Project – Part 2

1. Objective

1. Familiar with the structure of Ethernet frame.
2. Understand the ARP mechanism.
3. Know how to use Wireshark to analyze the captured frame, and locate the MAC addresses and IP addresses.
4. Construct the ARP cache.

2. Overview

Address resolution protocol (ARP) belongs to the MAC sublayer of the data-link layer. For the purposes of this project, it provides a single service to upper layers, namely to send a frame to a node on the local network whose protocol address (in our case, IP address) is known. Layers above ARP are blissfully unaware of what MAC addresses mean or that they even exist.

This is not as easy as it might seem. If ARP does not already know what MAC address goes with the specified IP address, it must broadcast an ARP request. Once the reply comes back, it must immediately send the frame. Care must be taken that in-coming packets are handled (at all protocol layers) while awaiting the ARP reply.

3. Results

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

3.1 Procedures 1–3

- Write code to respond to an ARP request.
  - Check opcode if incoming frame is request or reply.
  - If a request, check if target IP address matches your machine's IP address.
  - If a match, send an ARP reply frame (3.1.1).

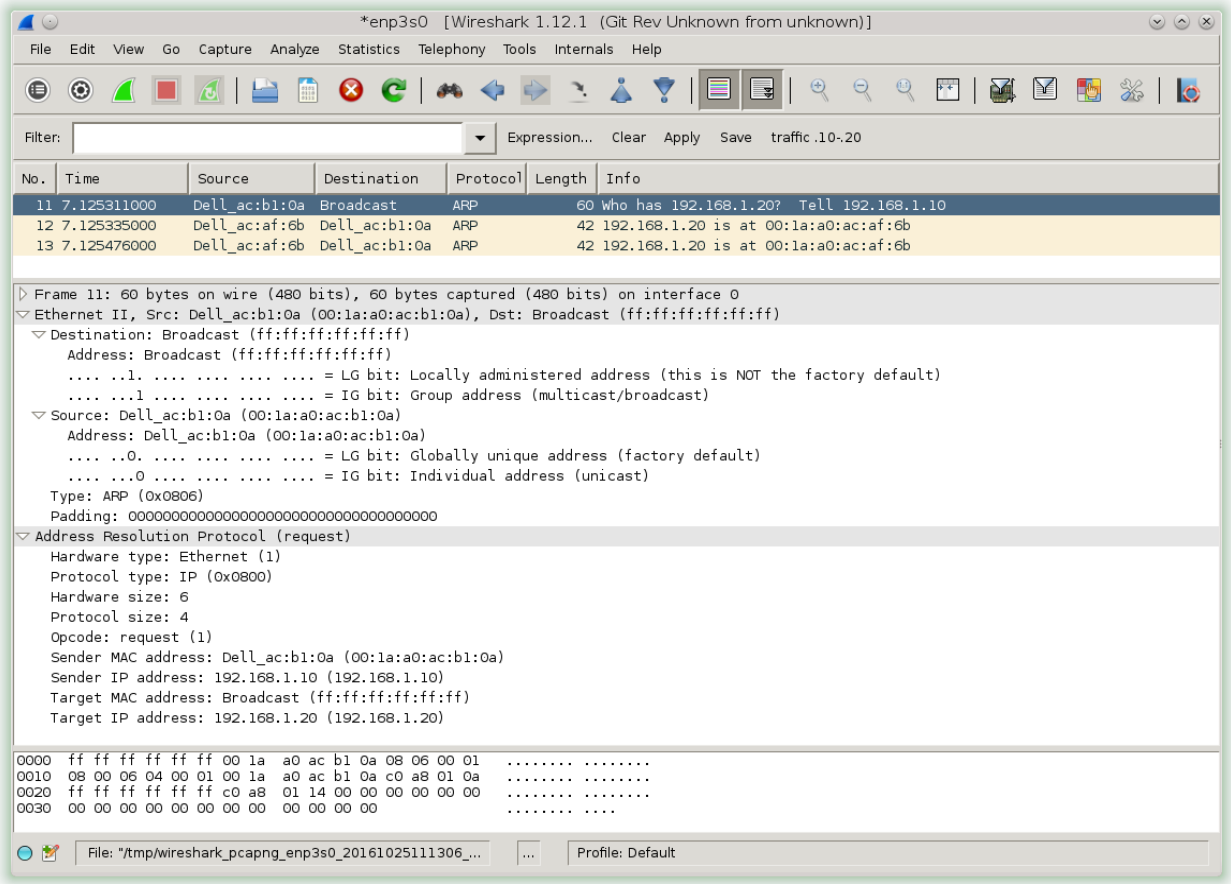
Program Output (see code in Appendix 4.1)

```
ARP request detected.

Target IP address detected.

ARP request received.

Target IP address matches.

Creating ethernet frame containing ARP reply payload...
```

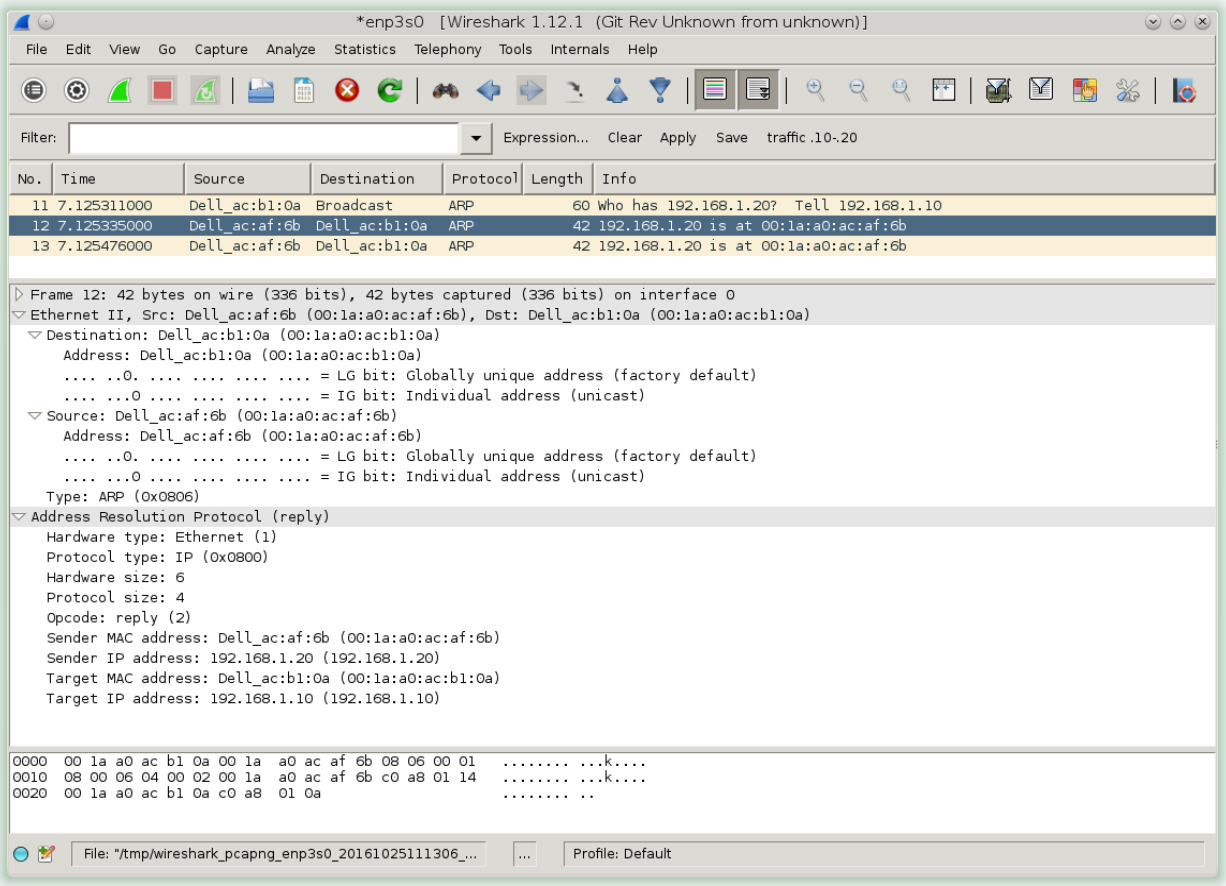
```
Sender's MAC Address = 00:1a:a0:ac:af:6b
Sender's IP Address = 192.168.1.20
Target MAC Address = 00:1a:a0:ac:b1:0a
Target IP Address = 192.168.1.10

ARP reply has been sent. END

ARP reply detected.

Target IP address detected.

ARP reply received. END
```

Network Capture

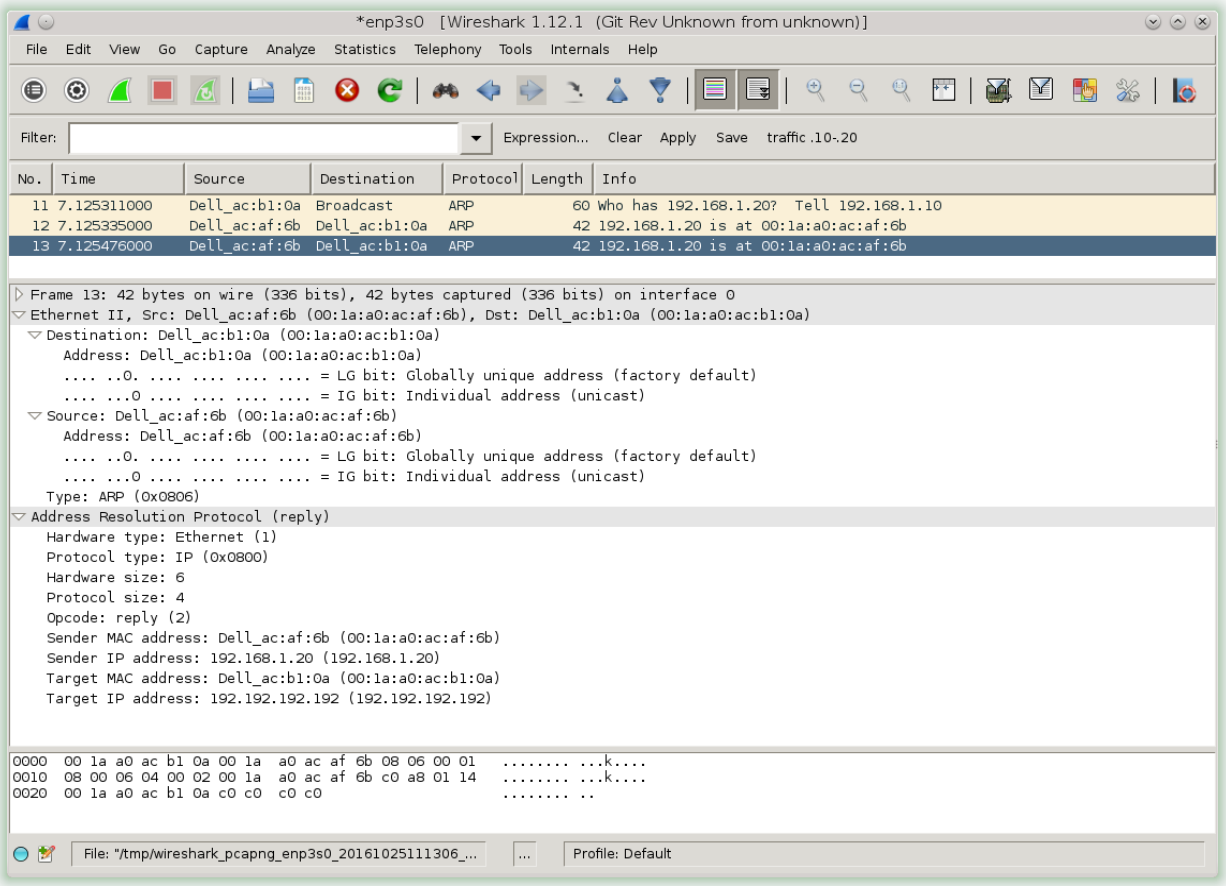
ARP Request from PC
Sender's MAC Address = 00:1A:A0:AC:B1:0A
Sender's IP Address = 192.168.1.10
Target MAC Address = FF:FF:FF:FF:FF:FF
Target IP Address = 192.168.1.20



ARP 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



ARP Reply from Code (3.1.1)
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 Procedures 4–5

- Write code to send an ARP frame with known IP address.
  - If the IP address is already in the cache, compose the frame and send it immediately (3.2.4).
  - Otherwise, send an ARP broadcast (3.2.1) and get the ARP reply (3.2.2), cache the IP and MAC address pair, then send the frame (3.2.3).

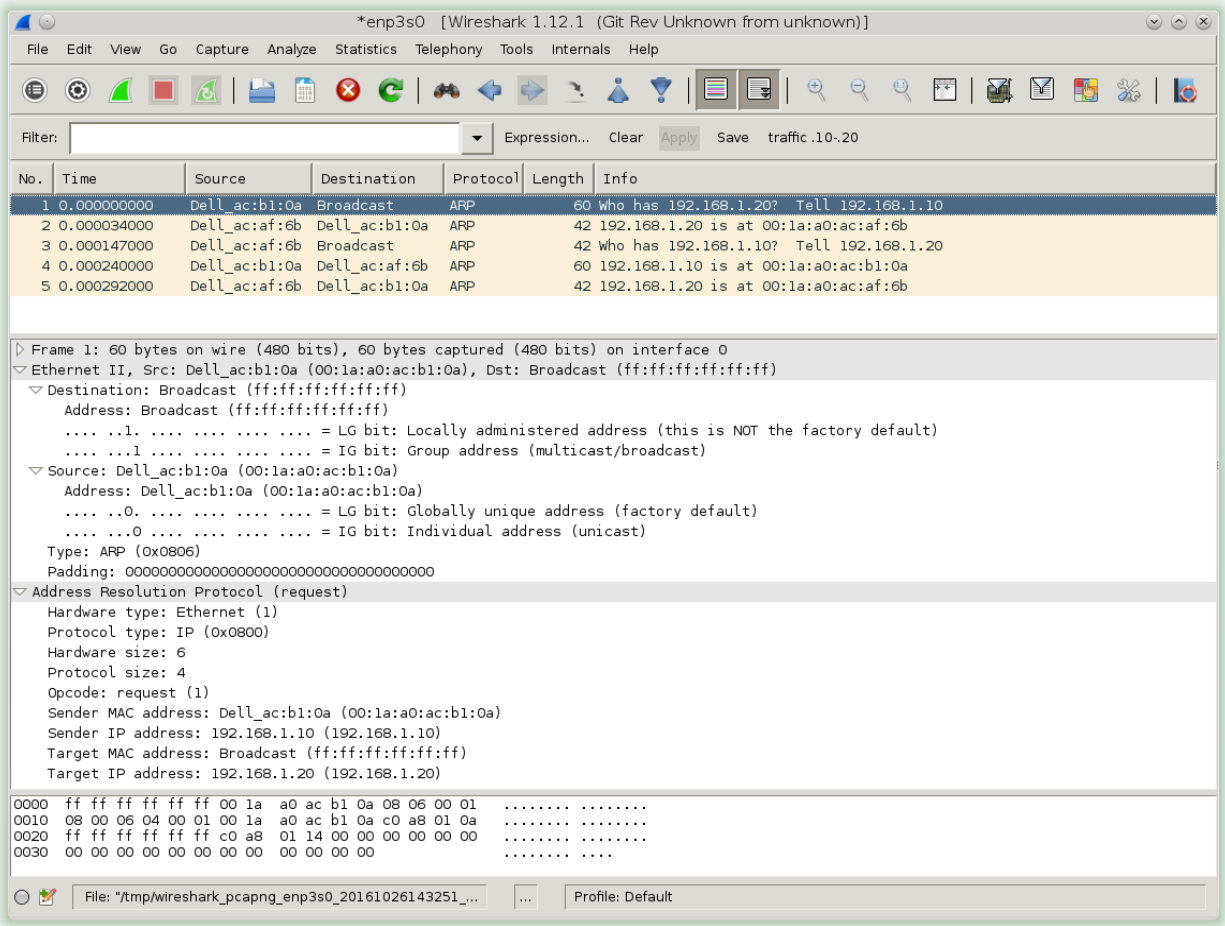
Program Output – IP-MAC not in cache (see code in Appendix 4.2)

```
Searching target IP address in IP-MAC table... NOT FOUND. Sending ARP request...

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 request has been sent.
```

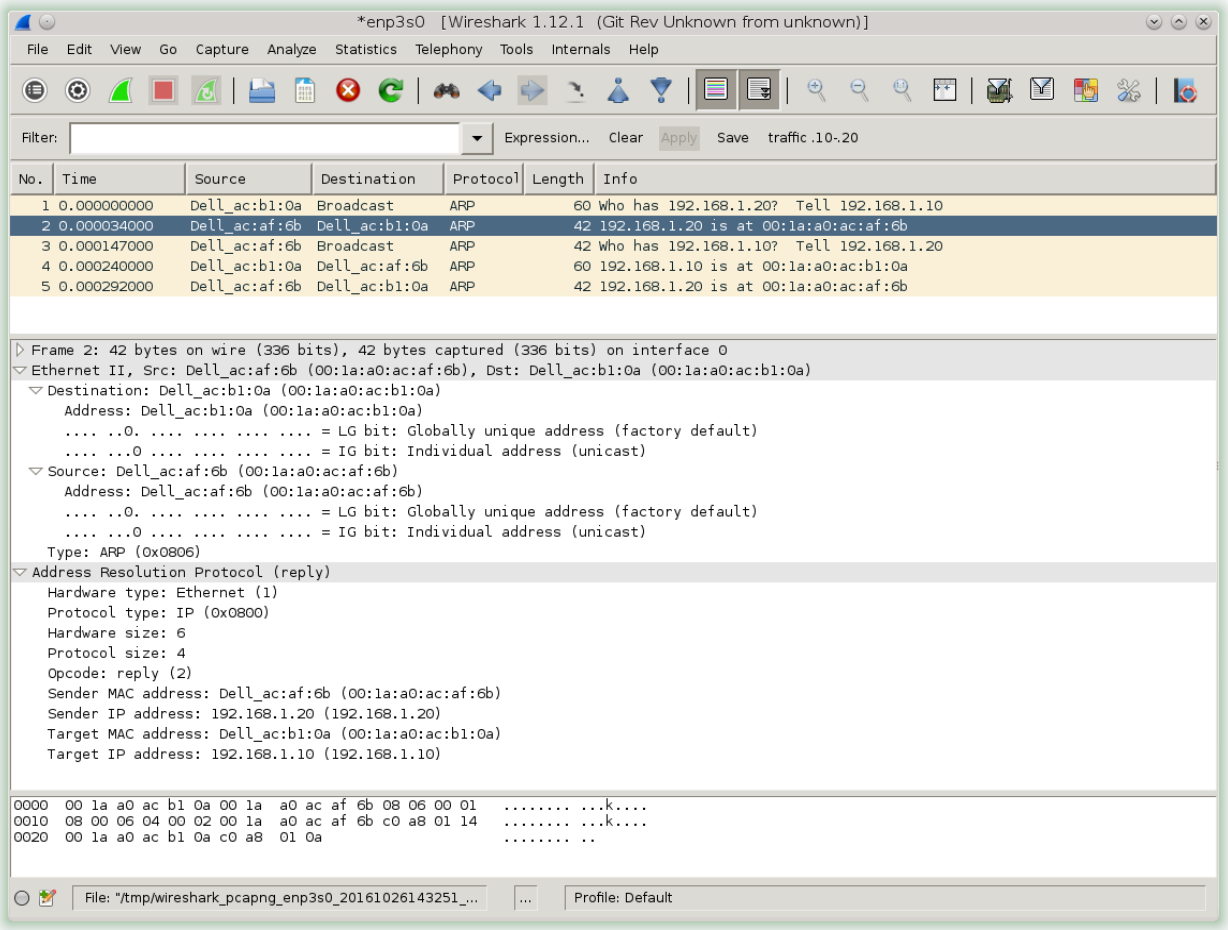
```
ARP request detected.

ARP reply detected.

Saving IP & MAC address pair to cache... DONE
>> IP saved = 192.168.1.10
>> MAC saved = 00:1a:a0:ac:b1:0a

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

ARP reply has been sent. END
```

Network Capture – IP-MAC not in cache

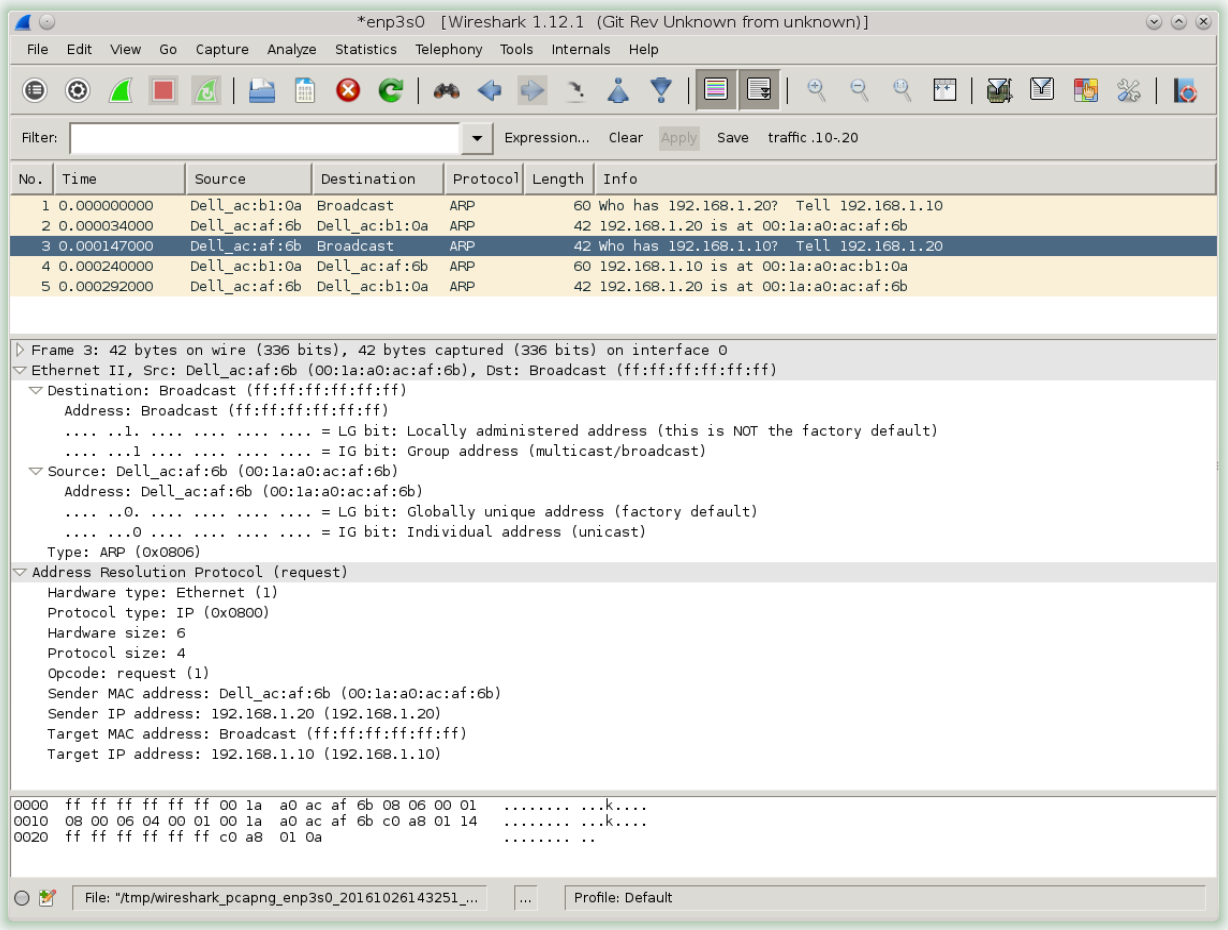
ARP Request from PC
Sender's MAC Address = 00:1A:A0:AC:B1:0A
Sender's IP Address = 192.168.1.10
Target MAC Address = FF:FF:FF:FF:FF:FF
Target IP Address = 192.168.1.20



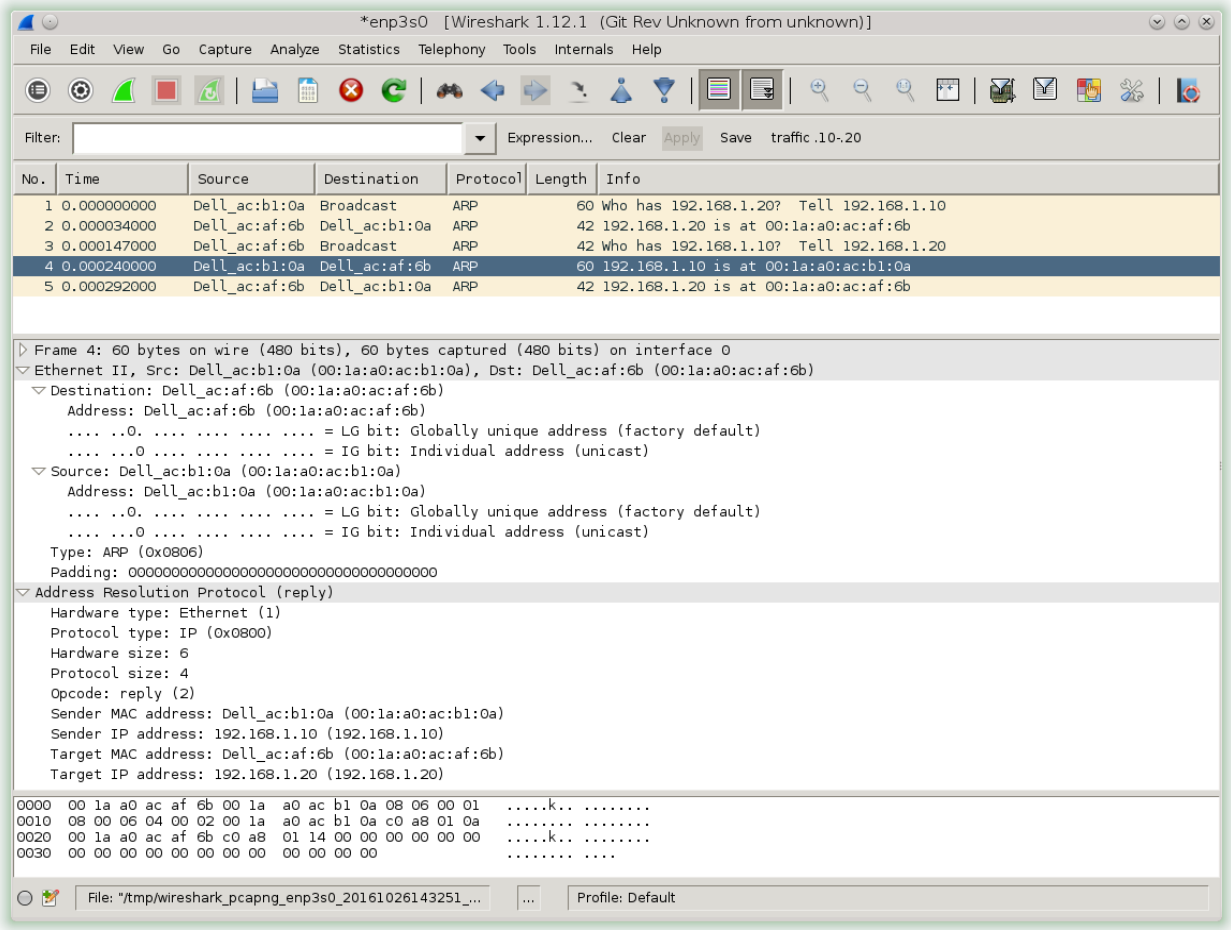
ARP Reply from PC
Sender's MAC Address = 00:1A:A0:AC:AF:6B
Sender's IP Address = 192.168.1.20
Target MAC Address = 00:1A:A0:AC:B1:0A
Target IP Address = 192.168.1.10



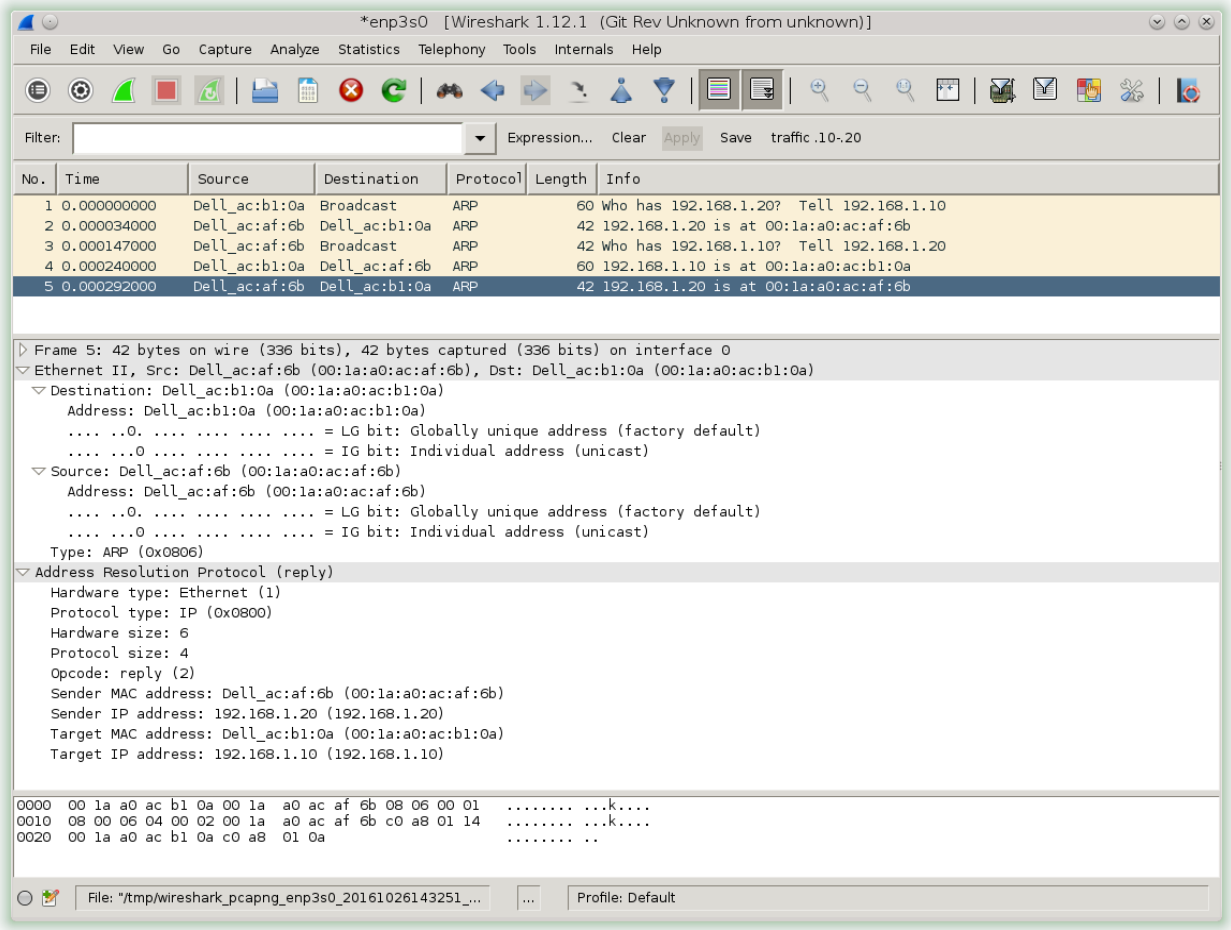
ARP Request from Code (3.2.1)
Sender's MAC Address = 00:1A:A0:AC:AF:6B
Sender's IP Address = 192.168.1.20
Target MAC Address = FF:FF:FF:FF:FF:FF
Target IP Address = 192.168.1.10



ARP Reply from Code (3.2.2)
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



ARP Reply from Code (3.2.3)
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

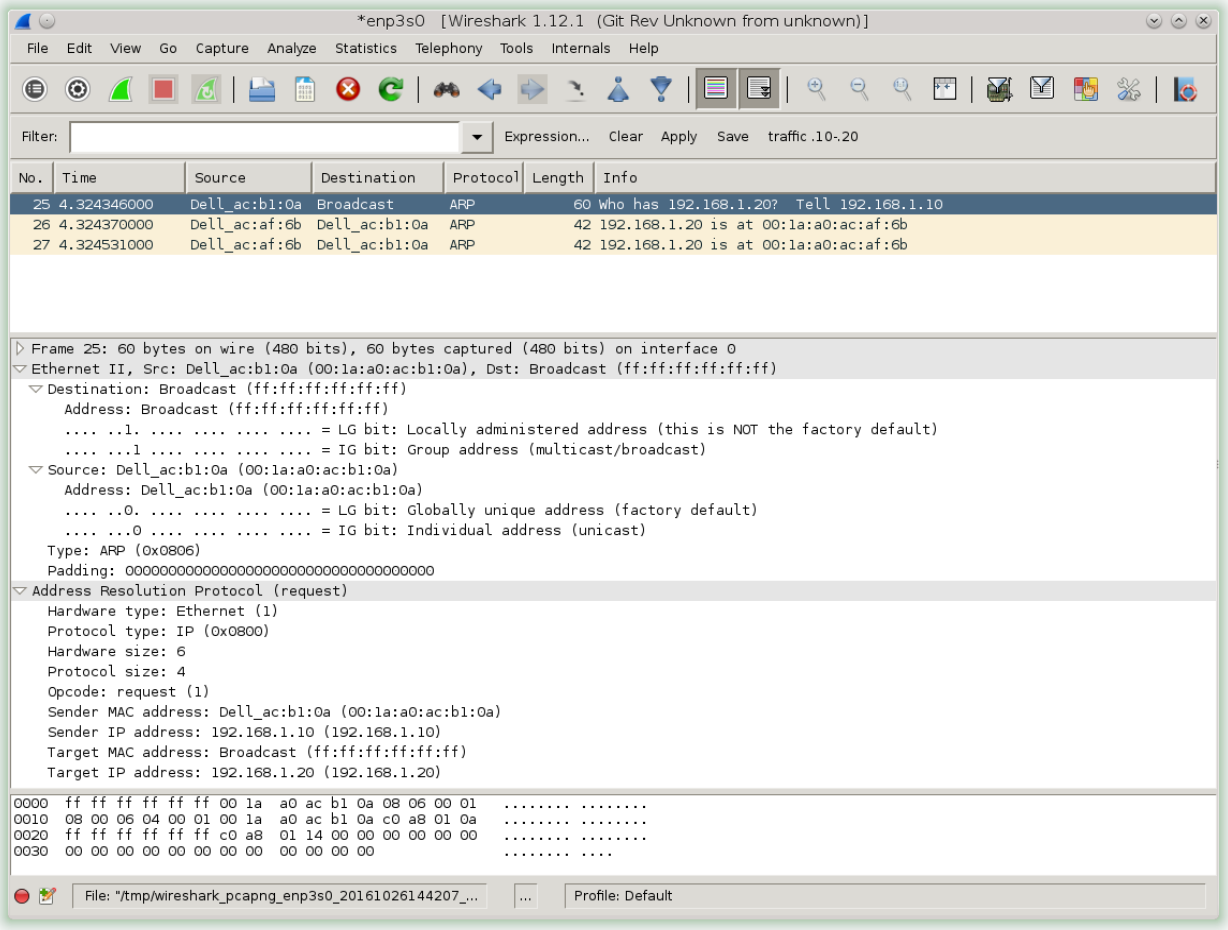


Program Output – IP-MAC is in cache (see code in Appendix 4.2)

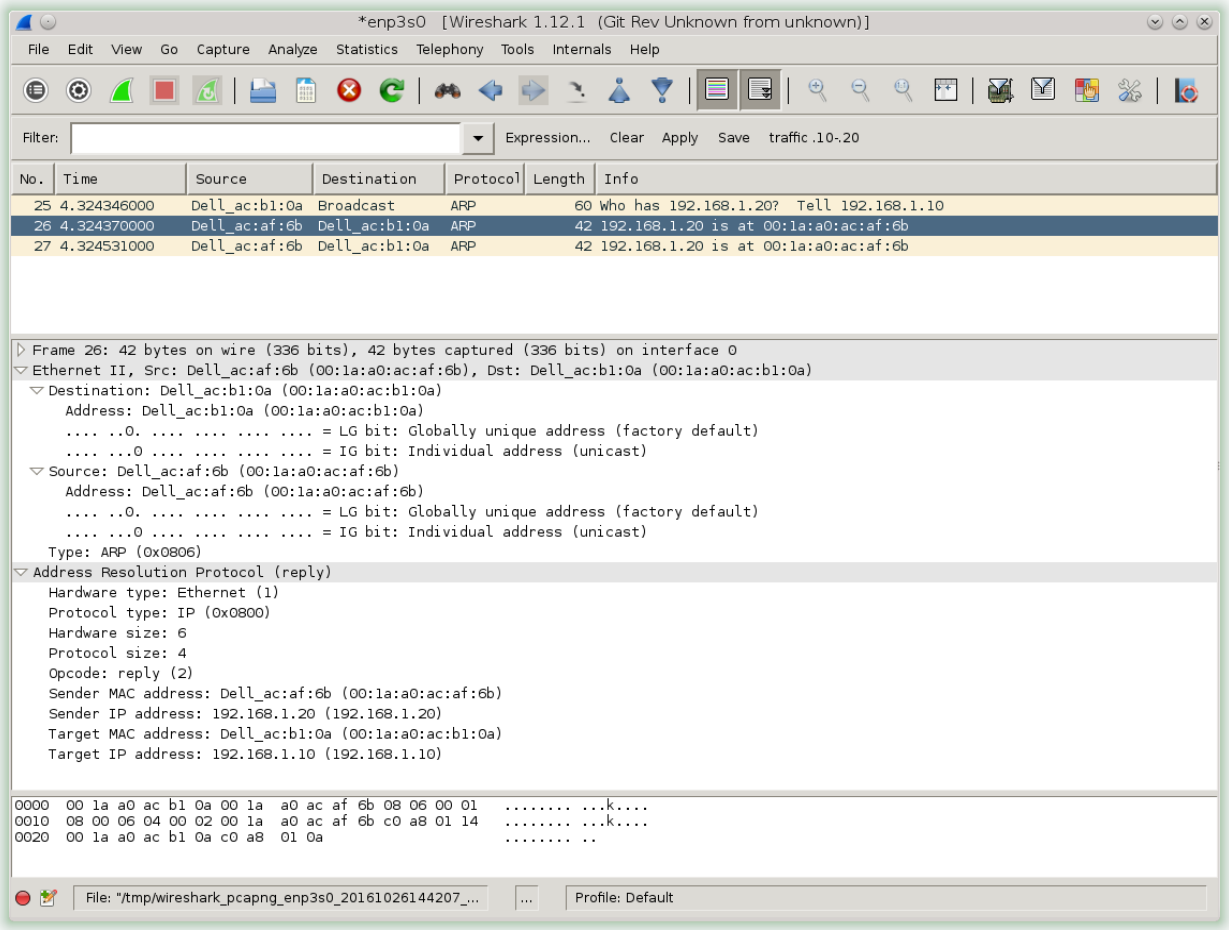
```
Searching target IP address in IP-MAC table... FOUND! Sending ARP reply...

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

ARP reply has been sent. END
```

Network Capture – IP-MAC is in cache

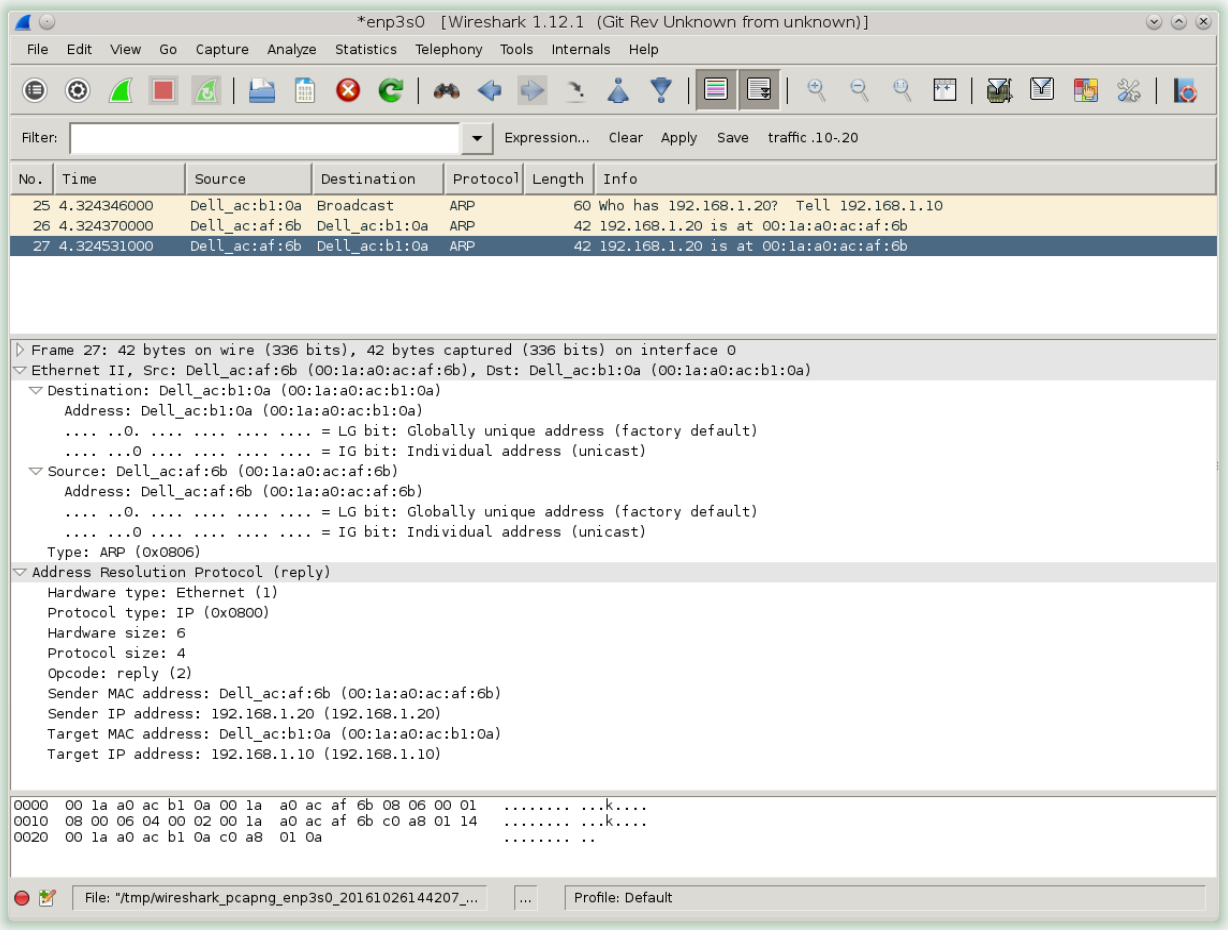
ARP Request from PC
Sender's MAC Address = 00:1A:A0:AC:B1:0A
Sender's IP Address = 192.168.1.10
Target MAC Address = FF:FF:FF:FF:FF:FF
Target IP Address = 192.168.1.20



ARP 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



ARP Reply 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:B1:0A
Target IP Address = 192.168.1.10



4. Appendix

4.1 Program Code – Procedures 1-3

```
#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
```

```
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 };

ether_frame frame;
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];
}

void print_frame()
{
   printf("Sender's MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n",
      sender_mac[0],sender_mac[1],sender_mac[2],sender_mac[3],sender_mac[4],sender_mac[5]);
   printf("Sender's IP Address = %d.%d.%d.%d\n",
         sender_ip[0],sender_ip[1],sender_ip[2],sender_ip[3]);
   printf("Target MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n",
         target_mac[0],target_mac[1],target_mac[2],target_mac[3],target_mac[4],target_mac[5]);
   printf("Target IP Address = %d.%d.%d.%d\n",
         target_ip[0],target_ip[1],target_ip[2],target_ip[3]);
   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 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 ARP frame arrives
//
void *arp_protocol_loop(void *arg)
{
   octet buf[1500];
   event_kind event;
   bool request;
   bool reply;

   /* 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
*/

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?
         {
            printf("ARP request detected.\n\n");
               request = true; // Yes it is a request.
         }
         else if ( buf[arp_byte] == 2 ) // Is this a reply?
            {
               printf("ARP reply detected.\n\n");
                     reply = true; // Yes it is a reply.
            }
                  else
                     printf("Opcode ERROR!\n\n");
      }
      if ( arp_byte == 24 ) // Detect the target IP address
      {
         printf("Target IP address detected.\n\n");
                  if ( request == true ) // Send ARP reply to ARP request
                  {
                        printf("ARP request received.\n\n");
                        request = 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("Target IP address matches.\n\n");
                        // Create and send the ethernet frame containing ARP reply payload.
                        printf("Creating ethernet frame containing ARP reply payload...\n\n");
                  //// Set ARP reply destination MAC address equal to ARP sender's MAC address.
                              //// buf_key: 08 to 13 = Sender's MAC Address
                              //>> frame.dst_mac = buf[8:13]
                              frame.dst_mac[0] = buf[8];
                              frame.dst_mac[1] = buf[9];
                              frame.dst_mac[2] = buf[10];
                              frame.dst_mac[3] = buf[11];
                              frame.dst_mac[4] = buf[12];
                              frame.dst_mac[5] = buf[13];
                  //// Set ARP reply source MAC address equal to current machine's MAC address.
                              //>> frame.src_mac = net.get_mac()
                              frame.src_mac[0] = net.get_mac()[0];
                              frame.src_mac[1] = net.get_mac()[1];
                              frame.src_mac[2] = net.get_mac()[2];
                              frame.src_mac[3] = net.get_mac()[3];
                              frame.src_mac[4] = net.get_mac()[4];
                              frame.src_mac[5] = net.get_mac()[5];
```

```
                                //>> frame.prot = { 0x08, 0x06 }
                                frame.prot[0] = 0x08;
                                frame.prot[1] = 0x06;
                                //// Create the ARP reply payload.
                                opcode[0] = 0;
                                opcode[1] = 2; // 2=Reply
                                //>> 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];
                                //// buf_key: 24 to 27 = Target IP Address
                                //>> sender_ip = buf[24:27]
                                sender_ip[0] = buf[24];
                                sender_ip[1] = buf[25];
                                sender_ip[2] = buf[26];
                                sender_ip[3] = buf[27];
                                //>> 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];
                                //// buf_key: 14 to 17 = Sender's IP Address
                                //>> target_ip = buf[14:17]
                                target_ip[0] = buf[14];
                                target_ip[1] = buf[15];
                                target_ip[2] = buf[16];
                                target_ip[3] = buf[17];
                                arp_frame();
                                print_frame();
                                // Send the ethernet frame containing ARP reply payload.
                                net.send_frame(&frame,42);
                                printf("ARP reply has been sent. END\n\n");
                        }
                }
                else
                {
                        printf("ARP reply received. END\n\n");
                        reply = false;
                }
        }
    }
  }
}

//
// if you're going to have pthreads, you'll need some thread descriptors
//
pthread_t loop_thread, arp_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"); // Ethernet port of lab room computer.
   pthread_create(&loop_thread,NULL,protocol_loop,NULL);
   pthread_create(&arp_thread,NULL,arp_protocol_loop,NULL);
   for ( ; ; )
      sleep(1);
}
```

4.2 Program Code – Procedures 4-5

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

#include <vector>
using namespace std;

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
};

struct ipmac
{
   octet ip[4];      // IP address
   octet mac[6];     // MAC address
};
ipmac IPMAC;
vector<ipmac> ipmac_table;
int ipmac_table_size = 0;
octet ipmac_IP[4];
octet ipmac_MAC[6];

void add_ipmac(octet ip[4], octet mac[6])
{
       IPMAC.ip[0] = ip[0];
       IPMAC.ip[1] = ip[1];
       IPMAC.ip[2] = ip[2];
       IPMAC.ip[3] = ip[3];
       IPMAC.mac[0] = mac[0];
       IPMAC.mac[1] = mac[1];
       IPMAC.mac[2] = mac[2];
```

```
	IPMAC.mac[3] = mac[3];
	IPMAC.mac[4] = mac[4];
	IPMAC.mac[5] = mac[5];
	ipmac_table.push_back(IPMAC);
	ipmac_table_size++;
}

bool find_ipmac(octet ip[4])
{
	bool found;
	printf("Searching target IP address in IP-MAC table... ");
	if ( ipmac_table_size == 0 )
	{
		printf("NOT FOUND. Sending ARP request...\n\n");
		found = false;
	}
	else
	{
		for (int i = 0; i < ipmac_table_size; i++)
		{
			if (ipmac_table[i].ip[0] == ip[0] &&
				ipmac_table[i].ip[1] == ip[1] &&
				ipmac_table[i].ip[2] == ip[2] &&
				ipmac_table[i].ip[3] == ip[3])
			{
				printf("FOUND! Sending ARP reply...\n\n");
				found = true;
				//>> ipmac_IP = ipmac_table[i].ip
				ipmac_IP[0] = ipmac_table[i].ip[0];
				ipmac_IP[1] = ipmac_table[i].ip[1];
				ipmac_IP[2] = ipmac_table[i].ip[2];
				ipmac_IP[3] = ipmac_table[i].ip[3];
				//>> ipmac_MAC = ipmac_table[i].mac
				ipmac_MAC[0] = ipmac_table[i].mac[0];
				ipmac_MAC[1] = ipmac_table[i].mac[1];
				ipmac_MAC[2] = ipmac_table[i].mac[2];
				ipmac_MAC[3] = ipmac_table[i].mac[3];
				ipmac_MAC[4] = ipmac_table[i].mac[4];
				ipmac_MAC[5] = ipmac_table[i].mac[5];
				i = ipmac_table_size;
			}
			else if (ipmac_table[i].ip[0] != ip[0] &&
					ipmac_table[i].ip[1] != ip[1] &&
					ipmac_table[i].ip[2] != ip[2] &&
					ipmac_table[i].ip[3] != ip[3] &&
					i == ipmac_table_size-1)
			{
				printf("NOT FOUND. Sending ARP request...\n\n");
				found = false;
			}
			else
			{
				printf("ERROR!\n\n");
			}
		}
	}
	return(found);
```

```
}

octet my_ip[4] = { 192, 168, 1, 20 };
octet my_mac[6];
octet ip_target[4] = { 192, 168, 1, 10 };
octet mac_target[6] = { 0x00, 0x1A, 0xA0, 0xAC, 0xB1, 0x0A };
octet mac_broadcast[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };

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];
}

void print_frame()
{
   printf("Sender's MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n",
      sender_mac[0],sender_mac[1],sender_mac[2],sender_mac[3],sender_mac[4],sender_mac[5]);
   printf("Sender's IP Address = %d.%d.%d.%d\n",
         sender_ip[0],sender_ip[1],sender_ip[2],sender_ip[3]);
   printf("Target MAC Address = %02x:%02x:%02x:%02x:%02x:%02x\n",
         target_mac[0],target_mac[1],target_mac[2],target_mac[3],target_mac[4],target_mac[5]);
   printf("Target IP Address = %d.%d.%d.%d\n",
         target_ip[0],target_ip[1],target_ip[2],target_ip[3]);
   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 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 ARP frame arrives
//
void *arp_protocol_loop(void *arg)
{
   octet buf[1500];
   event_kind event;
   bool request;
   bool reply;
   bool ip_found;
   bool skip = false;

   /* 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];

   while ( 1 )
   {
      arp_queue.recv(&event, buf, sizeof(buf));
         add_ipmac(ip_target,mac_target); //<< Enable this line for "IP-MAC exists".
         if ( skip != true )
            ip_found = find_ipmac(ip_target);
         if ( ip_found == true && skip != true ) // Send ARP reply
         {
            // Set ARP reply destination MAC address equal to ARP sender's MAC address.
               // Set ARP reply source MAC address equal to current machine's MAC address.
               ethernet_frame(ipmac_MAC,my_mac,ARP_type);
               // Create the ARP reply payload.
               opcode[0] = 0;
               opcode[1] = 2; // 2=Reply
               //>> 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];
            //>> target_ip = ipmac_IP
            target_ip[0] = ipmac_IP[0];
            target_ip[1] = ipmac_IP[1];
            target_ip[2] = ipmac_IP[2];
            target_ip[3] = ipmac_IP[3];
            arp_frame();
            print_frame();
            // Send the ethernet frame containing ARP reply payload.
            net.send_frame(&frame,42);
            printf("ARP reply has been sent. END\n\n");
            goto finish;
    }
       else if ( ip_found == false && skip != true ) // Send ARP request
       {
            // 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];
            //>> target_ip = ip_target
            target_ip[0] = ip_target[0];
            target_ip[1] = ip_target[1];
            target_ip[2] = ip_target[2];
            target_ip[3] = ip_target[3];
            arp_frame();
            print_frame();
            // Send the ethernet frame containing ARP request payload.
```

```
            net.send_frame(&frame,42);
            printf("ARP request has been sent.\n\n");
            skip = true;
      }
      else
         ;

   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?
         {
            printf("ARP request detected.\n\n");
               request = true; // Yes it is a request.
         }
         else if ( buf[arp_byte] == 2 ) // Is this a reply?
            {
               printf("ARP reply detected.\n\n");
                     reply = true; // Yes it is a reply.
                     printf("Saving IP & MAC address pair to cache... ");
                     //>> IPMAC.ip = buf[24:27]
                     IPMAC.ip[0] = buf[24];
                     IPMAC.ip[1] = buf[25];
                     IPMAC.ip[2] = buf[26];
                     IPMAC.ip[3] = buf[27];
                     //>> IPMAC.mac = buf[18:23]
                     IPMAC.mac[0] = buf[18];
                     IPMAC.mac[1] = buf[19];
                     IPMAC.mac[2] = buf[20];
                     IPMAC.mac[3] = buf[21];
                     IPMAC.mac[4] = buf[22];
                     IPMAC.mac[5] = buf[23];
                     ipmac_table.push_back(IPMAC); // Add IP-MAC pair to cache.
                     ipmac_table_size++; // Update IP-MAC cache size.
                     printf("DONE\n");
                     printf(">> IP saved = %d.%d.%d.%d\n",
                         ipmac_table[ipmac_table_size-1].ip[0],
                         ipmac_table[ipmac_table_size-1].ip[1],
                         ipmac_table[ipmac_table_size-1].ip[2],
                         ipmac_table[ipmac_table_size-1].ip[3]);
                     printf(">> MAC saved = %02x:%02x:%02x:%02x:%02x:%02x\n\n",
                         ipmac_table[ipmac_table_size-1].mac[0],
                         ipmac_table[ipmac_table_size-1].mac[1],
                         ipmac_table[ipmac_table_size-1].mac[2],
                         ipmac_table[ipmac_table_size-1].mac[3],
                         ipmac_table[ipmac_table_size-1].mac[4],
                         ipmac_table[ipmac_table_size-1].mac[5]);
                     // Set ARP reply destination MAC address equal to ARP sender's MAC address.
                     // Set ARP reply source MAC address equal to current machine's MAC address.
                     ethernet_frame(ipmac_table[ipmac_table_size-1].mac,my_mac,ARP_type);
                     //// Create the ARP reply payload.
                     opcode[0] = 0;
                     opcode[1] = 2; // 2=Reply
                     //>> 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];
                        //>> target_ip = ipmac_table[ipmac_table_size-1].ip
                        target_ip[0] = ipmac_table[ipmac_table_size-1].ip[0];
                        target_ip[1] = ipmac_table[ipmac_table_size-1].ip[1];
                        target_ip[2] = ipmac_table[ipmac_table_size-1].ip[2];
                        target_ip[3] = ipmac_table[ipmac_table_size-1].ip[3];
                        arp_frame();
                        print_frame();
                        // Send the ethernet frame containing ARP reply payload.
                        net.send_frame(&frame,42);
                        printf("ARP reply has been sent. END\n\n");
                        goto finish;
                }
                     else
                        printf("Opcode ERROR!\n\n");
            }
        }
    }
finish:;
}

//
// if you're going to have pthreads, you'll need some thread descriptors
//
pthread_t loop_thread, arp_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"); // Ethernet port of lab room computer.
    pthread_create(&loop_thread,NULL,protocol_loop,NULL);
    pthread_create(&arp_thread,NULL,arp_protocol_loop,NULL);
    for ( ; ; )
        sleep(1);
}
```