 168 - 0xa8
18: 1 - 0x1
19: 30 - 0x1e
THIS IS ICMP
got an IP frame from 192.168.1.30, queued timer 8
0: 69 - 0x45
1: 0 - 0x0
2: 0 - 0x0
3: 84 - 0x54
4: 127 - 0x7f
5: 209 - 0xd1
6: 0 - 0x0
7: 0 - 0x0
8: 64 - 0x40
9: 1 - 0x1
10: 119 - 0x77
11: 65 - 0x41
12: 192 - 0xc0
13: 168 - 0xa8
14: 1 - 0x1
15: 30 - 0x1e
16: 192 - 0xc0
17: 168 - 0xa8
18: 1 - 0x1
19: 40 - 0x28
THIS IS ICMP
```

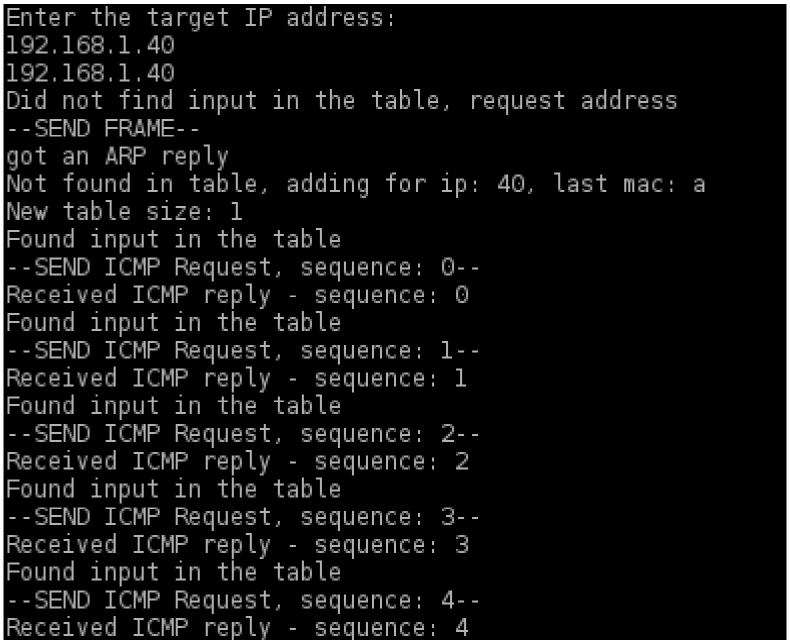
Figure 1: Screenshot of terminal identifying ICMP packet

Next, we enable the feature to send to a target IP that may nor may not be in the lab. We use the subnet mask that is found from the computer the program is running on at the time. The subnet mask is used to determine whether or not the target IP is in the lab. If it is, the program sends an ARP in order to obtain the mac address and send the IP frame. Otherwise the program sends an ARP request to the router.



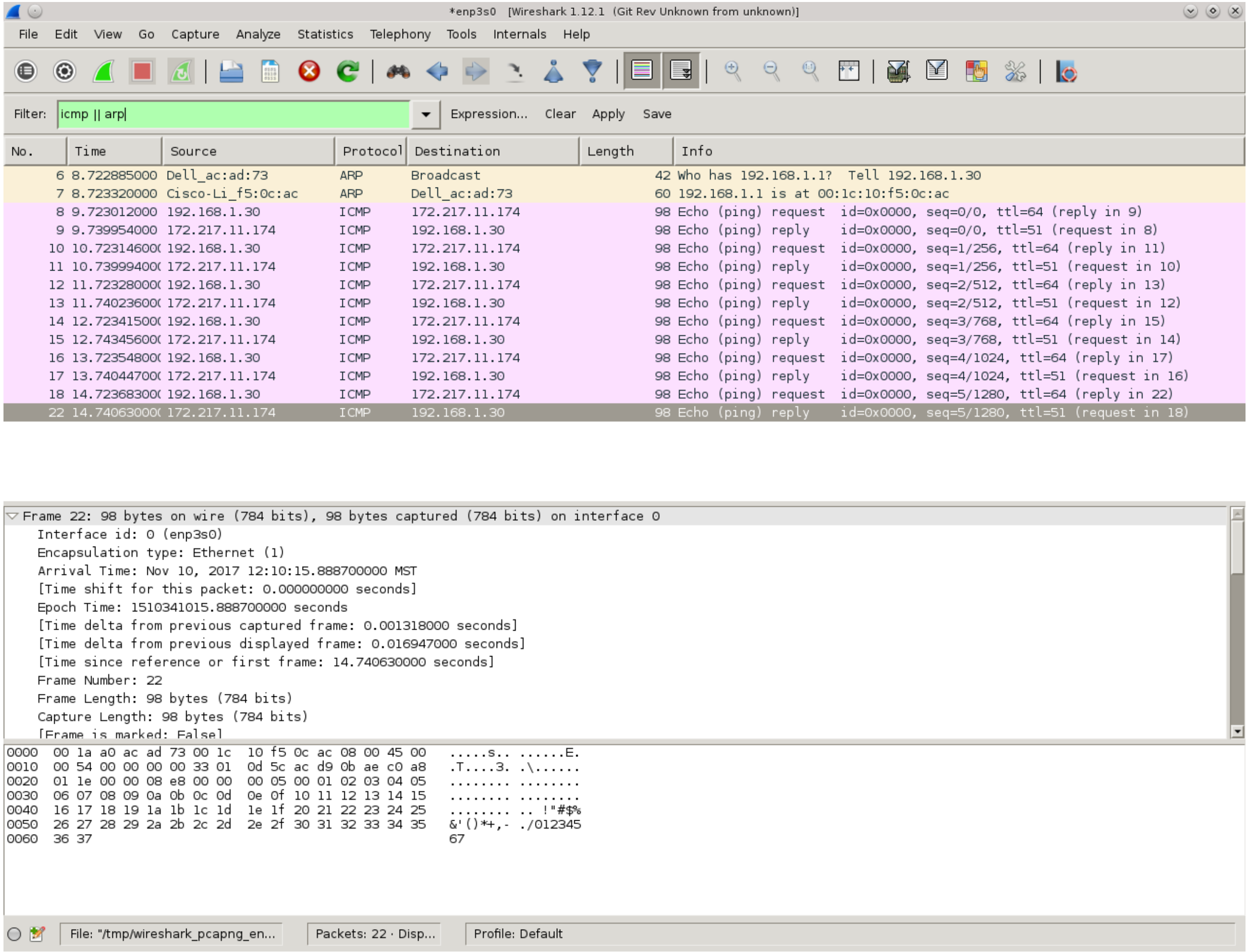
```
Enter the target IP address:
1Received ICMP request
Received ICMP reply - sequence: 1
72.217.11.ARP Target IP: 192.168.1.30
ARP Sender IP: 192.168.1.40
174
172.217.11.174
172.217.11.0
Not Local
Did not find input in the table, request address
--SEND FRAME--
Found input in the table
--SEND ICMP Request, sequence: 0--
Received ICMP reply - sequence: 0
Found input in the table
--SEND ICMP Request, sequence: 1--
Received ICMP reply - sequence: 1
Found input in the table
--SEND ICMP Request, sequence: 2--
Received ICMP reply - sequence: 2
Found input in the table
--SEND ICMP Request, sequence: 3--
Received ICMP reply - sequence: 3
Found input in the table
--SEND ICMP Request, sequence: 4--
Received ICMP reply - sequence: 4
Found input in the table
--SEND ICMP Request, sequence: 5--
ARP Target IP: 192.168.1.30
ARP Sender IP: 192.168.1.1
Received ICMP reply - sequence: 5
```

Figure 2: Target IP is not local



```
Enter the target IP address:
192.168.1.40
192.168.1.40
Did not find input in the table, request address
--SEND FRAME--
got an ARP reply
Not found in table, adding for ip: 40, last mac: a
New table size: 1
Found input in the table
--SEND ICMP Request, sequence: 0--
Received ICMP reply - sequence: 0
Found input in the table
--SEND ICMP Request, sequence: 1--
Received ICMP reply - sequence: 1
Found input in the table
--SEND ICMP Request, sequence: 2--
Received ICMP reply - sequence: 2
Found input in the table
--SEND ICMP Request, sequence: 3--
Received ICMP reply - sequence: 3
Found input in the table
--SEND ICMP Request, sequence: 4--
Received ICMP reply - sequence: 4
```

Figure 3: Target IP is local



Filter: icmp || arp

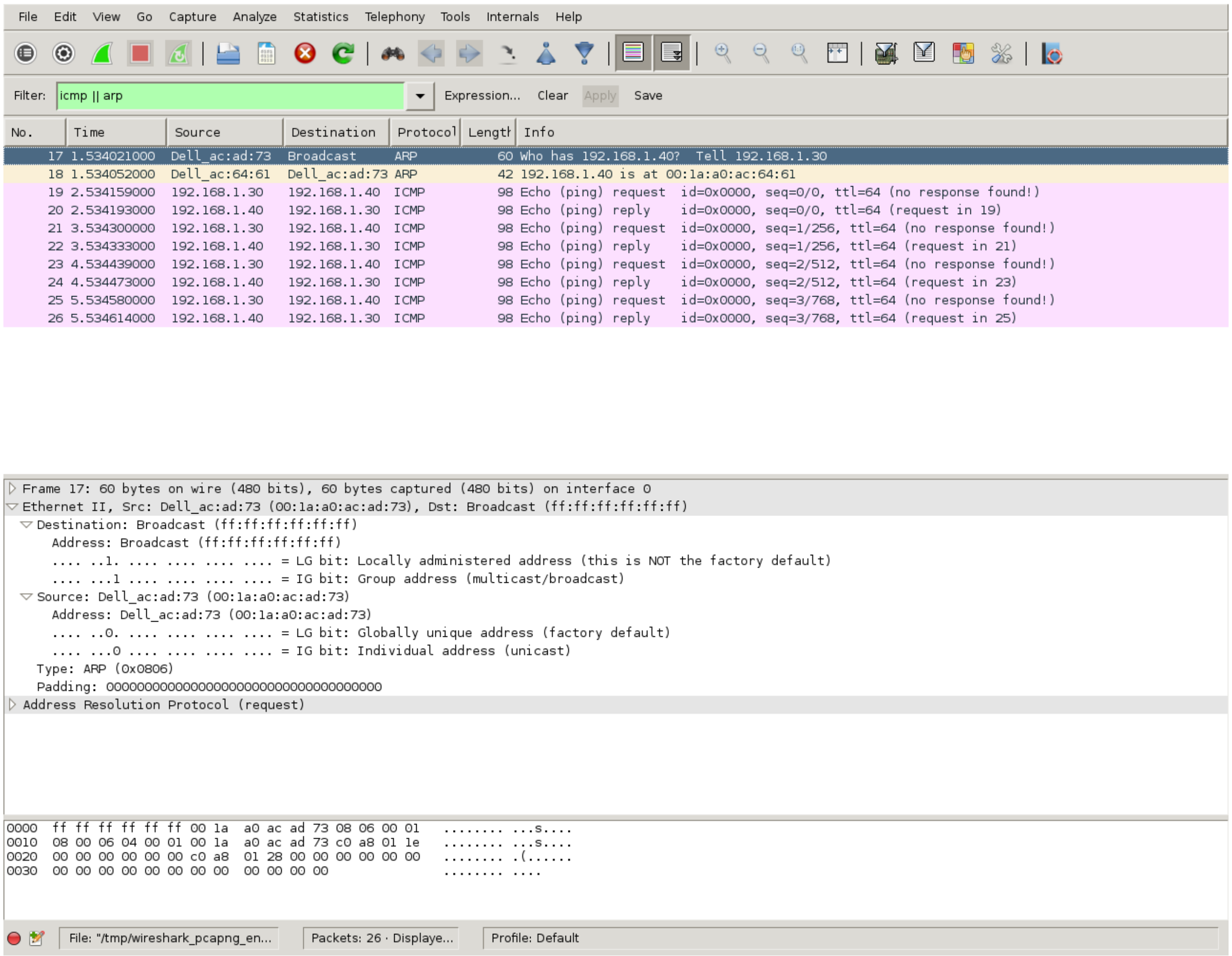
```
No.   Time          Source             Protocol Destination         Length Info
    6 8.722885000   Dell_ac:ad:73      ARP      Broadcast               42 Who has 192.168.1.1?  Tell 192.168.1.30
    7 8.723320000   Cisco-Li_f5:0c:ac  ARP      Dell_ac:ad:73           60 192.168.1.1 is at 00:1c:10:f5:0c:ac
    8 9.723012000   192.168.1.30       ICMP     172.217.11.174          98 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (reply in 9)
    9 9.739954000   172.217.11.174     ICMP     192.168.1.30            98 Echo (ping) reply    id=0x0000, seq=0/0, ttl=51 (request in 8)
   10 10.72314600(  192.168.1.30       ICMP     172.217.11.174          98 Echo (ping) request  id=0x0000, seq=1/256, ttl=64 (reply in 11)
   11 10.73999400(  172.217.11.174     ICMP     192.168.1.30            98 Echo (ping) reply    id=0x0000, seq=1/256, ttl=51 (request in 10)
   12 11.72328000(  192.168.1.30       ICMP     172.217.11.174          98 Echo (ping) request  id=0x0000, seq=2/512, ttl=64 (reply in 13)
   13 11.74023600(  172.217.11.174     ICMP     192.168.1.30            98 Echo (ping) reply    id=0x0000, seq=2/512, ttl=51 (request in 12)
   14 12.72341500(  192.168.1.30       ICMP     172.217.11.174          98 Echo (ping) request  id=0x0000, seq=3/768, ttl=64 (reply in 15)
   15 12.74345600(  172.217.11.174     ICMP     192.168.1.30            98 Echo (ping) reply    id=0x0000, seq=3/768, ttl=51 (request in 14)
   16 13.72354800(  192.168.1.30       ICMP     172.217.11.174          98 Echo (ping) request  id=0x0000, seq=4/1024, ttl=64 (reply in 17)
   17 13.74044700(  172.217.11.174     ICMP     192.168.1.30            98 Echo (ping) reply    id=0x0000, seq=4/1024, ttl=51 (request in 16)
   18 14.72368300(  192.168.1.30       ICMP     172.217.11.174          98 Echo (ping) request  id=0x0000, seq=5/1280, ttl=64 (reply in 22)
   22 14.74063000(  172.217.11.174     ICMP     192.168.1.30            98 Echo (ping) reply    id=0x0000, seq=5/1280, ttl=51 (request in 18)
```

```
▽ Frame 22: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
    Interface id: 0 (enp3s0)
    Encapsulation type: Ethernet (1)
    Arrival Time: Nov 10, 2017 12:10:15.888700000 MST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1510341015.888700000 seconds
    [Time delta from previous captured frame: 0.001318000 seconds]
    [Time delta from previous displayed frame: 0.016947000 seconds]
    [Time since reference or first frame: 14.740630000 seconds]
    Frame Number: 22
    Frame Length: 98 bytes (784 bits)
    Capture Length: 98 bytes (784 bits)
    [Frame is marked: False]
```

```
0000  00 1a a0 ac ad 73 00 1c  10 f5 0c ac 08 00 45 00   .....s.. ......E.
0010  00 54 00 00 00 00 33 01  0d 5c ac d9 0b ae c0 a8   .T....3. .\......
0020  01 1e 00 00 08 e8 00 00  00 05 00 01 02 03 04 05   ........ ........
0030  06 07 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                              67
```

File: "/tmp/wireshark_pcapng_en... | Packets: 22 · Disp... | Profile: Default

Figure 4: Wireshark screenshot of non-local target IP



Filter: icmp || arp

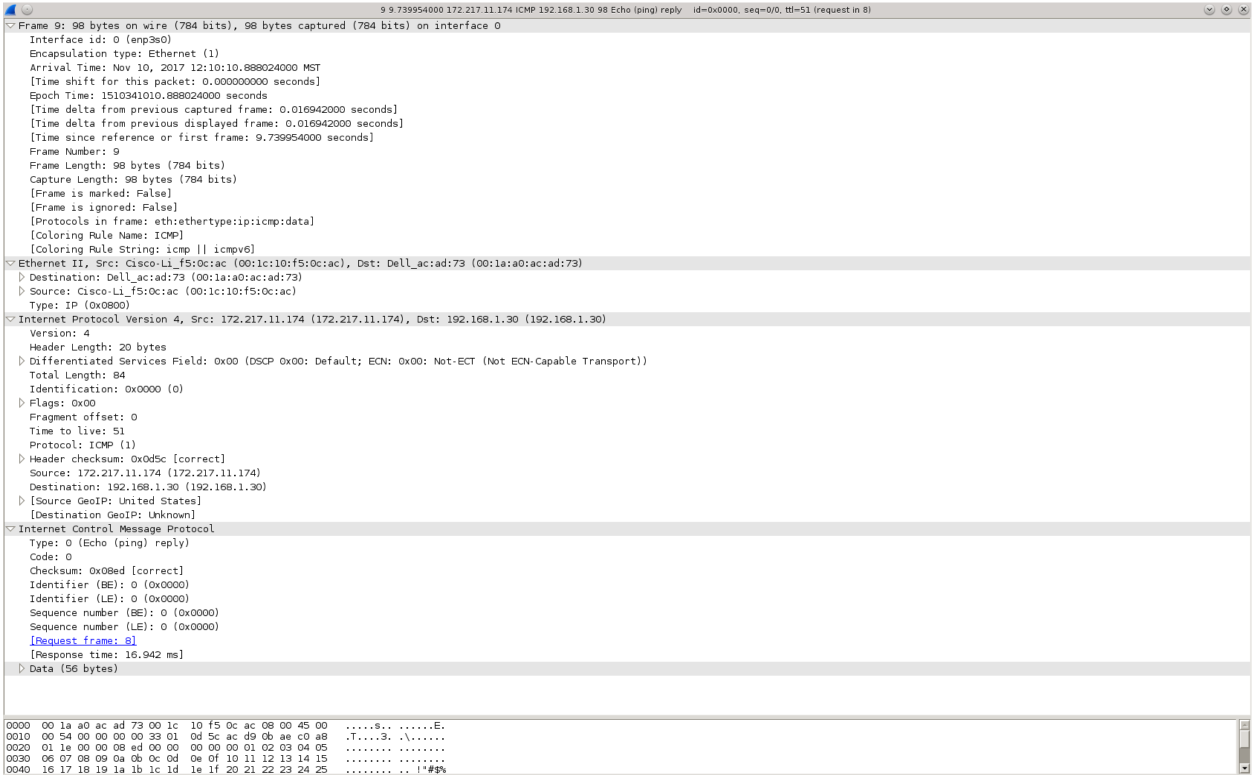
```
No.    Time         Source          Destination     Protocol Length Info
    17 1.534021000  Dell_ac:ad:73   Broadcast       ARP          60 Who has 192.168.1.40?  Tell 192.168.1.30
    18 1.534052000  Dell_ac:64:61   Dell_ac:ad:73   ARP          42 192.168.1.40 is at 00:1a:a0:ac:64:61
    19 2.534159000  192.168.1.30    192.168.1.40    ICMP         98 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    20 2.534193000  192.168.1.40    192.168.1.30    ICMP         98 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 19)
    21 3.534300000  192.168.1.30    192.168.1.40    ICMP         98 Echo (ping) request  id=0x0000, seq=1/256, ttl=64 (no response found!)
    22 3.534333000  192.168.1.40    192.168.1.30    ICMP         98 Echo (ping) reply    id=0x0000, seq=1/256, ttl=64 (request in 21)
    23 4.534439000  192.168.1.30    192.168.1.40    ICMP         98 Echo (ping) request  id=0x0000, seq=2/512, ttl=64 (no response found!)
    24 4.534473000  192.168.1.40    192.168.1.30    ICMP         98 Echo (ping) reply    id=0x0000, seq=2/512, ttl=64 (request in 23)
    25 5.534580000  192.168.1.30    192.168.1.40    ICMP         98 Echo (ping) request  id=0x0000, seq=3/768, ttl=64 (no response found!)
    26 5.534614000  192.168.1.40    192.168.1.30    ICMP         98 Echo (ping) reply    id=0x0000, seq=3/768, ttl=64 (request in 25)
```

```
▷ Frame 17: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▽ Ethernet II, Src: Dell_ac:ad:73 (00:1a:a0:ac:ad:73), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▽ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
      Address: Broadcast (ff:ff:ff:ff:ff:ff)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
      .... ...1 .... .... .... .... = IG bit: Group address (multicast/broadcast)
  ▽ Source: Dell_ac:ad:73 (00:1a:a0:ac:ad:73)
      Address: Dell_ac:ad:73 (00:1a:a0:ac:ad:73)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: ARP (0x0806)
    Padding: 000000000000000000000000000000000000
▷ Address Resolution Protocol (request)
```

```
0000  ff ff ff ff ff ff 00 1a  a0 ac ad 73 08 06 00 01   ........ ...s....
0010  08 00 06 04 00 01 00 1a  a0 ac ad 73 c0 a8 01 1e   ........ ...s....
0020  00 00 00 00 00 00 c0 a8  01 28 00 00 00 00 00 00   ........ .(......
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

File: "/tmp/wireshark_pcapng_en...   Packets: 26 · Displaye...   Profile: Default

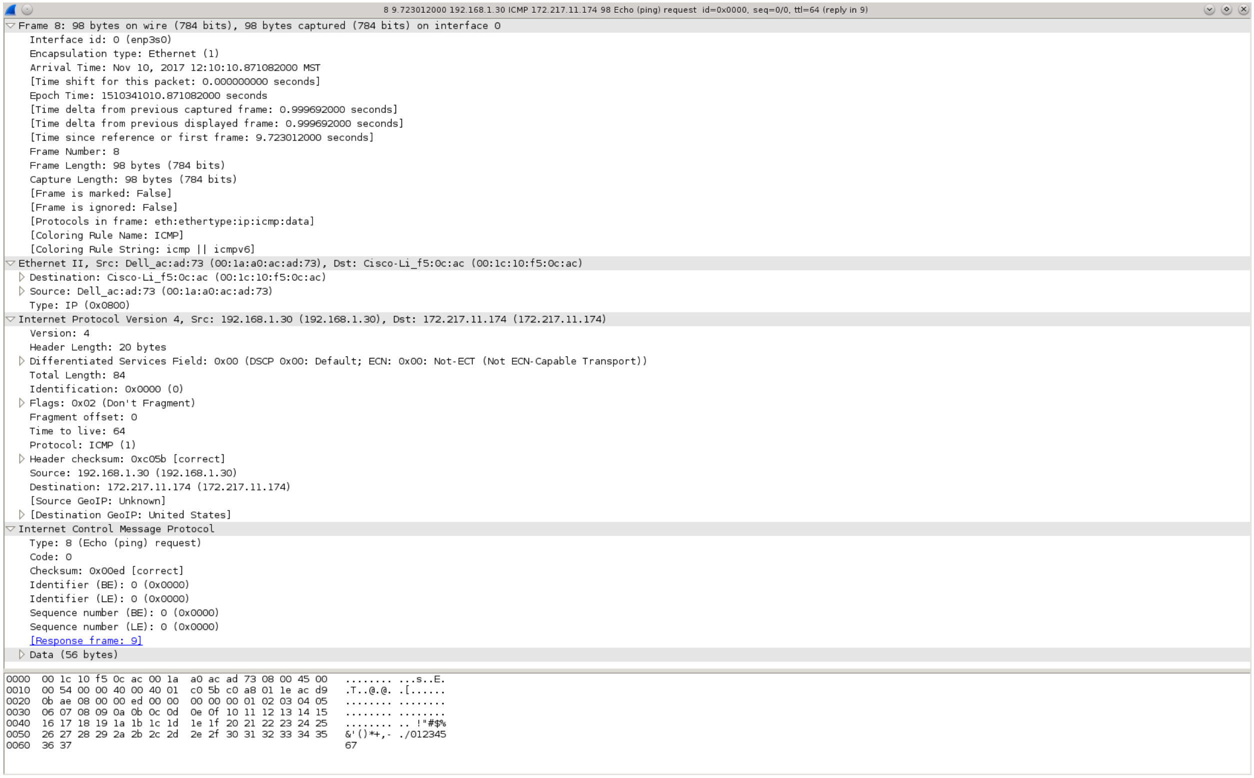
Figure 5: Wireshark screenshot of local target IP



```
9 9.739954000 172.217.11.174 ICMP 192.168.1.30 98 Echo (ping) reply    id=0x0000, seq=0/0, ttl=51 (request in 8)
▽ Frame 9: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
    Interface id: 0 (enp3s0)
    Encapsulation type: Ethernet (1)
    Arrival Time: Nov 10, 2017 12:10:10.888024000 MST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1510341010.888024000 seconds
    [Time delta from previous captured frame: 0.016942000 seconds]
    [Time delta from previous displayed frame: 0.016942000 seconds]
    [Time since reference or first frame: 9.739954000 seconds]
    Frame Number: 9
    Frame Length: 98 bytes (784 bits)
    Capture Length: 98 bytes (784 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:icmp:data]
    [Coloring Rule Name: ICMP]
    [Coloring Rule String: icmp || icmpv6]
▽ Ethernet II, Src: Cisco-Li_f5:0c:ac (00:1c:10:f5:0c:ac), Dst: Dell_ac:ad:73 (00:1a:a0:ac:ad:73)
  ▷ Destination: Dell_ac:ad:73 (00:1a:a0:ac:ad:73)
  ▷ Source: Cisco-Li_f5:0c:ac (00:1c:10:f5:0c:ac)
    Type: IP (0x0800)
▽ Internet Protocol Version 4, Src: 172.217.11.174 (172.217.11.174), Dst: 192.168.1.30 (192.168.1.30)
    Version: 4
    Header Length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 84
    Identification: 0x0000 (0)
  ▷ Flags: 0x00
    Fragment offset: 0
    Time to live: 51
    Protocol: ICMP (1)
  ▷ Header checksum: 0x0d5c [correct]
    Source: 172.217.11.174 (172.217.11.174)
    Destination: 192.168.1.30 (192.168.1.30)
  ▷ [Source GeoIP: United States]
    [Destination GeoIP: Unknown]
▽ Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x08ed [correct]
    Identifier (BE): 0 (0x0000)
    Identifier (LE): 0 (0x0000)
    Sequence number (BE): 0 (0x0000)
    Sequence number (LE): 0 (0x0000)
    [Request frame: 8]
    [Response time: 16.942 ms]
  ▷ Data (56 bytes)

0000  00 1a a0 ac ad 73 00 1c  10 f5 0c ac 08 00 45 00   .....s.. ......E.
0010  00 54 00 00 00 00 33 01  0d 5c ac d9 0b ae c0 a8   .T....3. .\......
0020  01 1e 00 00 08 ed 00 00  00 00 00 01 02 03 04 05   ........ ........
0030  06 07 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
```

Figure 6: Wireshark reply from non-local IP

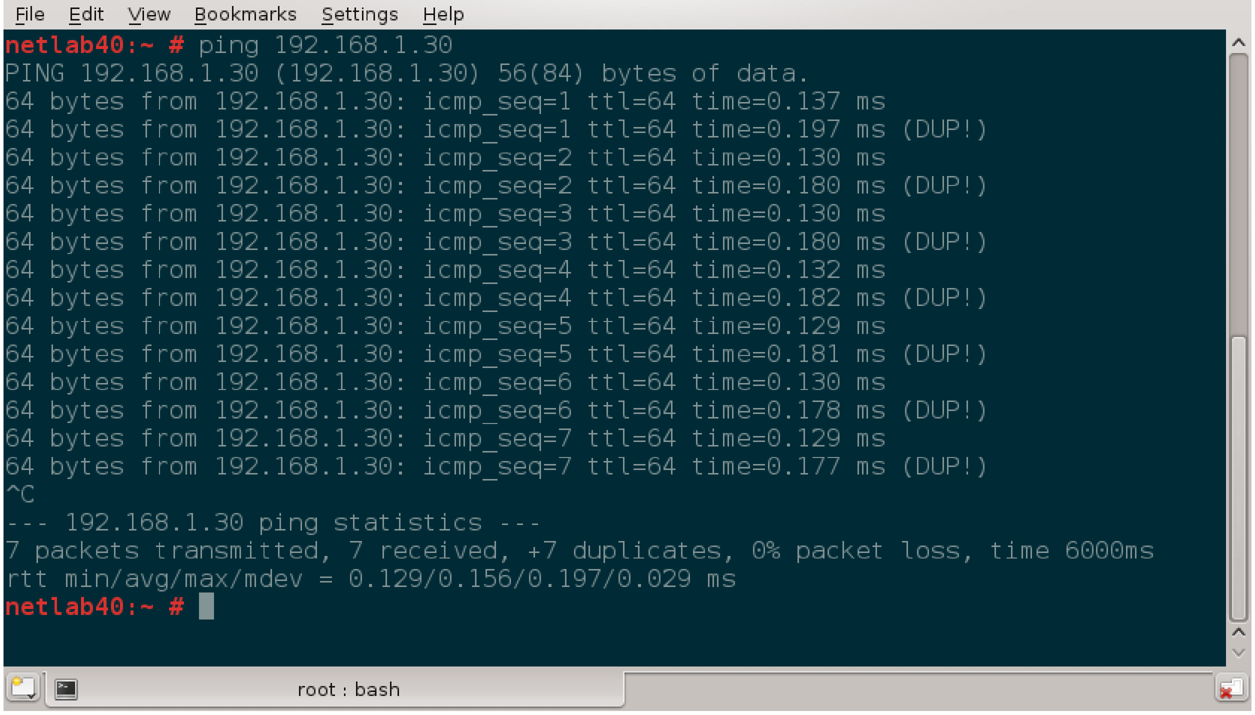


```
8 9.723012000 192.168.1.30 ICMP 172.217.11.174 98 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (reply in 9)
▽ Frame 8: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
    Interface id: 0 (enp3s0)
    Encapsulation type: Ethernet (1)
    Arrival Time: Nov 10, 2017 12:10:10.871082000 MST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1510341010.871082000 seconds
    [Time delta from previous captured frame: 0.999692000 seconds]
    [Time delta from previous displayed frame: 0.999692000 seconds]
    [Time since reference or first frame: 9.723012000 seconds]
    Frame Number: 8
    Frame Length: 98 bytes (784 bits)
    Capture Length: 98 bytes (784 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:icmp:data]
    [Coloring Rule Name: ICMP]
    [Coloring Rule String: icmp || icmpv6]
▽ Ethernet II, Src: Dell_ac:ad:73 (00:1a:a0:ac:ad:73), Dst: Cisco-Li_f5:0c:ac (00:1c:10:f5:0c:ac)
  ▷ Destination: Cisco-Li_f5:0c:ac (00:1c:10:f5:0c:ac)
  ▷ Source: Dell_ac:ad:73 (00:1a:a0:ac:ad:73)
    Type: IP (0x0800)
▽ Internet Protocol Version 4, Src: 192.168.1.30 (192.168.1.30), Dst: 172.217.11.174 (172.217.11.174)
    Version: 4
    Header Length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 84
    Identification: 0x0000 (0)
  ▷ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
  ▷ Header checksum: 0xc05b [correct]
    Source: 192.168.1.30 (192.168.1.30)
    Destination: 172.217.11.174 (172.217.11.174)
    [Source GeoIP: Unknown]
  ▷ [Destination GeoIP: United States]
▽ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x00ed [correct]
    Identifier (BE): 0 (0x0000)
    Identifier (LE): 0 (0x0000)
    Sequence number (BE): 0 (0x0000)
    Sequence number (LE): 0 (0x0000)
    [Response frame: 9]
  ▷ Data (56 bytes)

0000  00 1c 10 f5 0c ac 00 1a  a0 ac ad 73 08 00 45 00   ........ ...s..E.
0010  00 54 00 00 40 00 40 01  c0 5b c0 a8 01 1e ac d9   .T..@.@. .[......
0020  0b ae 08 00 00 ed 00 00  00 00 00 01 02 03 04 05   ........ ........
0030  06 07 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                              67
```

Figure 7: Wireshark request non-local IP

We implemented the ICMP protocol for echo reply and request. When "pinging" our PC we can see the replies from both our code and the PC. Comparing the reply from the computer and the reply from our program we can see that both replies are exactly the same.



```
netlab40:~ # ping 192.168.1.30
PING 192.168.1.30 (192.168.1.30) 56(84) bytes of data.
64 bytes from 192.168.1.30: icmp_seq=1 ttl=64 time=0.137 ms
64 bytes from 192.168.1.30: icmp_seq=1 ttl=64 time=0.197 ms (DUP!)
64 bytes from 192.168.1.30: icmp_seq=2 ttl=64 time=0.130 ms
64 bytes from 192.168.1.30: icmp_seq=2 ttl=64 time=0.180 ms (DUP!)
64 bytes from 192.168.1.30: icmp_seq=3 ttl=64 time=0.130 ms
64 bytes from 192.168.1.30: icmp_seq=3 ttl=64 time=0.180 ms (DUP!)
64 bytes from 192.168.1.30: icmp_seq=4 ttl=64 time=0.132 ms
64 bytes from 192.168.1.30: icmp_seq=4 ttl=64 time=0.182 ms (DUP!)
64 bytes from 192.168.1.30: icmp_seq=5 ttl=64 time=0.129 ms
64 bytes from 192.168.1.30: icmp_seq=5 ttl=64 time=0.181 ms (DUP!)
64 bytes from 192.168.1.30: icmp_seq=6 ttl=64 time=0.130 ms
64 bytes from 192.168.1.30: icmp_seq=6 ttl=64 time=0.178 ms (DUP!)
64 bytes from 192.168.1.30: icmp_seq=7 ttl=64 time=0.129 ms
64 bytes from 192.168.1.30: icmp_seq=7 ttl=64 time=0.177 ms (DUP!)
^C
--- 192.168.1.30 ping statistics ---
7 packets transmitted, 7 received, +7 duplicates, 0% packet loss, time 6000ms
rtt min/avg/max/mdev = 0.129/0.156/0.197/0.029 ms
netlab40:~ #
```

Figure 8: Wireshark screenshot of duplicate ICMP reply

3 Conclusion

Our code implements both the ARP from project 2 as well as ICMP. The user is able to enter in a target IP. The code uses the subnet mask to check whether the target IP is local or not. For a local IP, an ARP request is sent to receive the MAC address of that IP unless the IP is in the ARP cache. For a non-local IP, an ARP request is sent to the router unless the IP is already in the ARP cache. The code uses the ARP replies to obtain the desired MAC address. Once the MAC address is obtained an IP frame is assembled with the target IP and MAC address. An ICMP request is sent. If our PC obtains an ICMP request, a reply frame is assembled and sent. We used code from Project 2 in order to complete Project 3.

4 Appendix

```
#include "frameio.h"
#include "util.h"
#include "chksum.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <iostream>
#include <fstream>
#include <string>
#include <time.h>
```

```
#include <vector>
#include <fstream>

frameio net;              // gives us access to the raw network
message_queue ip_queue;   // message queue for the IP protocol stack
message_queue arp_queue;  // message queue for the ARP protocol stack

struct ether_frame        // handy template for 802.3/DIX frames
{
   octet dst_mac[6];      // destination MAC address
   octet src_mac[6];      // source MAC address
   octet prot[2];         // protocol (or length)
   octet data[1500];      // payload
};


class ARP_Table
{
public:
  octet ip_addr[4];
  octet mac_addr[6];
  time_t timer;

  ARP_Table(octet ip[4], octet mac[6]) {
    memcpy(ip_addr, ip, 4);
    memcpy(mac_addr, mac, 6);
    time(&timer);
    //std::cout << "Timer value: " << timer << std::endl;
    //printf("Cached IP: %d.%d.%d.%d\n", ip_addr[0], ip_addr[1], ip_addr[2], ip_addr[3]);
    //printf("Cached MAC: %x.%x.%x.%x\n", mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3]);
  };

  bool is_ip(octet ip[4]) {
    //printf("Cached IP: %d.%d.%d.%d\n", ip_addr[0], ip_addr[1], ip_addr[2], ip_addr[3]);
    //printf("Cached MAC: %x.%x.%x.%x.%x.%x\n", mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
    for (int i = 0; i < 4; i++) {
      if (ip_addr[i] != ip[i]) {
        return false;
      }
    }
    return true;
  }
};

std::vector<ARP_Table>cache_table;
octet local_addr[4];

//
// This thread sits around and receives frames from the network.
// When it gets one, it dispatches it to the proper protocol stack.
//
void *protocol_loop(void *arg)
{
   ether_frame buf;
   while(1)
   {
      int n = net.recv_frame(&buf,sizeof(buf));
      if ( n < 42 ) continue; // bad frame!
      switch ( buf.prot[0]<<8 | buf.prot[1] )
      {
          case 0x800:
             ip_queue.send(PACKET,buf.data,n);
             break;
          case 0x806:
             arp_queue.send(PACKET,buf.data,n);
             break;
      }
   }
}

//
// Toy function to print something interesting when an IP frame arrives
//
void *ip_protocol_loop(void *arg)
{
   octet buf[1500];
   event_kind event;
   int timer_no = 1;

   const octet *local_mac = net.get_mac();

   // for fun, fire a timer each time we get a frame
   while ( 1 )
   {
      ip_queue.recv(&event, buf, sizeof(buf));
      if ( event != TIMER )
      {
         //printf("got an IP frame from %d.%d.%d.%d, queued timer %d\n",
         //         buf[12],buf[13],buf[14],buf[15],timer_no);

         if (buf[9] == 1) {
           printf("Received ICMP");

           int hc = ~chksum(buf, 20, 0);
           if (hc & 0xFFFF != 0) {
             printf("Bad checksum! checksum: %x\n", hc & 0xFFFF);
             continue;
           }

           /*
           int hc = chksum(buf, 10, 0);
           hc = chksum(&buf[12], 8, hc);
           hc = ~hc;
           if ((buf[10] != (hc >> 8) & 0xFF) || (buf[11] != (hc & 0xFF))) {
             printf("Bad checksum! Should be %x, received %x%x\n", hc & 0xFFFF, buf[10], buf
    [11]);
             continue;
           }
           */

           if (buf[20] == 0x00)
            printf(" reply - sequence: %d\n", buf[27]);
           else if (buf[20] == 0x08) {
            printf(" request\n");


            ether_frame frame;

            frame.prot[0] = 0x08;
            frame.prot[1] = 0x00;

            bool found = false;
            for (int i = 0; i < cache_table.size(); i++) {
              ARP_Table target = cache_table[i];

              if (cache_table[i].is_ip(&buf[12])) {
                found = true;

                for (int i = 0; i < 6; i++)
                {
                  frame.dst_mac[i] = target.mac_addr[i];
                  frame.src_mac[i] = local_mac[i];
                }
              }
```

```
 }
 if (!found)
   continue;


 // IP Version + IHL
 frame.data[0] = 0x45;

 // Diff serivces
 frame.data[1] = 0x00;

 // Total length
 frame.data[2] = buf[2];
 frame.data[3] = buf[3];

 // Identification
 frame.data[4] = 0x00;
 frame.data[5] = 0x00;

 // Fragment
 frame.data[6] = 0x40;
 frame.data[7] = 0x00;

 // TTL
 frame.data[8] = 0x40;

 // Protocol
 frame.data[9] = 0x01;

 for (int i = 0; i < 4; i++)
 {
   // Sender's IP
   frame.data[12 + i] = buf[16 + i];
   // Target IP
   frame.data[16 + i] = buf[12 + i];
 }

 int hc = chksum(frame.data, 10, 0);
 hc = chksum(&frame.data[12], 8, hc);
 hc = ~hc;

 // Checksum
 frame.data[10] = (hc >> 8) & 0xFF;
 frame.data[11] = hc & 0xFF;

 // ICMP Type (request)
 frame.data[20] = 0x00;
 frame.data[21] = 0x00;

 int length = (buf[2] << 8) + buf[3];
 for (int i = 24; i < length; i++)
   frame.data[i] = buf[i];

 int dc = chksum(&frame.data[20], 2, 0);
 dc = chksum(&frame.data[24], 60, dc);
 dc = ~dc;

 // Checksum
 frame.data[22] = (dc >> 8) & 0xFF;
 frame.data[23] = dc & 0xFF;

 net.send_frame(&frame, length + 14);
 std::cout << "---SEND_ICMP_REPLY---" << std::endl;
}

/*
for (int i = 0; i < 20; i++) {
  printf("%d: %d - 0x%x\n", i, buf[i], buf[i]);
```

```
           }
           */
         }
         ip_queue.timer(10,timer_no);
         timer_no++;
      }
      else
      {
         //printf("timer %d fired\n",*(int *)buf);
      }
   }
}

//
// Toy function to print something interesting when an ARP frame arrives
//
void *arp_protocol_loop(void *arg)
{
   octet buf[1500];
   event_kind event;

   const octet *local_mac = net.get_mac();
   //for (int i = 0; i < 6; i++)
   //  printf("%02x ", mac[i]);


   FILE *ph = popen("ifconfig␣enp3s0␣|␣grep␣'inet␣addr'␣|␣cut␣-d':'␣-f2␣|␣cut␣-d'␣'␣-f1", "r
    ");
   char local_addr_string[15];
   fgets(local_addr_string, sizeof(local_addr_string) - 1, ph);
   local_addr_string[14] = 0;
   pclose(ph);

   char *str = local_addr_string;
   char *end = str;
   local_addr[0] = strtol(str, &end, 10);
   while (*end == '.') end++;
   str = end;
   local_addr[1] = strtol(str, &end, 10);
   while (*end == '.') end++;
   str = end;
   local_addr[2] = strtol(str, &end, 10);
   while (*end == '.') end++;
   str = end;
   local_addr[3] = strtol(str, &end, 10);
   //printf("Local IP Address: %d.%d.%d.%d\n", local_addr[0], local_addr[1], local_addr[2],
    local_addr[3]);

   //freopen("project2_output.txt", "w+", stdout);

   while ( 1 )
   {
      arp_queue.recv(&event, buf, sizeof(buf));
      //printf("got an ARP %s\n", buf[7]==1? "request":"reply");

      octet ip[4];
      octet mac[6];
      memcpy(ip, &buf[14], 4);
      memcpy(mac, &buf[8], 6);

      //std::cout << "Prev table size: " << cache_table.size() << std::endl;
      bool found = false;
      for (int i = 0; i < cache_table.size(); i++) {
        if (cache_table[i].is_ip(ip)) {
          //std::cout << "Found in table" << std::endl;
          found = true;
          break;
        }
      }
      if (!found) {
        //std::cout << "Not found in table, adding for ip: " << (int)ip[3] << ", last mac: "
    << mac[5] << std::endl;
        ARP_Table entry = ARP_Table(ip, mac);
        cache_table.push_back(entry);
      }
      //std::cout << "New table size: " << cache_table.size() << std::endl;

      if (buf[7] == 1)
      {
        printf("ARP Target IP: %d.%d.%d.%d\n", buf[24], buf[25], buf[26], buf[27]);
        printf("ARP Sender IP: %d.%d.%d.%d\n", buf[14], buf[15], buf[16], buf[17]);

        bool is_me = true;
  for (int i = 0; i < 4; i++) {
          if (buf[24 + i] != local_addr[i]) {
            is_me = false;
            break;
          }
        }

        ether_frame resp;

        if (!is_me) {
          continue;
        }
        else {
          //printf("Looking for me!\n");

          for (int i = 0; i < 6; i++)
          {
            resp.dst_mac[i] = buf[8 + i];
            resp.src_mac[i] = local_mac[i];
            // Sender's hardware address
            resp.data[8 + i] = local_mac[i];
            // Target hardware address
            resp.data[18 + i] = buf[8 + i];
          }
        }
        resp.prot[0] = 0x08;
        resp.prot[1] = 0x06;
        // hardware type (ethernet)
        resp.data[0] = 0x00;
        resp.data[1] = 0x01;
        // Protocol type (IPv4)
        resp.data[2] = 0x08;
        resp.data[3] = 0x00;
        // Hardware address length
        resp.data[4] = 0x06;
        // Protocol address length
        resp.data[5] = 0x04;
        // Opcode (2 = reply)
        resp.data[6] = 0x00;
        resp.data[7] = 0x02;

        for (int i = 0; i < 4; i++)
        {
          // Sender's IP
          resp.data[14 + i] = buf[24 + i];
          // Target IP
          resp.data[24 + i] = buf[14 + i];
        }

        net.send_frame(&resp, 42);
        //for (int i = 1; i < 60; i++)
          //printf("\t index: %d, value: 0x%x - %d\n", i, buf[i], buf[i]);

        //Is this me?
        //Find source address
        //Send response
      }
    }
}

void *cin_loop(void *arg) {
  const octet *mac = net.get_mac();

  FILE *ph = popen("ifconfig␣enp3s0␣|␣sed␣-rn␣'2s/␣.*:(.*)$/\\1/p'", "r");
  char mask_string[15];
  fgets(mask_string, sizeof(mask_string) - 1, ph);
  mask_string[14] = 0;
  pclose(ph);

  octet mask_addr[4];
  char *mask_str = mask_string;
  char *end = mask_str;
  mask_addr[0] = strtol(mask_str, &end, 10);
  while (*end == '.') end++;
  mask_str = end;
  mask_addr[1] = strtol(mask_str, &end, 10);
  while (*end == '.') end++;
  mask_str = end;
  mask_addr[2] = strtol(mask_str, &end, 10);
  while (*end == '.') end++;
  mask_str = end;
  mask_addr[3] = strtol(mask_str, &end, 10);
  std::cout << (int)mask_addr[0] << "." << (int)mask_addr[1] << "." << (int)mask_addr[2] <<
    "." << (int)mask_addr[3] << std::endl;


  std::cout << "Enter␣the␣target␣IP␣address:␣" << std::endl;
  int read = 0;
  std::string str;
  octet input[4];
  while (read < 3 && std::getline(std::cin, str, '.') || read < 4 && std::getline(std::cin,
    str)) {
    input[read] = std::stoi(str);
    read++;
  }

  //std::cin >> (int)input[0] >> (int)input[1] >> (int)input[2] >> (int)input[3];
  std::cout << (int)input[0] << "." << (int)input[1] << "." << (int)input[2] << "." << (int)
    input[3] << std::endl;
  std::cout << (int)(input[0] & mask_addr[0]) << "." << (int)(input[1] & mask_addr[1]) << ".
    " << (int)(input[2] & mask_addr[2]) << "." << (int)(input[3] & mask_addr[3]) << std::
    endl;

  bool local_network = false;
  if (((input[0] & mask_addr[0]) == (local_addr[0] & mask_addr[0])) &&
      ((input[1] & mask_addr[1]) == (local_addr[1] & mask_addr[1])) &&
      ((input[2] & mask_addr[2]) == (local_addr[2] & mask_addr[2])) &&
      ((input[3] & mask_addr[3]) == (local_addr[3] & mask_addr[3]))) {
    local_network = true;
  }
  std::cout << (local_network ? "Local" : "Not␣Local") << std::endl;

  octet gateway[4] = { 192, 168, 1, 1 };
  octet *dest_addr;
  if (local_network)
    dest_addr = input;
  else
    dest_addr = gateway;
```

```
int seq = 0;

while(1) {
  bool found_entry = false;
  for (int i = 0; i < cache_table.size(); i++) {
    ARP_Table target = cache_table[i];

    if (cache_table[i].is_ip(dest_addr)) {
      std::cout << "Found␣input␣in␣the␣table" << std::endl;
      //Send reply frame
      found_entry = true;

      ether_frame frame;

      frame.prot[0] = 0x08;
      frame.prot[1] = 0x00;

      for (int i = 0; i < 6; i++)
      {
        frame.dst_mac[i] = target.mac_addr[i];
        frame.src_mac[i] = mac[i];
        // Sender's hardware address
        // frame.data[8 + i] = mac[i];
        // Target hardware address
        // frame.data[18 + i] = target.mac_addr[i];
      }

      // IP Version + IHL
      frame.data[0] = 0x45;

      // Diff serivces
      frame.data[1] = 0x00;

      // Total length
      frame.data[2] = 0x00;
      frame.data[3] = 0x54;

      // Identification
      frame.data[4] = 0x00;
      frame.data[5] = 0x00;

      // Fragment
      frame.data[6] = 0x40;
      frame.data[7] = 0x00;

      // TTL
      frame.data[8] = 0x40;

      // Protocol
      frame.data[9] = 0x01;

      for (int i = 0; i < 4; i++)
      {
        // Sender's IP
        frame.data[12 + i] = local_addr[i];
        // Target IP
        frame.data[16 + i] = input[i];
      }

      int hc = chksum(frame.data, 10, 0);
      hc = chksum(&frame.data[12], 8, hc);
      hc = ~hc;

      // Checksum
      frame.data[10] = (hc >> 8) & 0xFF;
      frame.data[11] = hc & 0xFF;
```

```
        // ICMP Type (request)
        frame.data[20] = 0x08;
        frame.data[21] = 0x00;

        // ICMP Identifier
        frame.data[24] = 0x00;
        frame.data[25] = 0x00;

        // ICMP Sequence
        frame.data[26] = (seq >> 8) & 0xFF;
        frame.data[27] = seq & 0xFF;

        // ICMP Data
        for (int i = 28; i < 84; i++)
          frame.data[i] = i - 28;

        int dc = chksum(&frame.data[20], 2, 0);
        dc = chksum(&frame.data[24], 60, dc);
        dc = ~dc;

        // Checksum
        frame.data[22] = (dc >> 8) & 0xFF;
        frame.data[23] = dc & 0xFF;

        net.send_frame(&frame, 98);
        std::cout << "--SEND ICMP Request, sequence: " << seq << "--" << std::endl;

        seq++;
      }
    }

    if (!found_entry)
    {
      std::cout << "Did not find input in the table, request address" << std::endl;

      ether_frame resp;

      for (int i = 0; i < 6; i++)
      {
        resp.dst_mac[i] = 0xFF;
        resp.src_mac[i] = mac[i];
        // Sender's hardware address
        resp.data[8 + i] = mac[i];
        // Target hardware address
        resp.data[18 + i] = 0;
      }

      //Send request frame
      // Opcode (1 = request)
      resp.data[6] = 0x00;
      resp.data[7] = 0x01;


      resp.prot[0] = 0x08;
      resp.prot[1] = 0x06;
      // hardware type (ethernet)
      resp.data[0] = 0x00;
      resp.data[1] = 0x01;
      // Protocol type (IPv4)
      resp.data[2] = 0x08;
      resp.data[3] = 0x00;
      // Hardware address length
      resp.data[4] = 0x06;
      // Protocol address length
      resp.data[5] = 0x04;

      for (int i = 0; i < 4; i++)
      {
        // Sender's IP
        resp.data[14 + i] = local_addr[i];
        // Target IP
        resp.data[24 + i] = dest_addr[i];
      }

      net.send_frame(&resp, 42);
      std::cout << "---SEND FRAME---" << std::endl;
    }

    sleep(1);
  }
}

void *time_loop(void *arg)
{
  while(1)
  {
    sleep(1);

    time_t timer;
    time(&timer);
    int i = 0;
    while (i < cache_table.size()) {
      if (timer - cache_table[i].timer > 20) {
        std::cout << "Timer removed an item from the cache table" << std::endl;
        std::cout << "Removed item with IP: " << (int)cache_table[i].ip_addr[0] << "." << (
    int)cache_table[i].ip_addr[1] << "." << (int)cache_table[i].ip_addr[2] << "." << (int)
    cache_table[i].ip_addr[3] << std::endl;

        cache_table.erase(cache_table.begin() + i);
      }
      else {
        i++;
      }
    }
  }
}

//
// if you're going to have pthreads, you'll need some thread descriptors
//
pthread_t loop_thread, cin_thread, arp_thread, ip_thread, timer_thread;

//
// start all the threads then step back and watch (actually, the timer
// thread will be started later, but that is invisible to us.)
//
int main()
{
   net.open_net("enp3s0");
   pthread_create(&loop_thread,NULL,protocol_loop,NULL);
   pthread_create(&arp_thread,NULL,arp_protocol_loop,NULL);
   pthread_create(&cin_thread,NULL,cin_loop,NULL);
   pthread_create(&ip_thread,NULL,ip_protocol_loop,NULL);
   pthread_create(&timer_thread,NULL,time_loop,NULL);
   for ( ; ; )
      sleep(1);
}
```