Project – Part 3

1. Objective

1. Be familiar with the structure of IPv4 frame.
2. Understand and implement a send packet function via IP frame.
3. Know ICMP message types and implement ICMP echo.
4. Know the concepts of fragmentation and de-fragmentation.

2. Overview

Internet Protocol (IP) and Internet Message Control Protocol (ICMP) belong to the network layer. IP basically provides 2 services: send a packet and receive a packet. Unfortunately, it is difficult to test IP in isolation, so we will be implementing just enough of ICMP to send and respond to a "ping". Fortunately, ping packets are (or can be) small enough to transmit on Ethernet without fragmentation, so you can defer the whole fragment/de-fragment problem for now.

3. Results

- Destination MAC Address: 00:1A:A0:AC:AF:6B
- Destination IP Address: 192.168.1.20
- Source MAC Address: 00:1A:A0:AC:B1:0A
- Source IP Address: 192.168.1.10

3.1 Procedure 1

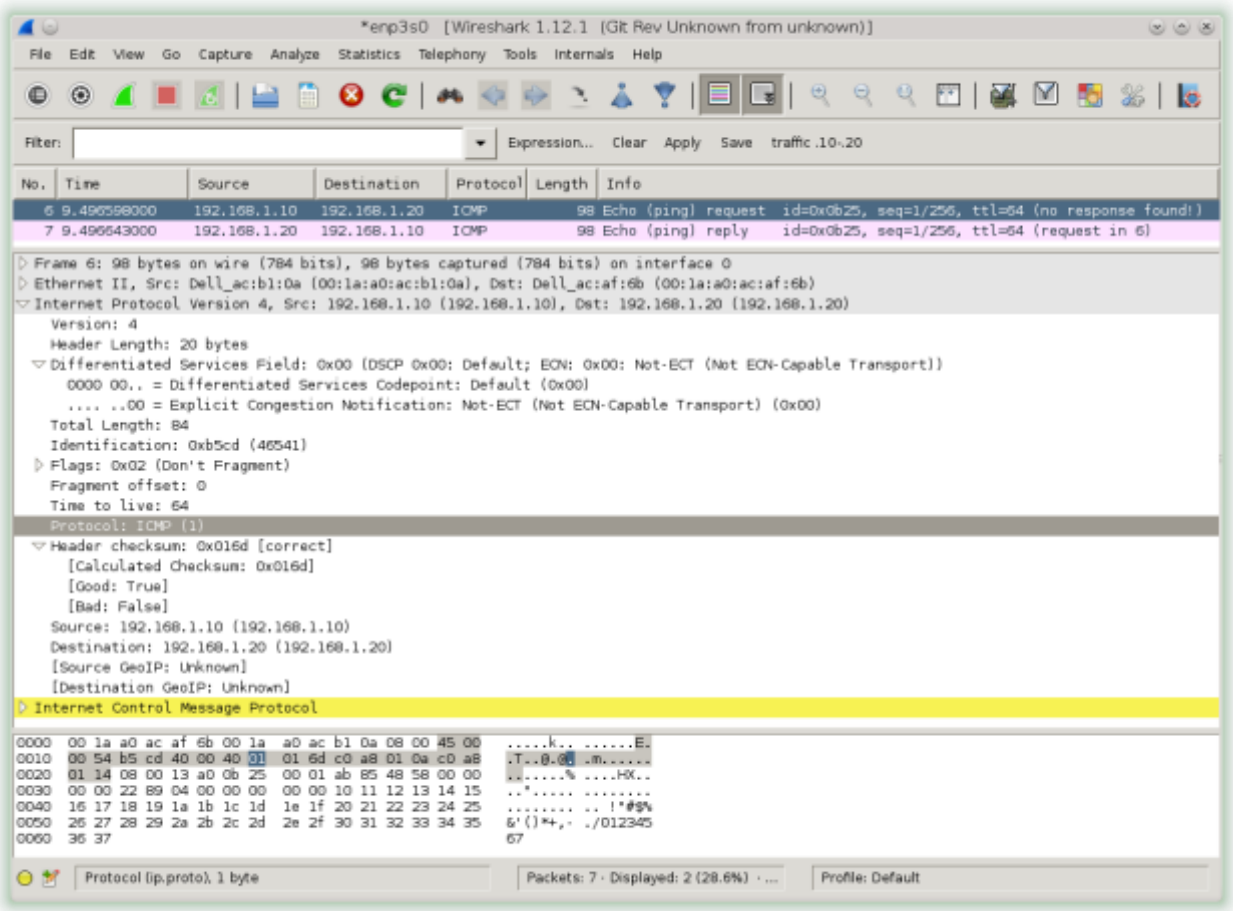
- Write code to identify if a received frame is type ICMP.
  - If the frame is ICMP, print to terminal.
  - If the received checksum equals the code's calculated checksum, print passed to terminal.

Program Output (see code in Appendix 4.1)

```
netlab20:~/Documents/jmeine # ./out
ICMP frame to my IP address received.

IP header checksum received is 01 6d.

IP header checksum calculated is 01 6d.

IP header checksum passed.
```

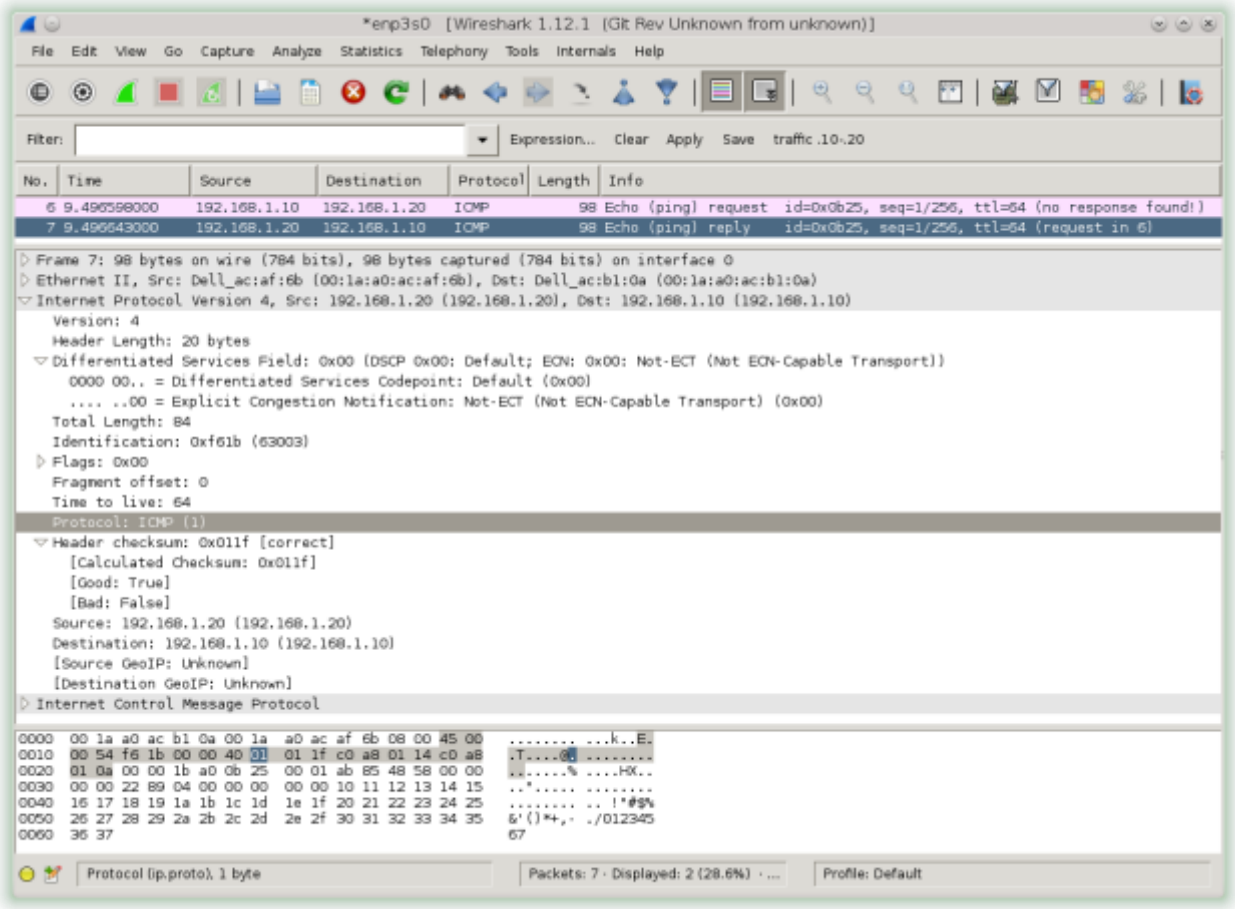
Network Capture

ICMP Request from PC
Sender's MAC Address = 00:1A:A0:AC:B1:0A
Sender's IP Address = 192.168.1.10
Target MAC Address = 00:1A:A0:AC:AF:6B
Target IP Address = 192.168.1.20



- The checksum 0x016d matches value received in the program output.

ICMP Reply from PC
Sender's MAC Address = 00:1A:A0:AC:AF:6B
Sender's IP Address = 192.168.1.20
Target MAC Address = 00:1A:A0:AC:B1:0A
Target IP Address = 192.168.1.10



3.2 Procedure 2

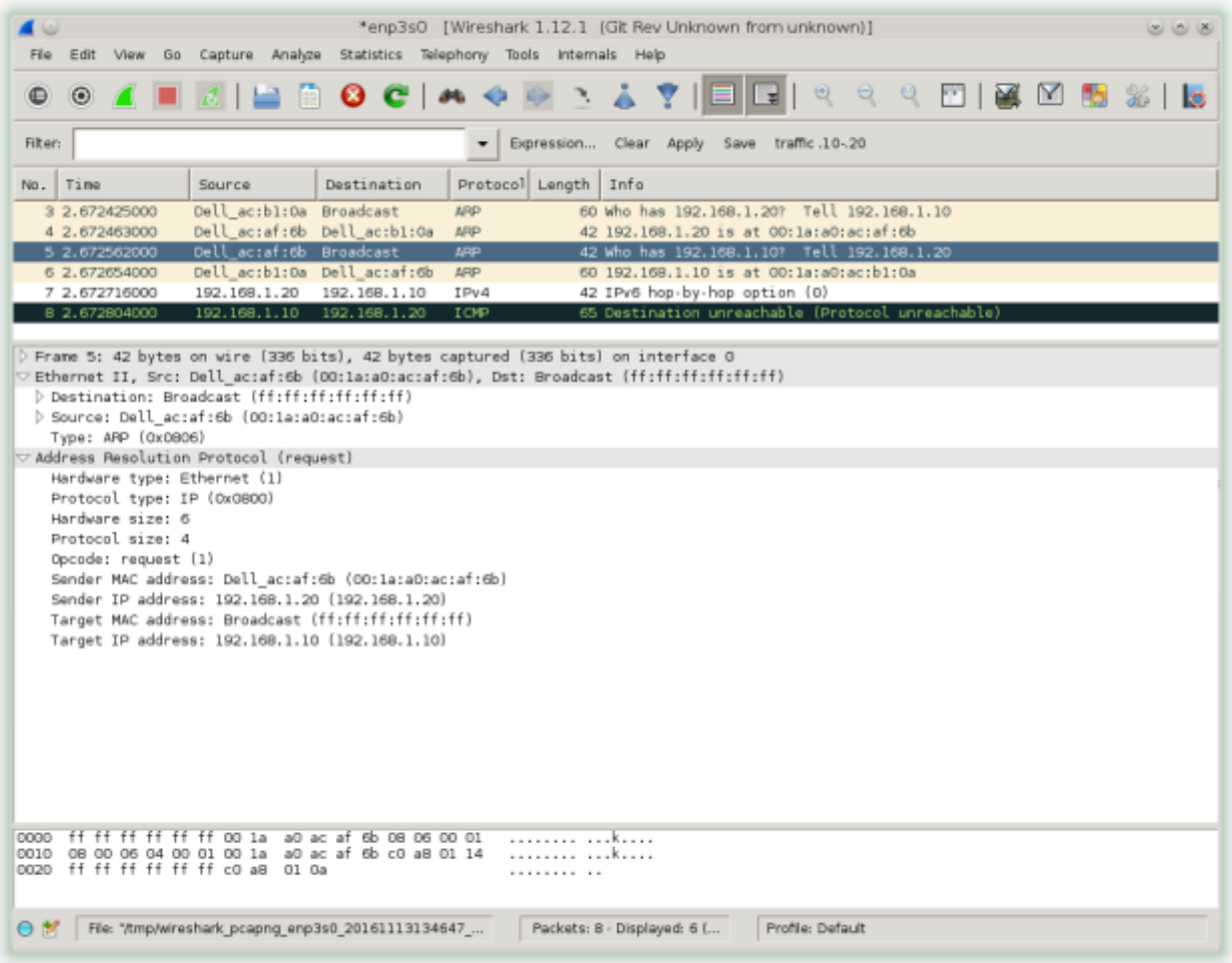
- Write code to send an IP frame to a specified target IP address.
  - If the target IP address is in the local network, send ARP request to target directly (3.2.1).
  - Otherwise, send ARP request to the router (3.2.2).
  - When ARP reply is received to sender's machine (3.2.3), send IP frame to target IP address (3.2.4).

Program Output – Target IP in local network (see code in Appendix 4.2)

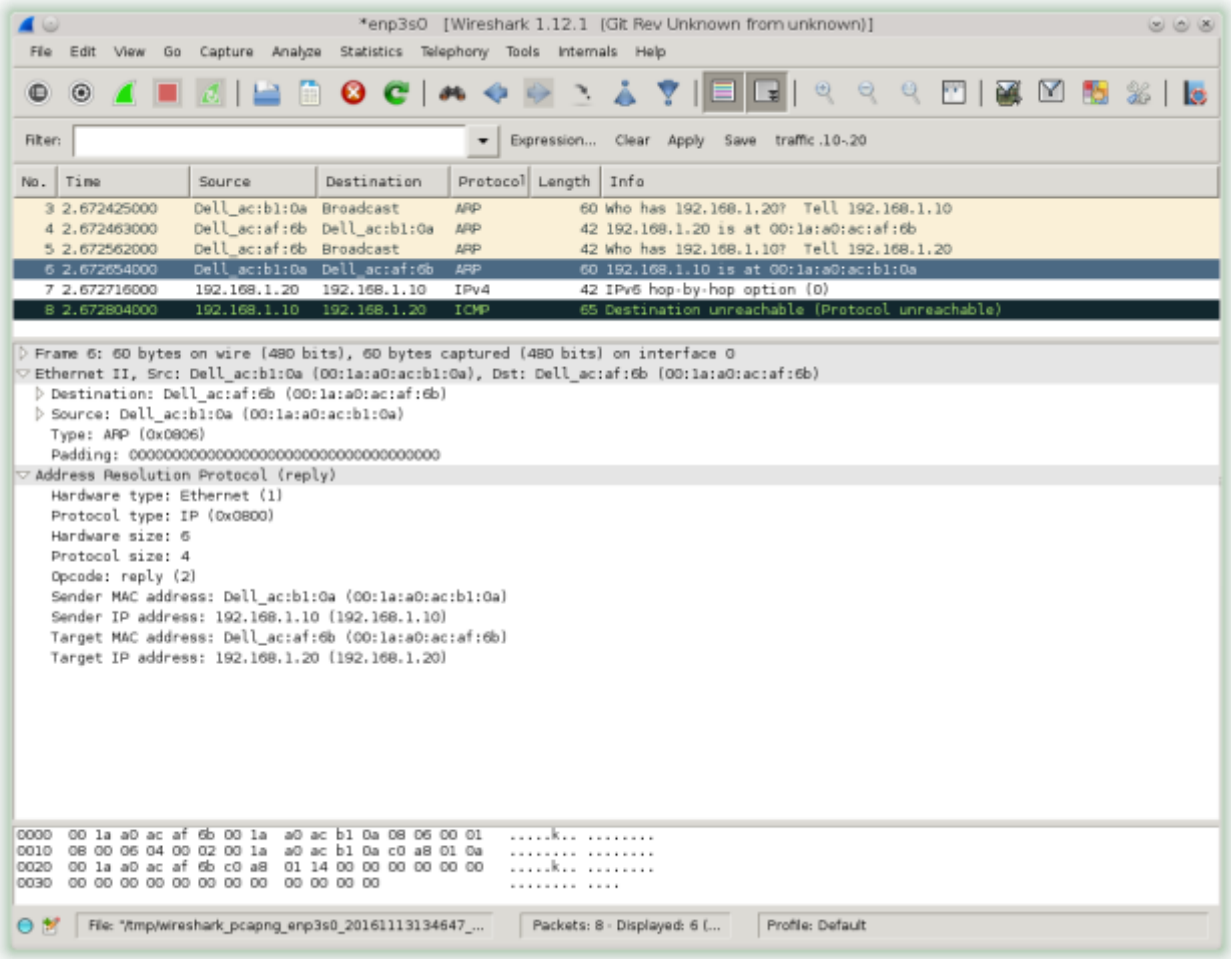
```
Target IP Address = 192.168.1.10

ARP request detected.

Sending ARP request...

Target IP is on local network.

Sender MAC Address = 00:1a:a0:ac:af:6b
Sender IP Address = 192.168.1.20
Target MAC Address = ff:ff:ff:ff:ff:ff
Target IP Address = 192.168.1.10

ARP request has been sent.

ARP reply detected.

ARP reply to my IP address received.

Target IP Address = 192.168.1.10
Target MAC Address = 00:1a:a0:ac:b1:0a

Sending IP frame...

Source MAC Address = 00:1a:a0:ac:af:6b
Source IP Address = 192.168.1.20
Destination MAC Address = 00:1a:a0:ac:b1:0a
Destination IP Address = 192.168.1.10

IP Data = ab:cd:ef
IP frame has been sent. END
```

Network Capture – Target IP in local network

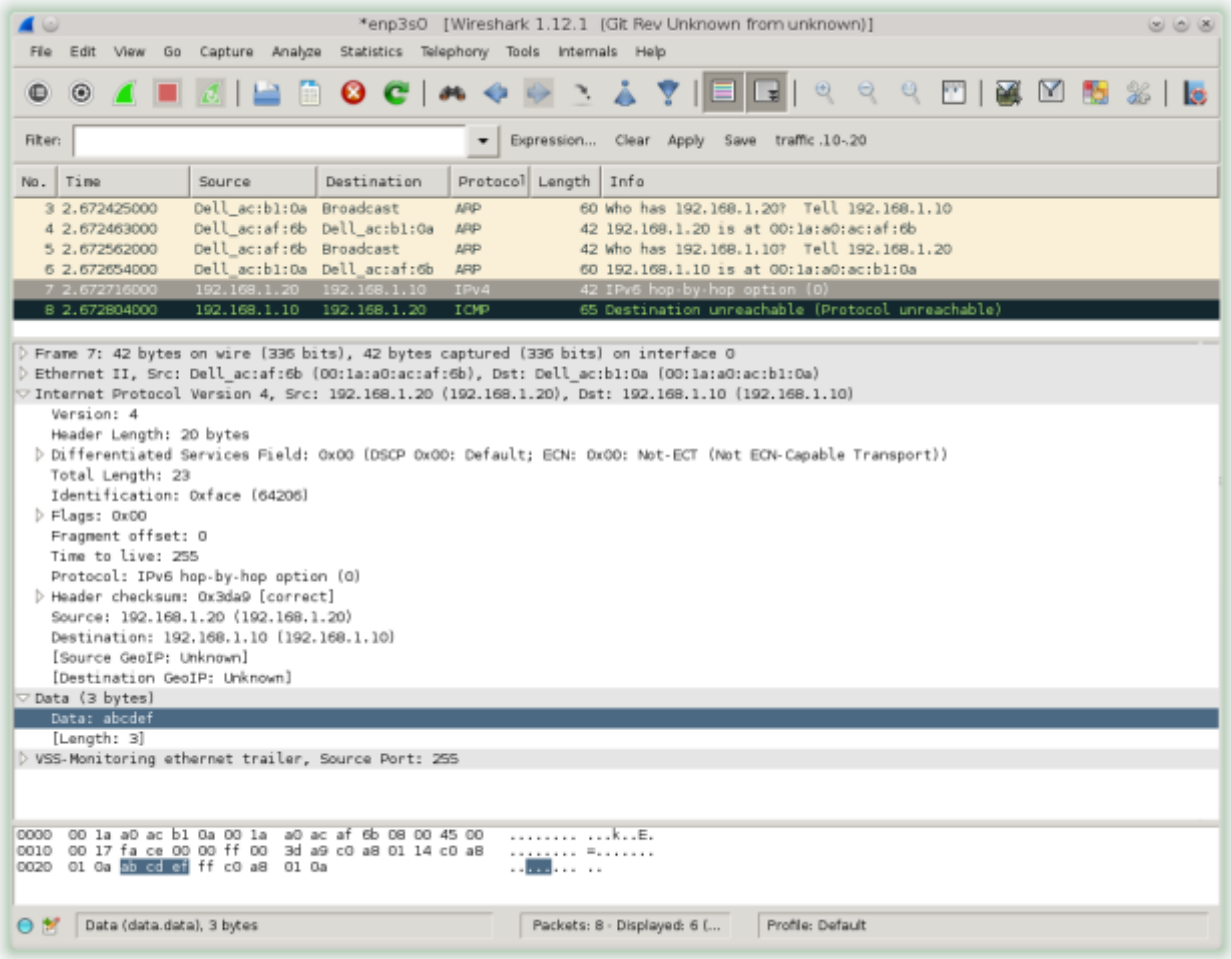
ARP Request from Code (3.2.1)
Sender's MAC Address = 00:1A:A0:AC:AF:6B
Sender's IP Address = 192.168.1.20
Target MAC Address = FF:FF:FF:FF:FF:FF
Target IP Address = 192.168.1.10



ARP Reply from PC (3.2.3)
Sender's MAC Address = 00:1A:A0:AC:B1:A0
Sender's IP Address = 192.168.1.10
Target MAC Address = 00:1A:A0:AC:AF:6B
Target IP Address = 192.168.1.20



IP Frame from Code (3.2.4)
Sender's MAC Address = 00:1A:A0:AC:AF:6B
Sender's IP Address = 192.168.1.20
Target MAC Address = 00:1A:A0:AC:B1:A0
Target IP Address = 192.168.1.10



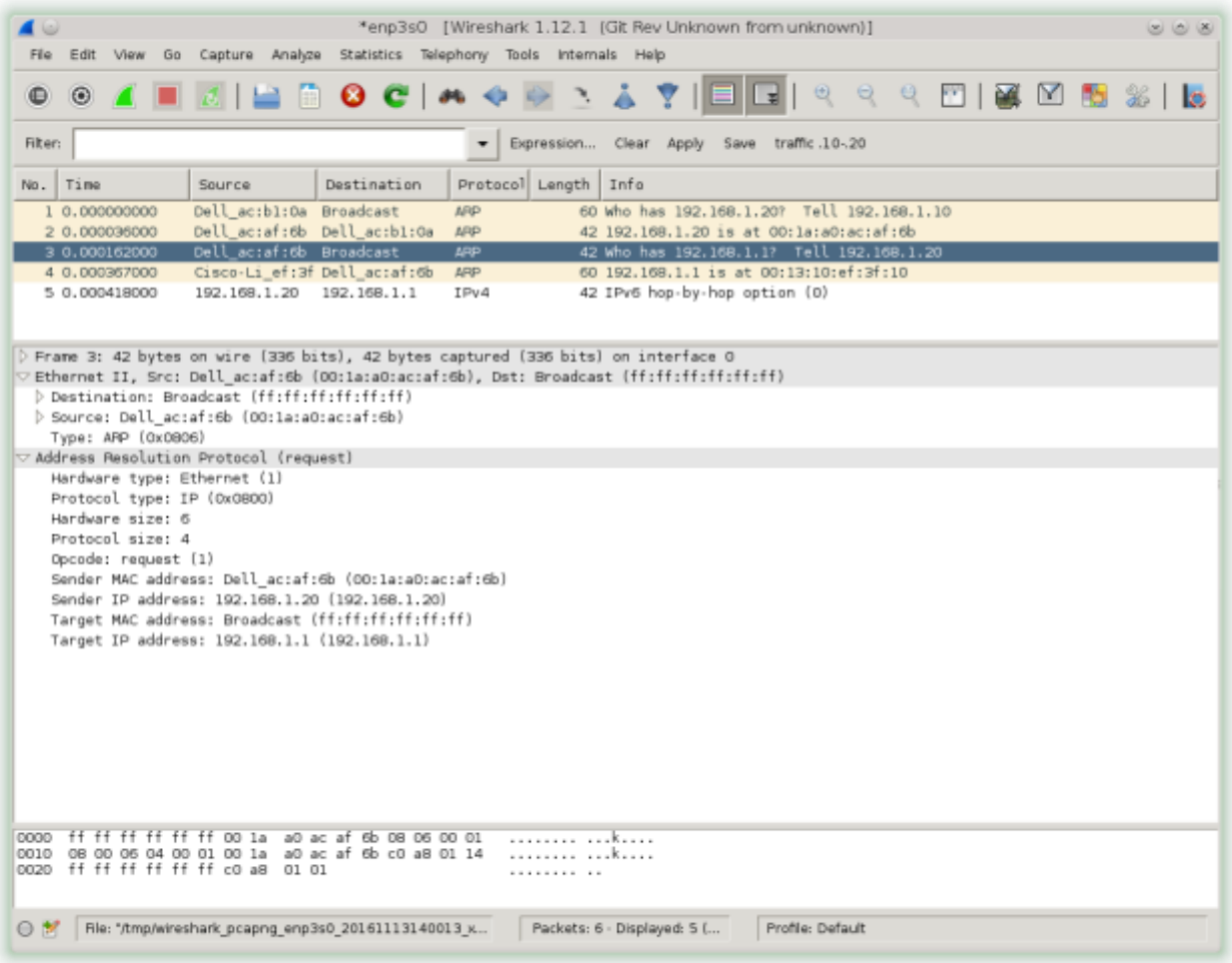
- In code, the packet length set is 42 bytes with data set to 0xabcdef.

Program Output – Target IP in remote network (see code in Appendix 4.2)

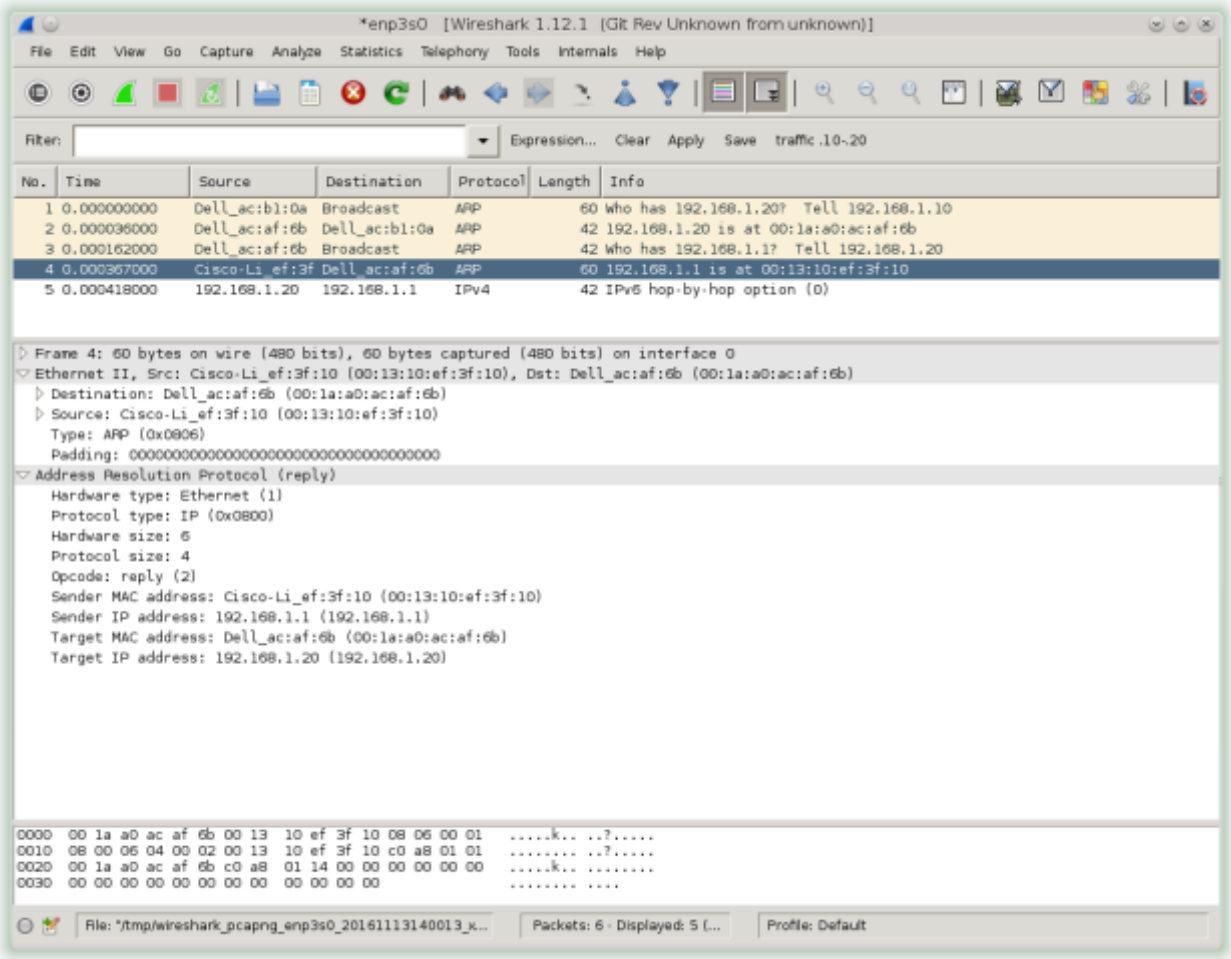
```
Target IP Address = 172.217.1.206

ARP request detected.

Sending ARP request...

Target IP is on remote network.

Sender MAC Address = 00:1a:a0:ac:af:6b
Sender IP Address = 192.168.1.20
Target MAC Address = ff:ff:ff:ff:ff:ff
Target IP Address = 192.168.1.1

ARP request has been sent.

ARP reply detected.

ARP reply to my IP address received.

Target IP Address = 192.168.1.1
Target MAC Address = 00:13:10:ef:3f:10

Sending IP frame...

Source MAC Address = 00:1a:a0:ac:af:6b
Source IP Address = 192.168.1.20
Destination MAC Address = 00:13:10:ef:3f:10
Destination IP Address = 192.168.1.1

IP Data = ab:cd:ef
IP frame has been sent. END
```

Network Capture – Target IP in remote network

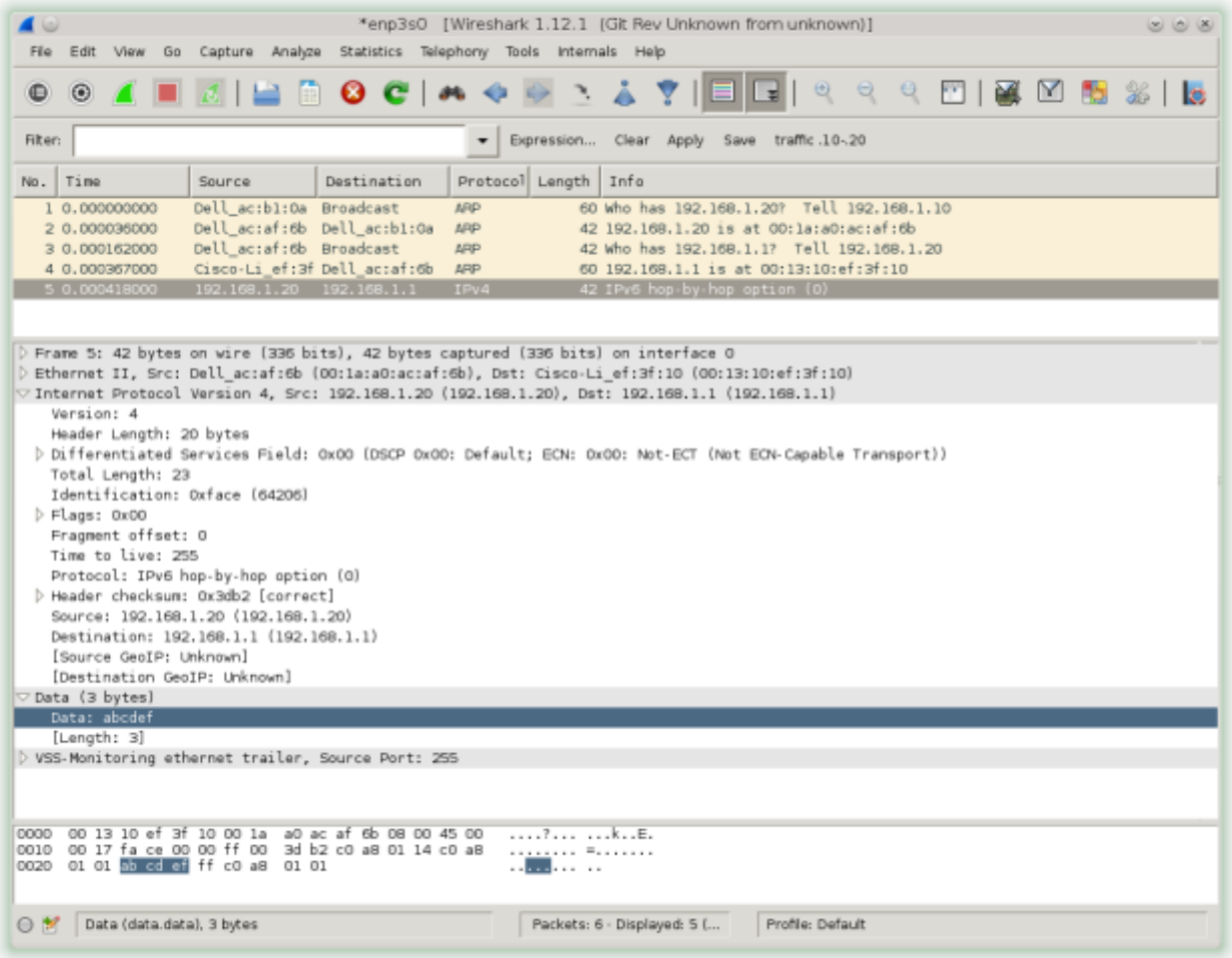
ARP Request from Code (3.2.2)
Sender's MAC Address = 00:1A:A0:AC:AF:6B
Sender's IP Address = 192.168.1.20
Target MAC Address = FF:FF:FF:FF:FF:FF
Target IP Address = 192.168.1.1



ARP Reply from PC (3.2.3)
Sender's MAC Address = 00:13:10:EF:3F:10
Sender's IP Address = 192.168.1.1
Target MAC Address = 00:1A:A0:AC:AF:6B
Target IP Address = 192.168.1.20



IP Frame from Code (3.2.4)
Sender's MAC Address = 00:1A:A0:AC:AF:6B
Sender's IP Address = 192.168.1.20
Target MAC Address = 00:13:10:EF:3F:10
Target IP Address = 192.168.1.1



- In code, the packet length set is 42 bytes with data set to 0xabcdef.

3.3 Procedure 3

- Write code to send an IP frame to a specified target IP address.
  - If the target IP address is in the local network, send ARP request to target directly.
  - Otherwise, send ARP request to the router.
  - When ARP reply is received to sender's machine, send IP frame with ICMP request to target IP address.
  - When ICMP reply is received to sender's machine, send another IP frame with ICMP request to target IP address.
  - Continue to capture the ICMP request and reply pairs.

- Destination MAC Address: 00:1A:A0:AC:B0:E8
- Destination IP Address: 192.168.1.20
- Source MAC Address: 00:1A:A0:AC:DF:57
- Source IP Address: 192.168.1.10

Program Output – Send ICMP Request from ARP Reply to Local IP (see code in Appendix 4.3.1)

```
Target IP Address = 192.168.1.20

ARP request detected.

Sending ARP request...

Target IP is on local network.

Sender MAC Address = 00:1a:a0:ac:df:57
Sender IP Address = 192.168.1.10
Target MAC Address = ff:ff:ff:ff:ff:ff
Target IP Address = 192.168.1.20

ARP request has been sent.

ARP reply detected.

ARP reply to my IP address received.

Target IP Address = 192.168.1.20
Target MAC Address = 00:1a:a0:ac:b0:e8

Sending ICMP request...

Sender MAC Address = 00:1a:a0:ac:df:57
Sender IP Address = 192.168.1.10
Target MAC Address = 00:1a:a0:ac:b0:e8
Target IP Address = 192.168.1.20

Type = 8 (Request)
Identifier = ac ed
Sequence No. = 00 01

ICMP Data = ab:cd:ef
```

ICMP request has been sent. END

Program Output (see code in Appendix 4.3.2)

Send ICMP Reply to ICMP Request
FROM 192.168.1.20 TO 192.168.1.10

ICMP frame detected.

ICMP frame received.

ICMP request has been received. Sending ICMP reply...

Source MAC Address = 00:1a:a0:ac:b0:e8
Source IP Address = 192.168.1.20
Destination MAC Address = 00:1a:a0:ac:df:57
Destination IP Address = 192.168.1.10

Type = 0 (Reply)
Identifier = ac ed
Sequence No. = 00 01

ICMP Data = fe:dc:ba

ICMP reply has been sent. → **Go to 1**

ICMP frame detected. ← **2**

ICMP frame received.

ICMP request has been received. Sending ICMP reply...

Source MAC Address = 00:1a:a0:ac:b0:e8
Source IP Address = 192.168.1.20
Destination MAC Address = 00:1a:a0:ac:df:57
Destination IP Address = 192.168.1.10

Type = 0 (Reply)
Identifier = ac ed
Sequence No. = 00 02

ICMP Data = fe:dc:ba

ICMP reply has been sent. → **Go to 3**

ICMP frame detected. ← **4**

ICMP frame received.

ICMP request has been received. Sending ICMP reply...

Source MAC Address = 00:1a:a0:ac:b0:e8
Source IP Address = 192.168.1.20
Destination MAC Address = 00:1a:a0:ac:df:57
Destination IP Address = 192.168.1.10

Send ICMP Request to ICMP Reply
FROM 192.168.1.10 TO 192.168.1.20

1 → ICMP frame detected.

ICMP frame received.

ICMP reply has been received. Sending ICMP request...

Source MAC Address = 00:1a:a0:ac:df:57
Source IP Address = 192.168.1.10
Destination MAC Address = 00:1a:a0:ac:b0:e8
Destination IP Address = 192.168.1.20

Type = 8 (Request)
Identifier = ac ed
Sequence No. = 00 02

ICMP Data = ab:cd:ef

Go to 2 ← ICMP request has been sent.

3 → ICMP frame detected.

ICMP frame received.

ICMP reply has been received. Sending ICMP request...

Source MAC Address = 00:1a:a0:ac:df:57
Source IP Address = 192.168.1.10
Destination MAC Address = 00:1a:a0:ac:b0:e8
Destination IP Address = 192.168.1.20

Type = 8 (Request)
Identifier = ac ed
Sequence No. = 00 03

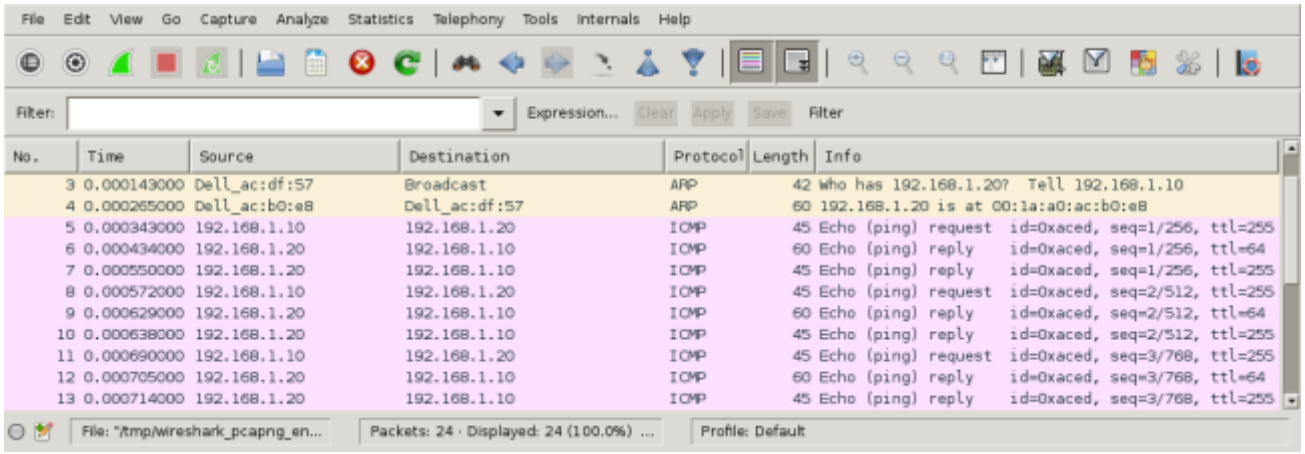
ICMP Data = ab:cd:ef

Go to 4 ← ICMP request has been sent.

END

Type = 0 (Reply) Identifier = ac ed Sequence No. = 00 03 ICMP Data = fe dc ba ICMP reply has been sent. END	

Sender's MAC Address = 00:1A:A0:AC:DF:57
Sender's IP Address = 192.168.1.10
Target MAC Address = 00:1A:A0:AC:B0:E8
Target IP Address = 192.168.1.20



- In code, the packet length set is 45 bytes, identifier is 0xaced, and ttl is 0xff or 255.
- The reply packets from target code are identified by packet length of 45 bytes and ttl of 255.
- The reply packets from target PC are identified by packet length of 60 bytes and ttl of 64.
- In screen capture, the sequence number is incremented with each request packet.
- Each request packet received one reply packet from target code and another reply packet from target PC.

```
netlab10:~/Documents/jmeine # ping -c 3 192.168.1.20
PING 192.168.1.20 (192.168.1.20) 56(84) bytes of data.
64 bytes from 192.168.1.20: icmp_seq=1 ttl=64 time=0.517 ms
64 bytes from 192.168.1.20: icmp_seq=1 ttl=255 time=0.174 ms (DUP!)
64 bytes from 192.168.1.20: icmp_seq=2 ttl=64 time=0.439 ms
64 bytes from 192.168.1.20: icmp_seq=2 ttl=255 time=0.266 ms (DUP!)
64 bytes from 192.168.1.20: icmp_seq=3 ttl=64 time=0.290 ms
64 bytes from 192.168.1.20: icmp_seq=3 ttl=255 time=0.084 ms (DUP!)

--- 192.168.1.20 ping statistics ---
3 packets transmitted, 3 received, +3 duplicates, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.084/0.295/0.517/0.147 ms
```

Program Output – Send ICMP Request from ARP Reply to Remote IP (see code in Appendix 4.3.1)

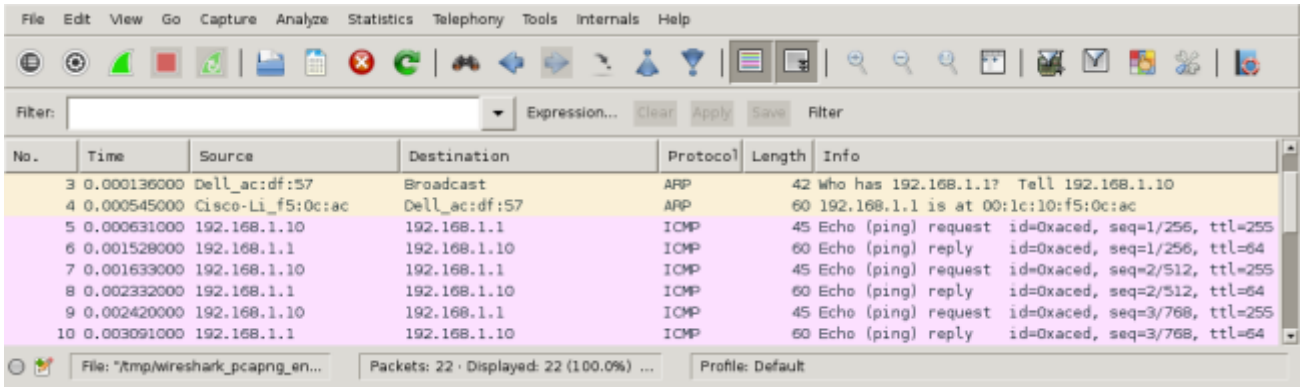
```
Target IP Address = 172.217.1.206

ARP request detected.

Sending ARP request...

Target IP is on remote network.

Sender MAC Address = 00:1a:a0:ac:df:57
Sender IP Address = 192.168.1.10
Target MAC Address = ff:ff:ff:ff:ff:ff
Target IP Address = 192.168.1.1

ARP request has been sent.

ARP reply detected.

ARP reply to my IP address received.

Target IP Address = 192.168.1.1
Target MAC Address = 00:1c:10:f5:0c:ac

Sending ICMP request...

Sender MAC Address = 00:1a:a0:ac:df:57
Sender IP Address = 192.168.1.10
Target MAC Address = 00:1c:10:f5:0c:ac
Target IP Address = 192.168.1.20

Type = 8 (Request)
Identifier = ac ed
Sequence No. = 00 01

ICMP Data = ab:cd:ef

ICMP request has been sent. END
```

Program Output – Send ICMP Request to ICMP Reply (see code in Appendix 4.3.2)

```
ICMP frame detected.

ICMP frame received.

ICMP frame detected.

ICMP frame received.

ICMP reply has been received. Sending ICMP request...

Source MAC Address = 00:1a:a0:ac:df:57
Source IP Address = 192.168.1.10
Destination MAC Address = 00:1c:10:f5:0c:ac
Destination IP Address = 192.168.1.1
```

```
Type = 8 (Request)
Identifier = ac ed
Sequence No. = 00 02

ICMP Data = ab:cd:ef

ICMP request has been sent.

ICMP frame detected.

ICMP frame received.

ICMP reply has been received. Sending ICMP request...

Source MAC Address = 00:1a:a0:ac:df:57
Source IP Address = 192.168.1.10
Destination MAC Address = 00:1c:10:f5:0c:ac
Destination IP Address = 192.168.1.1

Type = 8 (Request)
Identifier = ac ed
Sequence No. = 00 03

ICMP Data = ab:cd:ef

ICMP request has been sent.

END
```

Sender's MAC Address = 00:1A:A0:AC:DF:57
Sender's IP Address = 192.168.1.10
Target MAC Address = 00:1C:10:F5:0C:AC
Target IP Address = 192.168.1.1



- In code, the packet length set is 45 bytes, identifier is 0xaced, and ttl is 0xff or 255.
- The request packets from sender's code are identified by packet length of 45 bytes and ttl of 255.
- The reply packets from target router are identified by packet length of 60 bytes and ttl of 64.
- In screen capture, the sequence number is incremented with each request packet.
- Each request packet received one reply packet from target router.

4. Appendix

4.1 Program Code – Procedure 1

```
#include "frameio.h"
#include "util.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

frameio net;             // gives us access to the raw network
message_queue ip_queue;  // message queue for the IP 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
};

//
// 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;
      }
   }
}

int chksum(octet *s, int bytes, int initial)
{
   long sum = initial;
   int i;
   for ( i=0; i<bytes-1; i+=2 )
   {
      sum += s[i]*256 + s[i+1];
   }
   //
   // handle the odd byte
   //
   if ( i < bytes ) sum += s[i]*256;
```

```
   //
   // wrap carries back into sum
   //
   while ( sum > 0xffff ) sum = (sum & 0xffff) + (sum >> 16);
   return sum;
}

//
// Toy function to print something interesting when an IP frame arrives
//
void *ip_protocol_loop(void *arg)
{
   octet buf[1500];
   event_kind event;
   octet chksum_in[2];
   octet chksum_out[2];
   bool chksum_pass;
   octet my_ip[4] = { 192, 168, 1, 20 };
   int ip_header_bytes = 20;

   /* buf_key
   00_0:3 = Version (0b0100_=IPv4 0b0110=IPv6)
   00_4:7 = Internet Header Length
   01 = Differentiated Services
   02 to 03 = Total Length
   04 to 05 = Identification
   06 to 07 = Fragment Offset
   08 = Time to Live
   09 = Protocol (0x01=ICMP)
   10 to 11 = Header Checksum
   12 to 15 = Source IP Address
   16 to 19 = Destination IP Address
   20 to .. = Data
   */

   while ( 1 )
   {
      ip_queue.recv(&event, buf, sizeof(buf));
      for ( int ip_byte = 0; ip_byte < 42; ip_byte++) /* Read first 42 IP bytes */
      {
              // Is destination IP address my IP address?
              // Is frame received of type ICMP?
              if ( buf[16] == my_ip[0] &&
                     buf[17] == my_ip[1] &&
                     buf[18] == my_ip[2] &&
                     buf[19] == my_ip[3] &&
                     buf[9] == 1)
              {
                     printf("ICMP frame to my IP address received.\n\n");
                     chksum_in[0] = buf[10]; chksum_in[1] = buf[11]; // Save incoming IP frame
checksum
                     printf("IP header checksum received is %02x
%02x.\n\n",chksum_in[0],chksum_in[1]);
                     buf[10] = 0; buf[11] = 0; // Clear checksum of IP frame received
                     octet ip_header[ip_header_bytes];
                     for (int i = 0; i < ip_header_bytes; i++)
                            ip_header[i] = buf[i];
```

```
                    int sum = chksum((octet *)ip_header,ip_header_bytes,0);
                    chksum_out[0] = ~sum >> 8;
                    chksum_out[1] = ~sum & 0xFF;
                    printf("IP header checksum calculated is %02x
%02x.\n\n",chksum_out[0],chksum_out[1]);
                    // Is the IP header checksum correct?
                    if ( chksum_in[0] == chksum_out[0] && chksum_in[1] == chksum_out[1] )
                    {
                           printf("IP header checksum passed.\n\n");
                           chksum_pass = true;
                    }
                    else
                    {
                           printf("IP header checksum failed.\n\n");
                           chksum_pass = false;
                    }
                    goto finish;
             }
      }
   }
finish:;
}

//
// if you're going to have pthreads, you'll need some thread descriptors
//
pthread_t loop_thread, ip_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(&ip_thread,NULL,ip_protocol_loop,NULL);
   for ( ; ; )
      sleep(1);
}
```

4.2 Program Code – Procedure 2

```
#include "frameio.h"
#include "util.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

frameio net;             // gives us access to the raw network
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
};

octet my_ip[4] = { 192, 168, 1, 20 };
octet my_mac[6];
// ip_target = local ip or remote ip (choose one)
octet ip_target[4] = { 192, 168, 1, 10 }; // local ip
// octet ip_target[4] = { 172, 217, 1, 206 }; // remote ip
octet mac_target[6];
octet mac_broadcast[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
octet subnet_mask[4] = { 255, 255, 255, 0 };
octet gateway_ip[4] = { 192, 168, 1, 1 };
octet network_ip[4];

// returns true if target IP is on the local network
bool ip_result;
void in_network()
{
       network_ip[0] = my_ip[0] & subnet_mask[0];
       network_ip[1] = my_ip[1] & subnet_mask[1];
       network_ip[2] = my_ip[2] & subnet_mask[2];
       network_ip[3] = my_ip[3] & subnet_mask[3];
       if ( ip_target[0] == network_ip[0] &&
               ip_target[1] == network_ip[1] &&
               ip_target[2] == network_ip[2] )
       {
              printf("Target IP is on local network.\n\n");
              ip_result = true;
       }
       else
       {
              printf("Target IP is on remote network.\n\n");
              ip_result = false;
       }
}

ether_frame frame;
```

```
octet IP_type[2] = { 0x08, 0x00 };
octet ARP_type[2] = { 0x08, 0x06 };

void ethernet_frame(octet dst_mac[6], octet src_mac[6], octet ether_type[2])
{
	// Destination MAC Address
	frame.dst_mac[0] = dst_mac[0];
	frame.dst_mac[1] = dst_mac[1];
	frame.dst_mac[2] = dst_mac[2];
	frame.dst_mac[3] = dst_mac[3];
	frame.dst_mac[4] = dst_mac[4];
	frame.dst_mac[5] = dst_mac[5];
	// Source MAC Address
	frame.src_mac[0] = src_mac[0];
	frame.src_mac[1] = src_mac[1];
	frame.src_mac[2] = src_mac[2];
	frame.src_mac[3] = src_mac[3];
	frame.src_mac[4] = src_mac[4];
	frame.src_mac[5] = src_mac[5];
	// Ethernet Type = 0x0800 for IP, 0x0806 for ARP
	frame.prot[0] = ether_type[0];
	frame.prot[1] = ether_type[1];
}

octet opcode[2];
octet sender_mac[6];
octet sender_ip[4];
octet target_mac[6];
octet target_ip[4];

void arp_frame()
{
	// Hardware Type = 0x0001 for Ethernet
	frame.data[0] = 0x00;
	frame.data[1] = 0x01;
	// Protocol Type = 0x0800 for IPv4, 0x86DD for IPv6
	frame.data[2] = 0x08;
	frame.data[3] = 0x00;
	// Hardware Size = 6 for Ethernet
	frame.data[4] = 6;
	// Protocol Size = 4 for IPv4, 16 for IPv6
	frame.data[5] = 4;
	// Opcode = 1 for Request, 2 for Reply
	frame.data[6] = opcode[0];
	frame.data[7] = opcode[1];
	// Sender's MAC Address
	frame.data[8] = sender_mac[0];
	frame.data[9] = sender_mac[1];
	frame.data[10] = sender_mac[2];
	frame.data[11] = sender_mac[3];
	frame.data[12] = sender_mac[4];
	frame.data[13] = sender_mac[5];
	// Sender's IP Address
	frame.data[14] = sender_ip[0];
	frame.data[15] = sender_ip[1];
	frame.data[16] = sender_ip[2];
	frame.data[17] = sender_ip[3];
```

```
        // Target MAC Address
        frame.data[18] = target_mac[0];
        frame.data[19] = target_mac[1];
        frame.data[20] = target_mac[2];
        frame.data[21] = target_mac[3];
        frame.data[22] = target_mac[4];
        frame.data[23] = target_mac[5];
        // Target IP Address
        frame.data[24] = target_ip[0];
        frame.data[25] = target_ip[1];
        frame.data[26] = target_ip[2];
        frame.data[27] = target_ip[3];
}

int chksum(octet *s, int bytes, int initial)
{
   long sum = initial;
   int i;
   for ( i=0; i<bytes-1; i+=2 )
   {
      sum += s[i]*256 + s[i+1];
   }
   //
   // handle the odd byte
   //
   if ( i < bytes ) sum += s[i]*256;
   //
   // wrap carries back into sum
   //
   while ( sum > 0xffff ) sum = (sum & 0xffff) + (sum >> 16);
   return sum;
}

int ip_header_bytes = 20;
void chksum_ip_header()
{
        octet ip_header[ip_header_bytes];
        for (int i = 0; i < ip_header_bytes; i++)
                ip_header[i] = frame.data[i];
        int sum = chksum((octet *)ip_header,ip_header_bytes,0);
        frame.data[10] = ~sum >> 8;
        frame.data[11] = ~sum & 0xFF;
}

octet src_ip[4];
octet dst_ip[4];
bool ip_data;
int ip_data_bytes;

void ip_frame()
{
        // IP Version = 0b0100**** for IPv4, 0b0110**** for IPv6
        // IP Header Length = 0b****0101 for no Options
        frame.data[0] = 0x45;
        // Type of Service
        frame.data[1] = 0x00;
        // Total Length = IP Header Length (20 bytes) + Data
```

```
        frame.data[2] = 0x00;
        frame.data[3] = 20 + ip_data_bytes;
        // Identification
        frame.data[4] = 0xFA;
        frame.data[5] = 0xCE;
        // Fragment Offset
        frame.data[6] = 0x00;
        frame.data[7] = 0x00;
        // Time to Live = 255 for max hops
        frame.data[8] = 0xFF;
        // Protocol = 1 for ICMP
        frame.data[9] = 0x00;
        // Header Checksum
        frame.data[10] = 0x00;
        frame.data[11] = 0x00;
        // Source IP Address
        frame.data[12] = src_ip[0];
        frame.data[13] = src_ip[1];
        frame.data[14] = src_ip[2];
        frame.data[15] = src_ip[3];
        // Destination IP Address
        frame.data[16] = dst_ip[0];
        frame.data[17] = dst_ip[1];
        frame.data[18] = dst_ip[2];
        frame.data[19] = dst_ip[3];
        chksum_ip_header();
}

void print_arp_frame()
{
   printf("Sender MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n",

frame.src_mac[0],frame.src_mac[1],frame.src_mac[2],frame.src_mac[3],frame.src_mac[4],frame.src_mac[
5]);
   printf("Sender IP Address = %d.%d.%d.%d\n",
         frame.data[14],frame.data[15],frame.data[16],frame.data[17]);
   printf("Target MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n",

frame.dst_mac[0],frame.dst_mac[1],frame.dst_mac[2],frame.dst_mac[3],frame.dst_mac[4],frame.dst_mac[
5]);
   printf("Target IP Address = %d.%d.%d.%d\n",
         frame.data[24],frame.data[25],frame.data[26],frame.data[27]);
   printf("\n");
}

void print_ip_frame()
{
   printf("Source MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n",

frame.src_mac[0],frame.src_mac[1],frame.src_mac[2],frame.src_mac[3],frame.src_mac[4],frame.src_mac[
5]);
   printf("Source IP Address = %d.%d.%d.%d\n",
         frame.data[12],frame.data[13],frame.data[14],frame.data[15]);
   printf("Destination MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n",

frame.dst_mac[0],frame.dst_mac[1],frame.dst_mac[2],frame.dst_mac[3],frame.dst_mac[4],frame.dst_mac[
5]);
```

```
   printf("Destination IP Address = %d.%d.%d.%d\n",
         frame.data[16],frame.data[17],frame.data[18],frame.data[19]);
   printf("\n");
   if ( ip_data == true )
   {
            printf("IP Data = %02x:%02x:%02x\n",
            frame.data[20],frame.data[21],frame.data[22]);
            ip_data = false;
   }
}

//
// 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 0x806:
             arp_queue.send(PACKET,buf.data,n);
             break;
      }
   }
}

//
// Toy function to print something interesting when an ARP frame arrives
//
void *arp_protocol_loop(void *arg)
{
   octet buf[1500];
   event_kind event;
   bool request = false;
   bool reply = false;
   bool sent = false;
   int request_count = 0;
   int reply_count = 0;

   /* buf_key
   00 to 01 = Hardware Type (0x0001=Ethernet)
   02 to 03 = Protocol Type (0x0800=IPv4 0x86DD=IPv6)
   04 = Hardware Size (6=Ethernet)
   05 = Protocol Size (4=IPv4 16=IPv6)
   06 to 07 = Opcode (1=Request 2=Reply)
   08 to 13 = Sender's MAC Address
   14 to 17 = Sender's IP Address
   18 to 23 = Target MAC Address
   24 to 27 = Target IP Address
   28 to .. = Data
   */
```

```
my_mac[0] = net.get_mac()[0];
my_mac[1] = net.get_mac()[1];
my_mac[2] = net.get_mac()[2];
my_mac[3] = net.get_mac()[3];
my_mac[4] = net.get_mac()[4];
my_mac[5] = net.get_mac()[5];

printf("Target IP Address = %d.%d.%d.%d\n\n",
   ip_target[0],ip_target[1],ip_target[2],ip_target[3]);

while ( 1 )
{
    arp_queue.recv(&event, buf, sizeof(buf));
    for (int arp_byte = 0; arp_byte < 42; arp_byte++) // Read first 42 bytes
    {
           if ( arp_byte == 7 ) // Detect the opcode byte
           {
                  if ( buf[arp_byte] == 1 ) // Is this a request?
                  {
                         if ( request_count == 0 )
                         {
                                printf("ARP request detected.\n\n");
                                request_count++;
                         }
                         request = true;
                  }
                  else if ( buf[arp_byte] == 2 ) // Is this a reply?
                  {
                         if ( reply_count == 0 )
                         {
                                printf("ARP reply detected.\n\n");
                                reply_count++;
                         }
                         reply = true;
                  }
           }
    }
    if ( sent == true ) // Save MAC address of target IP.
           {
                  if ( reply == true ) // Is this a reply?
                  {
                         reply = false;
                         // Is target IP address my IP address?
                         if ( buf[24] == my_ip[0] &&
                                buf[25] == my_ip[1] &&
                                buf[26] == my_ip[2] &&
                                buf[27] == my_ip[3] )
                         {
                                printf("ARP reply to my IP address received.\n\n");
                                //>> mac_target = buf[8:13]
                                mac_target[0] = buf[8];
                                mac_target[1] = buf[9];
                                mac_target[2] = buf[10];
                                mac_target[3] = buf[11];
                                mac_target[4] = buf[12];
                                mac_target[5] = buf[13];
```

```
                                printf("Target IP Address = %d.%d.%d.%d\n",
                                       target_ip[0],target_ip[1],target_ip[2],target_ip[3]);
                                printf("Target MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n\n",

      mac_target[0],mac_target[1],mac_target[2],mac_target[3],mac_target[4],mac_target[5]);

                                printf("Sending IP frame...\n\n");
                                // Set destination MAC address equal to target MAC address.
                                // Set source MAC address equal to current machine's MAC address.
                                ethernet_frame(mac_target,my_mac,IP_type);
                                // Create the IP frame payload.
                                //>> src_ip = my_ip
                                src_ip[0] = my_ip[0];
                                src_ip[1] = my_ip[1];
                                src_ip[2] = my_ip[2];
                                src_ip[3] = my_ip[3];
                                //>> dst_ip = target_ip
                                dst_ip[0] = target_ip[0];
                                dst_ip[1] = target_ip[1];
                                dst_ip[2] = target_ip[2];
                                dst_ip[3] = target_ip[3];
                                ip_data = true;
                                ip_data_bytes = 3;
                                ip_frame();
                                frame.data[20] = 0xAB;
                                frame.data[21] = 0xCD;
                                frame.data[22] = 0xEF;
                                print_ip_frame();
                                // Send the ethernet frame containing IP frame payload.
                                net.send_frame(&frame,42);
                                printf("IP frame has been sent. END\n\n");
                                goto finish;
                        }
                }
                else
                        printf("Opcode ERROR!\n\n");
        }
        else if ( sent == false ) // Send ARP request to target IP.
        {
                printf("Sending ARP request...\n\n");
                // Set ARP request destination MAC address equal to broadcast MAC address.
                // Set ARP request source MAC address equal to current machine's MAC address.
                ethernet_frame(mac_broadcast,my_mac,ARP_type);
                // Create the ARP request payload.
                opcode[0] = 0;
                opcode[1] = 1; // 1=Request
                //>> sender_mac = frame.src_mac
                sender_mac[0] = frame.src_mac[0];
                sender_mac[1] = frame.src_mac[1];
                sender_mac[2] = frame.src_mac[2];
                sender_mac[3] = frame.src_mac[3];
                sender_mac[4] = frame.src_mac[4];
                sender_mac[5] = frame.src_mac[5];
                //>> sender_ip = my_ip
                sender_ip[0] = my_ip[0];
                sender_ip[1] = my_ip[1];
                sender_ip[2] = my_ip[2];
```

```
                    sender_ip[3] = my_ip[3];
                    //>> target_mac = frame.dst_mac
                    target_mac[0] = frame.dst_mac[0];
                    target_mac[1] = frame.dst_mac[1];
                    target_mac[2] = frame.dst_mac[2];
                    target_mac[3] = frame.dst_mac[3];
                    target_mac[4] = frame.dst_mac[4];
                    target_mac[5] = frame.dst_mac[5];
                    in_network();
                    if ( ip_result == true ) // Is target IP in local network?
                    {
                           // Yes. Send ARP request to target IP.
                           target_ip[0] = ip_target[0];
                           target_ip[1] = ip_target[1];
                           target_ip[2] = ip_target[2];
                           target_ip[3] = ip_target[3];
                    }
                    else
                    {
                           // No. Send ARP request to gateway IP.
                           target_ip[0] = gateway_ip[0];
                           target_ip[1] = gateway_ip[1];
                           target_ip[2] = gateway_ip[2];
                           target_ip[3] = gateway_ip[3];
                    }
                    arp_frame();
                    print_arp_frame();
                    // Send the ethernet frame containing ARP request payload.
                    net.send_frame(&frame,42);
                    sent = true;
                    printf("ARP request has been sent.\n\n");
             }
             else
                    printf("ERROR!\n\n");
   }
finish:;
}

//
// if you're going to have pthreads, you'll need some thread descriptors
//
pthread_t loop_thread, arp_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);
   for ( ; ; )
      sleep(1);
}
```

4.3.1 Program Code – Procedure 3: ARP

```
#include "frameio.h"
#include "util.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

frameio net;             // gives us access to the raw network
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
};

octet my_ip[4] = { 192, 168, 1, 10 };
octet my_mac[6];
// ip_target = local ip or remote ip (choose one)
// octet ip_target[4] = { 192, 168, 1, 20 }; // local ip
octet ip_target[4] = { 172, 217, 1, 206 }; // remote ip
octet mac_target[6];
octet mac_broadcast[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
octet subnet_mask[4] = { 255, 255, 255, 0 };
octet gateway_ip[4] = { 192, 168, 1, 1 };
octet network_ip[4];

// returns true if target IP is on the local network
bool ip_result;
void in_network()
{
       network_ip[0] = my_ip[0] & subnet_mask[0];
       network_ip[1] = my_ip[1] & subnet_mask[1];
       network_ip[2] = my_ip[2] & subnet_mask[2];
       network_ip[3] = my_ip[3] & subnet_mask[3];
       if ( ip_target[0] == network_ip[0] &&
               ip_target[1] == network_ip[1] &&
               ip_target[2] == network_ip[2] )
       {
              printf("Target IP is on local network.\n\n");
              ip_result = true;
       }
       else
       {
              printf("Target IP is on remote network.\n\n");
              ip_result = false;
       }
}

ether_frame frame;
```

```
octet IP_type[2] = { 0x08, 0x00 };
octet ARP_type[2] = { 0x08, 0x06 };

void ethernet_frame(octet dst_mac[6], octet src_mac[6], octet ether_type[2])
{
	// Destination MAC Address
	frame.dst_mac[0] = dst_mac[0];
	frame.dst_mac[1] = dst_mac[1];
	frame.dst_mac[2] = dst_mac[2];
	frame.dst_mac[3] = dst_mac[3];
	frame.dst_mac[4] = dst_mac[4];
	frame.dst_mac[5] = dst_mac[5];
	// Source MAC Address
	frame.src_mac[0] = src_mac[0];
	frame.src_mac[1] = src_mac[1];
	frame.src_mac[2] = src_mac[2];
	frame.src_mac[3] = src_mac[3];
	frame.src_mac[4] = src_mac[4];
	frame.src_mac[5] = src_mac[5];
	// Ethernet Type = 0x0800 for IP, 0x0806 for ARP
	frame.prot[0] = ether_type[0];
	frame.prot[1] = ether_type[1];
}

octet opcode[2];
octet sender_mac[6];
octet sender_ip[4];
octet target_mac[6];
octet target_ip[4];

void arp_frame()
{
	// Hardware Type = 0x0001 for Ethernet
	frame.data[0] = 0x00;
	frame.data[1] = 0x01;
	// Protocol Type = 0x0800 for IPv4, 0x86DD for IPv6
	frame.data[2] = 0x08;
	frame.data[3] = 0x00;
	// Hardware Size = 6 for Ethernet
	frame.data[4] = 6;
	// Protocol Size = 4 for IPv4, 16 for IPv6
	frame.data[5] = 4;
	// Opcode = 1 for Request, 2 for Reply
	frame.data[6] = opcode[0];
	frame.data[7] = opcode[1];
	// Sender's MAC Address
	frame.data[8] = sender_mac[0];
	frame.data[9] = sender_mac[1];
	frame.data[10] = sender_mac[2];
	frame.data[11] = sender_mac[3];
	frame.data[12] = sender_mac[4];
	frame.data[13] = sender_mac[5];
	// Sender's IP Address
	frame.data[14] = sender_ip[0];
	frame.data[15] = sender_ip[1];
	frame.data[16] = sender_ip[2];
	frame.data[17] = sender_ip[3];
```

```
	// Target MAC Address
	frame.data[18] = target_mac[0];
	frame.data[19] = target_mac[1];
	frame.data[20] = target_mac[2];
	frame.data[21] = target_mac[3];
	frame.data[22] = target_mac[4];
	frame.data[23] = target_mac[5];
	// Target IP Address
	frame.data[24] = target_ip[0];
	frame.data[25] = target_ip[1];
	frame.data[26] = target_ip[2];
	frame.data[27] = target_ip[3];
}

int chksum(octet *s, int bytes, int initial)
{
   long sum = initial;
   int i;
   for ( i=0; i<bytes-1; i+=2 )
   {
      sum += s[i]*256 + s[i+1];
   }
   //
   // handle the odd byte
   //
   if ( i < bytes ) sum += s[i]*256;
   //
   // wrap carries back into sum
   //
   while ( sum > 0xffff ) sum = (sum & 0xffff) + (sum >> 16);
   return sum;
}

int ip_header_bytes = 20;
void chksum_ip_header()
{
	octet ip_header[ip_header_bytes];
	for (int i = 0; i < ip_header_bytes; i++)
		ip_header[i] = frame.data[i];
	int sum = chksum((octet *)ip_header,ip_header_bytes,0);
	frame.data[10] = ~sum >> 8;
	frame.data[11] = ~sum & 0xFF;
}

int data_bytes = 3;

octet src_ip[4];
octet dst_ip[4];
bool ip_data;
int ip_data_bytes;

void ip_frame()
{
	// IP Version = 0b0100**** for IPv4, 0b0110**** for IPv6
	// IP Header Length = 0b****0101 for no Options
	frame.data[0] = 0x45;
	// Type of Service
```

```
	frame.data[1] = 0x00;
	// Total Length = IP Header Length (20 bytes) + Data
	frame.data[2] = 0x00;
	frame.data[3] = 20 + ip_data_bytes;
	// Identification
	frame.data[4] = 0xFA;
	frame.data[5] = 0xCE;
	// Fragment Offset
	frame.data[6] = 0x00;
	frame.data[7] = 0x00;
	// Time to Live = 255 for max hops
	frame.data[8] = 0xFF;
	// Protocol = 1 for ICMP
	frame.data[9] = 0x01;
	// Header Checksum
	frame.data[10] = 0x00;
	frame.data[11] = 0x00;
	// Source IP Address
	frame.data[12] = src_ip[0];
	frame.data[13] = src_ip[1];
	frame.data[14] = src_ip[2];
	frame.data[15] = src_ip[3];
	// Destination IP Address
	frame.data[16] = dst_ip[0];
	frame.data[17] = dst_ip[1];
	frame.data[18] = dst_ip[2];
	frame.data[19] = dst_ip[3];
	chksum_ip_header();
}

int icmp_header_bytes = 8;
void chksum_icmp_header_data()
{
	int icmp_header_data_bytes = icmp_header_bytes + data_bytes;
	octet icmp_header_data[icmp_header_data_bytes];
	int I = ip_header_bytes + icmp_header_data_bytes;
	for (int i = ip_header_bytes; i < I; i++)
		icmp_header_data[i-ip_header_bytes] = frame.data[i];
	int sum = chksum((octet *)icmp_header_data,icmp_header_data_bytes,0);
	frame.data[22] = ~sum >> 8;
	frame.data[23] = ~sum & 0xFF;
}

int icmp_type;
octet icmp_identifier[2];
octet icmp_sequence_no[2];
bool icmp_data;
int icmp_data_bytes;

void icmp_frame()
{
	// Type = 8 for Request, 0 for Reply
	frame.data[20] = icmp_type;
	// Code
	frame.data[21] = 0x00;
	// Checksum
	frame.data[22] = 0x00;
```

```
        frame.data[23] = 0x00;
        // Identifier
        frame.data[24] = icmp_identifier[0];
        frame.data[25] = icmp_identifier[1];
        // Sequence No.
        frame.data[26] = icmp_sequence_no[0];
        frame.data[27] = icmp_sequence_no[1];
}

void print_arp_frame()
{
   printf("Sender MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n",

frame.src_mac[0],frame.src_mac[1],frame.src_mac[2],frame.src_mac[3],frame.src_mac[4],frame.src_mac[
5]);
   printf("Sender IP Address = %d.%d.%d.%d\n",
         frame.data[14],frame.data[15],frame.data[16],frame.data[17]);
   printf("Target MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n",

frame.dst_mac[0],frame.dst_mac[1],frame.dst_mac[2],frame.dst_mac[3],frame.dst_mac[4],frame.dst_mac[
5]);
   printf("Target IP Address = %d.%d.%d.%d\n",
         frame.data[24],frame.data[25],frame.data[26],frame.data[27]);
   printf("\n");
}

void print_ip_frame()
{
   printf("Source MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n",

frame.src_mac[0],frame.src_mac[1],frame.src_mac[2],frame.src_mac[3],frame.src_mac[4],frame.src_mac[
5]);
   printf("Source IP Address = %d.%d.%d.%d\n",
         frame.data[12],frame.data[13],frame.data[14],frame.data[15]);
   printf("Destination MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n",

frame.dst_mac[0],frame.dst_mac[1],frame.dst_mac[2],frame.dst_mac[3],frame.dst_mac[4],frame.dst_mac[
5]);
   printf("Destination IP Address = %d.%d.%d.%d\n",
         frame.data[16],frame.data[17],frame.data[18],frame.data[19]);
   printf("\n");
}

void print_icmp_frame()
{
   if ( icmp_type == 8 )
         printf("Type = 8 (Request)\n");
   else if ( icmp_type == 0 )
         printf("Type = 0 (Reply)\n");
   else
         printf("Type = ERROR!\n");
   printf("Identifier = %02x %02x\n",icmp_identifier[0],icmp_identifier[1]);
   printf("Sequence No. = %02x %02x\n",icmp_sequence_no[0],icmp_sequence_no[1]);
   printf("\n");
   if ( icmp_data == true )
   {
            printf("ICMP Data = %02x:%02x:%02x\n",
```

```
            frame.data[28],frame.data[29],frame.data[30]);
            icmp_data = false;
   }
   printf("\n");
}

//
// 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 0x806:
             arp_queue.send(PACKET,buf.data,n);
             break;
      }
   }
}

//
// Toy function to print something interesting when an ARP frame arrives
//
void *arp_protocol_loop(void *arg)
{
   octet buf[1500];
   event_kind event;
   bool request = false;
   bool reply = false;
   bool sent = false;
   int request_count = 0;
   int reply_count = 0;

   /* buf_key
   00 to 01 = Hardware Type (0x0001=Ethernet)
   02 to 03 = Protocol Type (0x0800=IPv4 0x86DD=IPv6)
   04 = Hardware Size (6=Ethernet)
   05 = Protocol Size (4=IPv4 16=IPv6)
   06 to 07 = Opcode (1=Request 2=Reply)
   08 to 13 = Sender's MAC Address
   14 to 17 = Sender's IP Address
   18 to 23 = Target MAC Address
   24 to 27 = Target IP Address
   28 to .. = Data
   */

   my_mac[0] = net.get_mac()[0];
   my_mac[1] = net.get_mac()[1];
   my_mac[2] = net.get_mac()[2];
   my_mac[3] = net.get_mac()[3];
   my_mac[4] = net.get_mac()[4];
```

```
my_mac[5] = net.get_mac()[5];

printf("Target IP Address = %d.%d.%d.%d\n\n",
   ip_target[0],ip_target[1],ip_target[2],ip_target[3]);

while ( 1 )
{
    arp_queue.recv(&event, buf, sizeof(buf));
    for (int arp_byte = 0; arp_byte < 42; arp_byte++) // Read first 42 bytes
    {
           if ( arp_byte == 7 ) // Detect the opcode byte
           {
                  if ( buf[arp_byte] == 1 ) // Is this a request?
                  {
                         if ( request_count == 0 )
                         {
                                printf("ARP request detected.\n\n");
                                request_count++;
                         }
                         request = true;
                  }
                  else if ( buf[arp_byte] == 2 ) // Is this a reply?
                  {
                         if ( reply_count == 0 )
                         {
                                printf("ARP reply detected.\n\n");
                                reply_count++;
                         }
                         reply = true;
                  }
           }
    }
    if ( sent == true ) // Save MAC address of target IP.
           {
                  if ( reply == true ) // Is this a reply?
                  {
                         reply = false;
                         // Is target IP address my IP address?
                         if ( buf[24] == my_ip[0] &&
                                buf[25] == my_ip[1] &&
                                buf[26] == my_ip[2] &&
                                buf[27] == my_ip[3] )
                         {
                                printf("ARP reply to my IP address received.\n\n");
                                //>> mac_target = buf[8:13]
                                mac_target[0] = buf[8];
                                mac_target[1] = buf[9];
                                mac_target[2] = buf[10];
                                mac_target[3] = buf[11];
                                mac_target[4] = buf[12];
                                mac_target[5] = buf[13];

                                printf("Target IP Address = %d.%d.%d.%d\n",
                                       target_ip[0],target_ip[1],target_ip[2],target_ip[3]);
                                printf("Target MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n\n",

    mac_target[0],mac_target[1],mac_target[2],mac_target[3],mac_target[4],mac_target[5]);
```

```
                        printf("Sending ICMP request...\n\n");
                        // Set destination MAC address equal to target MAC address.
                        // Set source MAC address equal to current machine's MAC address.
                        ethernet_frame(mac_target,my_mac,IP_type);
                        // Create the ICMP request payload.
                        //>> src_ip = my_ip
                        src_ip[0] = my_ip[0];
                        src_ip[1] = my_ip[1];
                        src_ip[2] = my_ip[2];
                        src_ip[3] = my_ip[3];
                        //>> dst_ip = target_ip
                        dst_ip[0] = target_ip[0];
                        dst_ip[1] = target_ip[1];
                        dst_ip[2] = target_ip[2];
                        dst_ip[3] = target_ip[3];
                        ip_data = true;
                        // ip_data_bytes = icmp_header_bytes (8) + data_bytes (3)
                        ip_data_bytes = icmp_header_bytes + data_bytes;
                        ip_frame();
                        icmp_identifier[0] = 0xAC;
                        icmp_identifier[1] = 0xED;
                        icmp_sequence_no[0] = 0x00;
                        icmp_sequence_no[1] = 0x01;
                        icmp_data = true;
                        icmp_data_bytes = data_bytes;
                        // printf("icmp_data_bytes = %d\n",icmp_data_bytes);
                        icmp_type = 8; // 8=Request
                        icmp_frame();
                        frame.data[28] = 0xAB;
                        frame.data[29] = 0xCD;
                        frame.data[30] = 0xEF;
                        chksum_icmp_header_data();
                        print_ip_frame();
                        print_icmp_frame();
                        // Send the ethernet frame containing ICMP request payload.
                        net.send_frame(&frame,42 + data_bytes);
                        printf("ICMP request has been sent. END\n");
                        goto finish;
                }
        }
        else
                printf("Opcode ERROR!\n\n");
}
else if ( sent == false ) // Send ARP request to target IP.
{
        printf("Sending ARP request...\n\n");
        // Set ARP request destination MAC address equal to broadcast MAC address.
        // Set ARP request source MAC address equal to current machine's MAC address.
        ethernet_frame(mac_broadcast,my_mac,ARP_type);
        // Create the ARP request payload.
        opcode[0] = 0;
        opcode[1] = 1; // 1=Request
        //>> sender_mac = frame.src_mac
        sender_mac[0] = frame.src_mac[0];
        sender_mac[1] = frame.src_mac[1];
        sender_mac[2] = frame.src_mac[2];
```

```
                    sender_mac[3] = frame.src_mac[3];
                    sender_mac[4] = frame.src_mac[4];
                    sender_mac[5] = frame.src_mac[5];
                    //>> sender_ip = my_ip
                    sender_ip[0] = my_ip[0];
                    sender_ip[1] = my_ip[1];
                    sender_ip[2] = my_ip[2];
                    sender_ip[3] = my_ip[3];
                    //>> target_mac = frame.dst_mac
                    target_mac[0] = frame.dst_mac[0];
                    target_mac[1] = frame.dst_mac[1];
                    target_mac[2] = frame.dst_mac[2];
                    target_mac[3] = frame.dst_mac[3];
                    target_mac[4] = frame.dst_mac[4];
                    target_mac[5] = frame.dst_mac[5];
                    in_network();
                    if ( ip_result == true ) // Is target IP in local network?
                    {
                           // Yes. Send ARP request to target IP.
                           target_ip[0] = ip_target[0];
                           target_ip[1] = ip_target[1];
                           target_ip[2] = ip_target[2];
                           target_ip[3] = ip_target[3];
                    }
                    else
                    {
                           // No. Send ARP request to gateway IP.
                           target_ip[0] = gateway_ip[0];
                           target_ip[1] = gateway_ip[1];
                           target_ip[2] = gateway_ip[2];
                           target_ip[3] = gateway_ip[3];
                    }
                    arp_frame();
                    print_arp_frame();
                    // Send the ethernet frame containing ARP request payload.
                    net.send_frame(&frame,42);
                    sent = true;
                    printf("ARP request has been sent.\n\n");
             }
             else
                    printf("ERROR!\n\n");
   }
finish:;
}

//
// if you're going to have pthreads, you'll need some thread descriptors
//
pthread_t loop_thread, arp_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);
   for ( ; ; )
      sleep(1);
}
```

4.3.2 Program Code – Procedure 3: IP

```
#include "frameio.h"
#include "util.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

frameio net;             // gives us access to the raw network
message_queue ip_queue;  // message queue for the IP 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
};

octet my_mac[6];

// Choose one value for my_ip:
// octet my_ip[4] = { 192, 168, 1, 20 };
octet my_ip[4] = { 192, 168, 1, 10 };

// Choose one value for mac_target:
// octet mac_target[6] = { 0x00, 0x1A, 0xA0, 0xAC, 0xB0, 0xE8 }; // MAC address of 192.168.1.20
// octet mac_target[6] = { 0x00, 0x1A, 0xA0, 0xAC, 0xDF, 0x57 }; // MAC address of 192.168.1.10
octet mac_target[6] = { 0x00, 0x1C, 0x10, 0xF5, 0x0C, 0xAC }; // MAC address of 192.168.1.1

ether_frame frame;

octet IP_type[2] = { 0x08, 0x00 };
octet ARP_type[2] = { 0x08, 0x06 };

void ethernet_frame(octet dst_mac[6], octet src_mac[6], octet ether_type[2])
{
       // Destination MAC Address
       frame.dst_mac[0] = dst_mac[0];
       frame.dst_mac[1] = dst_mac[1];
       frame.dst_mac[2] = dst_mac[2];
       frame.dst_mac[3] = dst_mac[3];
       frame.dst_mac[4] = dst_mac[4];
       frame.dst_mac[5] = dst_mac[5];
       // Source MAC Address
       frame.src_mac[0] = src_mac[0];
       frame.src_mac[1] = src_mac[1];
```

```
        frame.src_mac[2] = src_mac[2];
        frame.src_mac[3] = src_mac[3];
        frame.src_mac[4] = src_mac[4];
        frame.src_mac[5] = src_mac[5];
        // Ethernet Type = 0x0800 for IP, 0x0806 for ARP
        frame.prot[0] = ether_type[0];
        frame.prot[1] = ether_type[1];
}

int chksum(octet *s, int bytes, int initial)
{
   long sum = initial;
   int i;
   for ( i=0; i<bytes-1; i+=2 )
   {
      sum += s[i]*256 + s[i+1];
   }
   //
   // handle the odd byte
   //
   if ( i < bytes ) sum += s[i]*256;
   //
   // wrap carries back into sum
   //
   while ( sum > 0xffff ) sum = (sum & 0xffff) + (sum >> 16);
   return sum;
}

int ip_header_bytes = 20;
void chksum_ip_header()
{
        octet ip_header[ip_header_bytes];
        for (int i = 0; i < ip_header_bytes; i++)
                ip_header[i] = frame.data[i];
        int sum = chksum((octet *)ip_header,ip_header_bytes,0);
        frame.data[10] = ~sum >> 8;
        frame.data[11] = ~sum & 0xFF;
}

int data_bytes = 3;

octet src_ip[4];
octet dst_ip[4];
bool ip_data;
int ip_data_bytes;

void ip_frame()
{
        // IP Version = 0b0100**** for IPv4, 0b0110**** for IPv6
        // IP Header Length = 0b****0101 for no Options
        frame.data[0] = 0x45;
        // Type of Service
        frame.data[1] = 0x00;
        // Total Length = IP Header Length (20 bytes) + Data
        frame.data[2] = 0x00;
        frame.data[3] = 20 + ip_data_bytes;
        // Identification
```

```
	frame.data[4] = 0xFA;
	frame.data[5] = 0xCE;
	// Fragment Offset
	frame.data[6] = 0x00;
	frame.data[7] = 0x00;
	// Time to Live = 255 for max hops
	frame.data[8] = 0xFF;
	// Protocol = 1 for ICMP
	frame.data[9] = 0x01;
	// Header Checksum
	frame.data[10] = 0x00;
	frame.data[11] = 0x00;
	// Source IP Address
	frame.data[12] = src_ip[0];
	frame.data[13] = src_ip[1];
	frame.data[14] = src_ip[2];
	frame.data[15] = src_ip[3];
	// Destination IP Address
	frame.data[16] = dst_ip[0];
	frame.data[17] = dst_ip[1];
	frame.data[18] = dst_ip[2];
	frame.data[19] = dst_ip[3];
	chksum_ip_header();
}

int icmp_header_bytes = 8;
void chksum_icmp_header_data()
{
	int icmp_header_data_bytes = icmp_header_bytes + data_bytes;
	octet icmp_header_data[icmp_header_data_bytes];
	int I = ip_header_bytes + icmp_header_data_bytes;
	for (int i = ip_header_bytes; i < I; i++)
		icmp_header_data[i-ip_header_bytes] = frame.data[i];
	int sum = chksum((octet *)icmp_header_data,icmp_header_data_bytes,0);
	frame.data[22] = ~sum >> 8;
	frame.data[23] = ~sum & 0xFF;
}

int icmp_type;
octet icmp_identifier[2];
octet icmp_sequence_no[2];
bool icmp_data;
int icmp_data_bytes;

void icmp_frame()
{
	// Type = 8 for Request, 0 for Reply
	frame.data[20] = icmp_type;
	// Code
	frame.data[21] = 0x00;
	// Checksum
	frame.data[22] = 0x00;
	frame.data[23] = 0x00;
	// Identifier
	frame.data[24] = icmp_identifier[0];
	frame.data[25] = icmp_identifier[1];
	// Sequence No.
```

```
       frame.data[26] = icmp_sequence_no[0];
       frame.data[27] = icmp_sequence_no[1];
}

void print_ip_frame()
{
   printf("Source MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n",

frame.src_mac[0],frame.src_mac[1],frame.src_mac[2],frame.src_mac[3],frame.src_mac[4],frame.src_mac[
5]);
   printf("Source IP Address = %d.%d.%d.%d\n",
         frame.data[12],frame.data[13],frame.data[14],frame.data[15]);
   printf("Destination MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n",

frame.dst_mac[0],frame.dst_mac[1],frame.dst_mac[2],frame.dst_mac[3],frame.dst_mac[4],frame.dst_mac[
5]);
   printf("Destination IP Address = %d.%d.%d.%d\n",
         frame.data[16],frame.data[17],frame.data[18],frame.data[19]);
   printf("\n");
}

void print_icmp_frame()
{
   if ( icmp_type == 8 )
         printf("Type = 8 (Request)\n");
   else if ( icmp_type == 0 )
         printf("Type = 0 (Reply)\n");
   else
         printf("Type = ERROR!\n");
   printf("Identifier = %02x %02x\n",icmp_identifier[0],icmp_identifier[1]);
   printf("Sequence No. = %02x %02x\n",icmp_sequence_no[0],icmp_sequence_no[1]);
   printf("\n");
   if ( icmp_data == true )
   {
            printf("ICMP Data = %02x:%02x:%02x\n",
            frame.data[28],frame.data[29],frame.data[30]);
            icmp_data = false;
   }
   printf("\n");
}

//
// 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 + data_bytes ) continue; // bad frame!
      switch ( buf.prot[0]<<8 | buf.prot[1] )
      {
          case 0x800:
             ip_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 sent = 0;
   int sent_max = 4;
   bool icmp;

   /* buf_key
   00_0:3 = IP Version (0b0100_=IPv4 0b0110=IPv6)
   00_4:7 = IP Header Length
   01 = Type of Service
   02 to 03 = Total Length
   04 to 05 = Identification
   06 to 07 = Fragment Offset
   08 = Time to Live
   09 = Protocol (0x01=ICMP)
   10 to 11 = Header Checksum
   12 to 15 = Source IP Address
   16 to 19 = Destination IP Address
   20 to .. = IP Data
   20 = ICMP Type (8=Request 0=Reply)
   21 = ICMP Code
   22 to 23 = ICMP Checksum
   24 to 25 = ICMP Identifier
   26 to 27 = ICMP Sequence No.
   28 to .. = ICMP Data
   */

   my_mac[0] = net.get_mac()[0];
   my_mac[1] = net.get_mac()[1];
   my_mac[2] = net.get_mac()[2];
   my_mac[3] = net.get_mac()[3];
   my_mac[4] = net.get_mac()[4];
   my_mac[5] = net.get_mac()[5];

   while ( 1 )
   {
            ip_queue.recv(&event, buf, sizeof(buf));
            for (int ip_byte = 0; ip_byte < 42 + data_bytes; ip_byte++) /* Read first 42 IP bytes
*/
            {
                  if ( ip_byte == 9 ) // Detect the protocol byte
                  {
                        if ( buf[ip_byte] == 1 ) // Is this ICMP?
                        {
                              printf("ICMP frame detected.\n\n");
                              icmp = true; // Yes it is an ICMP.
                        }
                  }
```

```
		}
		if ( icmp == true )
		{
			printf("ICMP frame received.\n\n");
			icmp = false;
			// Is target IP address my IP address?
			if ( buf[16] == my_ip[0] &&
				buf[17] == my_ip[1] &&
				buf[18] == my_ip[2] &&
				buf[19] == my_ip[3] )
			{
				// Set destination MAC address equal to target MAC address.
				// Set source MAC address equal to current machine's MAC address.
				ethernet_frame(mac_target,my_mac,IP_type);
				// Create the ICMP payload.
				//>> src_ip = my_ip
				src_ip[0] = my_ip[0];
				src_ip[1] = my_ip[1];
				src_ip[2] = my_ip[2];
				src_ip[3] = my_ip[3];
				//>> dst_ip = buf[12:15]
				dst_ip[0] = buf[12];
				dst_ip[1] = buf[13];
				dst_ip[2] = buf[14];
				dst_ip[3] = buf[15];
				ip_data = true;
				// ip_data_bytes = icmp_header_bytes (8) + data_bytes (3)
				ip_data_bytes = icmp_header_bytes + data_bytes;
				ip_frame();
				// icmp identifier = buf[24:25]
				icmp_identifier[0] = buf[24];
				icmp_identifier[1] = buf[25];
				// icmp_sequence_no = buf[26:27]
				icmp_sequence_no[0] = buf[26];
				icmp_sequence_no[1] = buf[27];
				icmp_data = true;
				icmp_data_bytes = data_bytes;
				if ( buf[20] == 8 ) // Is this an ICMP request? Then send ICMP reply.
				{
					printf("ICMP request has been received. Sending ICMP
reply...\n\n");
					icmp_type = 0; // 0=Reply
					icmp_frame();
					frame.data[28] = 0xFE;
					frame.data[29] = 0xDC;
					frame.data[30] = 0xBA;
					chksum_icmp_header_data();
					print_ip_frame();
					print_icmp_frame();
					// Send the ethernet frame containing ICMP reply payload.
					net.send_frame(&frame,42 + data_bytes);
					printf("ICMP reply has been sent.\n\n");
					if ( sent != sent_max )
						sent++;
					else
						goto finish;
				}
```

```
                        else if ( buf[20] == 0 ) // Is this an ICMP reply? Then send ICMP
request.
                        {
                                icmp_sequence_no[1]++; // Increment ICMP sequence number.
                                printf("ICMP reply has been received. Sending ICMP
request...\n\n");
                                icmp_type = 8; // 8=Request
                                icmp_frame();
                                frame.data[28] = 0xAB;
                                frame.data[29] = 0xCD;
                                frame.data[30] = 0xEF;
                                chksum_icmp_header_data();
                                print_ip_frame();
                                print_icmp_frame();
                                // Send the ethernet frame containing ICMP request payload.
                                net.send_frame(&frame,42 + data_bytes);
                                printf("ICMP request has been sent.\n\n");
                                if ( sent != sent_max )
                                        sent++;
                                else
                                        goto finish;
                        }
                        else
                                printf("ICMP Type ERROR!\n\n");
                }
        }
   }
finish: printf("END\n");
}

//
// if you're going to have pthreads, you'll need some thread descriptors
//
pthread_t loop_thread, ip_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(&ip_thread,NULL,ip_protocol_loop,NULL);
   for ( ; ; )
      sleep(1);
}
```