Project – Part 1

1. Objective

1. Be familiar with Wireshark and Linux OS.
2. Know how to capture traffic and C programming under Linux OS environment.
3. Know how to customize Wireshark filter to capture and analyze the data frames.

2. Overview

During the course of this semester, students will implement some of the TCP/IP protocol stack using Linux. Linux allows a user to configure a socket in packet mode so that they can send and receive raw packets over the Ethernet. Students will be able to build up their own network stack without interfering with the normal operation of the network connection. In order to avoid chaos, each student needs an IP address, while has the following format:
192.168.1.xxx
Use the command "`ifconfig`" to get your IP address.

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

- Write code to capture and print the bytes of IP and ARP request/reply frames.

Program Output (see code in Appendix 4.1)

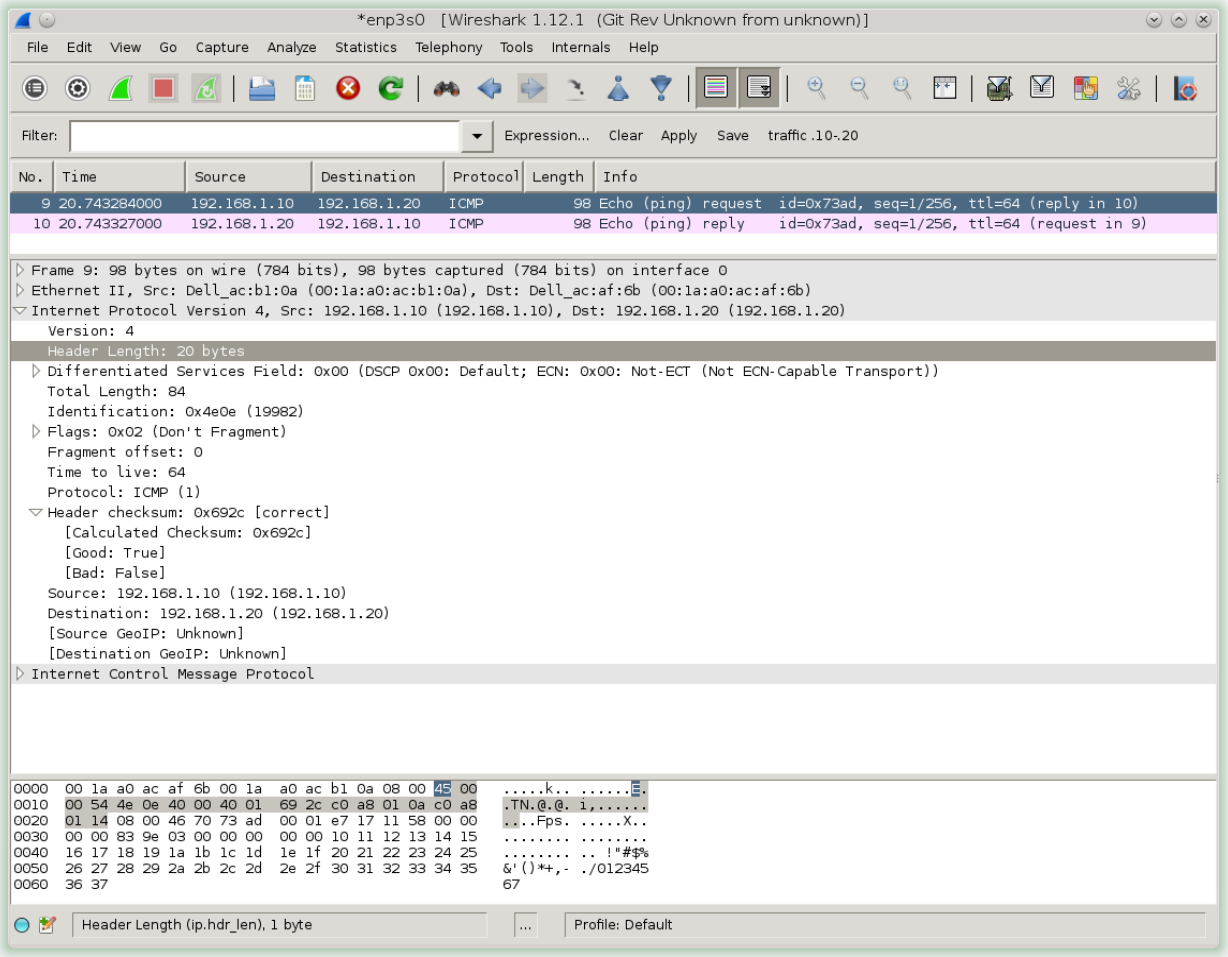
```
IP detected.
45 00 00 54 4e 0e 40 00 40 01 69 2c c0 a8 01 0a c0 a8 01 14 08 00
46 70 73 ad 00 01 e7 17 11 58 00 00 00 00 83 9e 03 00 00 00
IP detected.
45 00 00 54 61 bd 00 00 40 01 95 7d c0 a8 01 14 c0 a8 01 0a 00 00
4e 70 73 ad 00 01 e7 17 11 58 00 00 00 00 83 9e 03 00 00 00
```

Program Output (see code in Appendix 4.1)

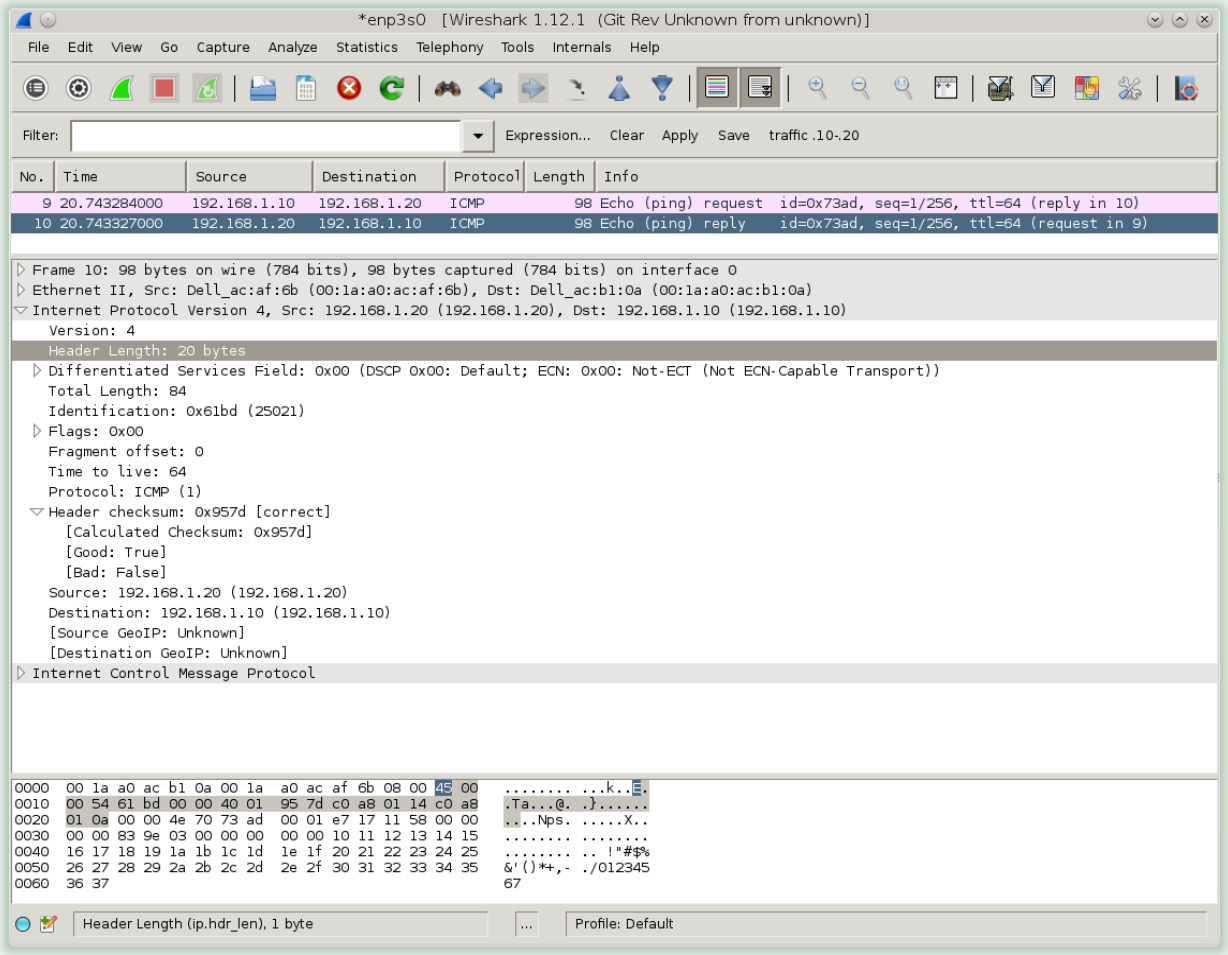
```
ARP detected.
00 01 08 00 06 04 00 01 00 1a a0 ac b1 0a c0 a8 01 0a ff ff ff ff
ff ff c0 a8 01 14 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ARP detected.
00 01 08 00 06 04 00 02 00 1a a0 ac af 6b c0 a8 01 14 00 1a a0 ac
b1 0a c0 a8 01 0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Network Capture

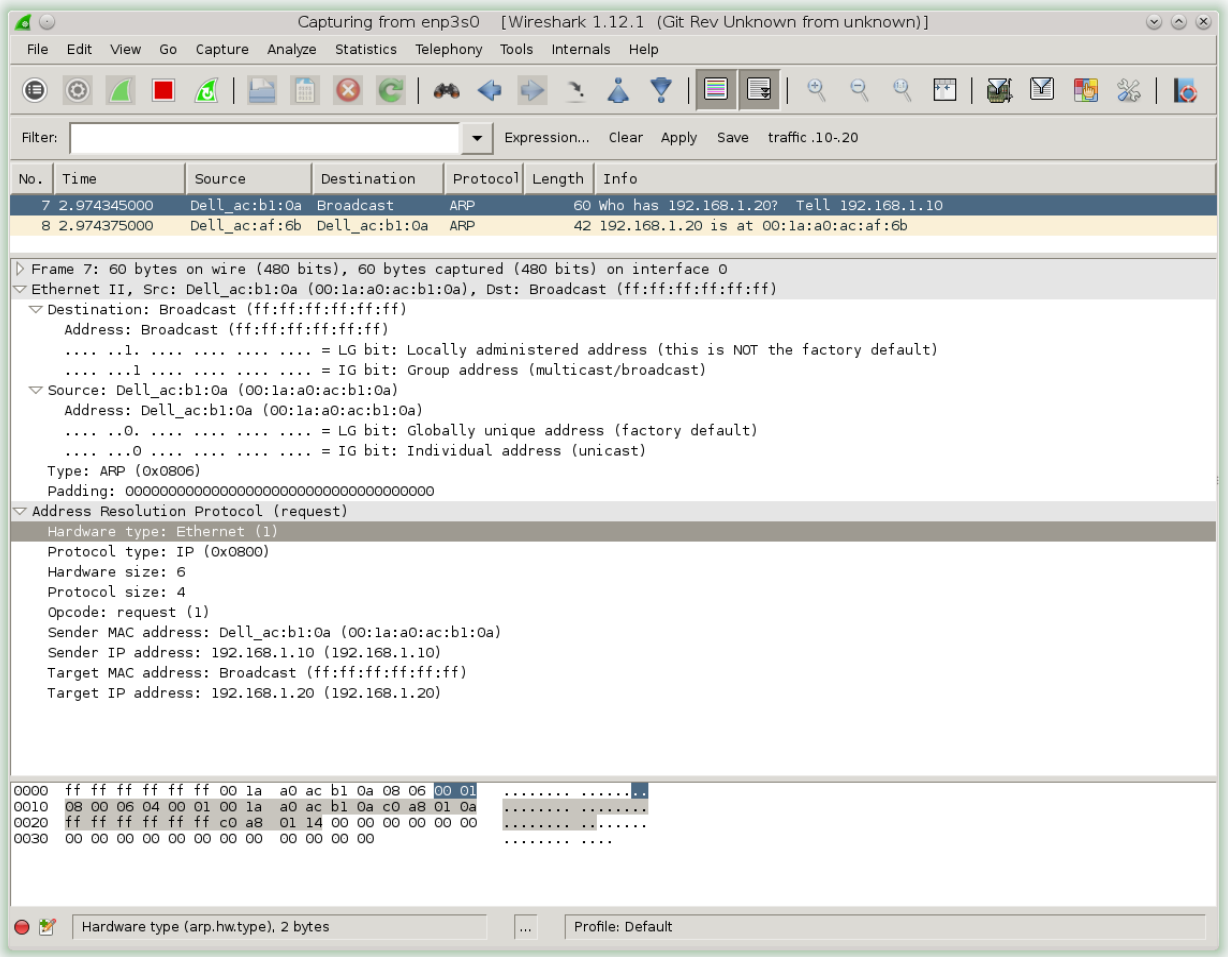
IP Request from PC
Source IP Address = 192.168.1.10
Destination IP Address = 192.168.1.20



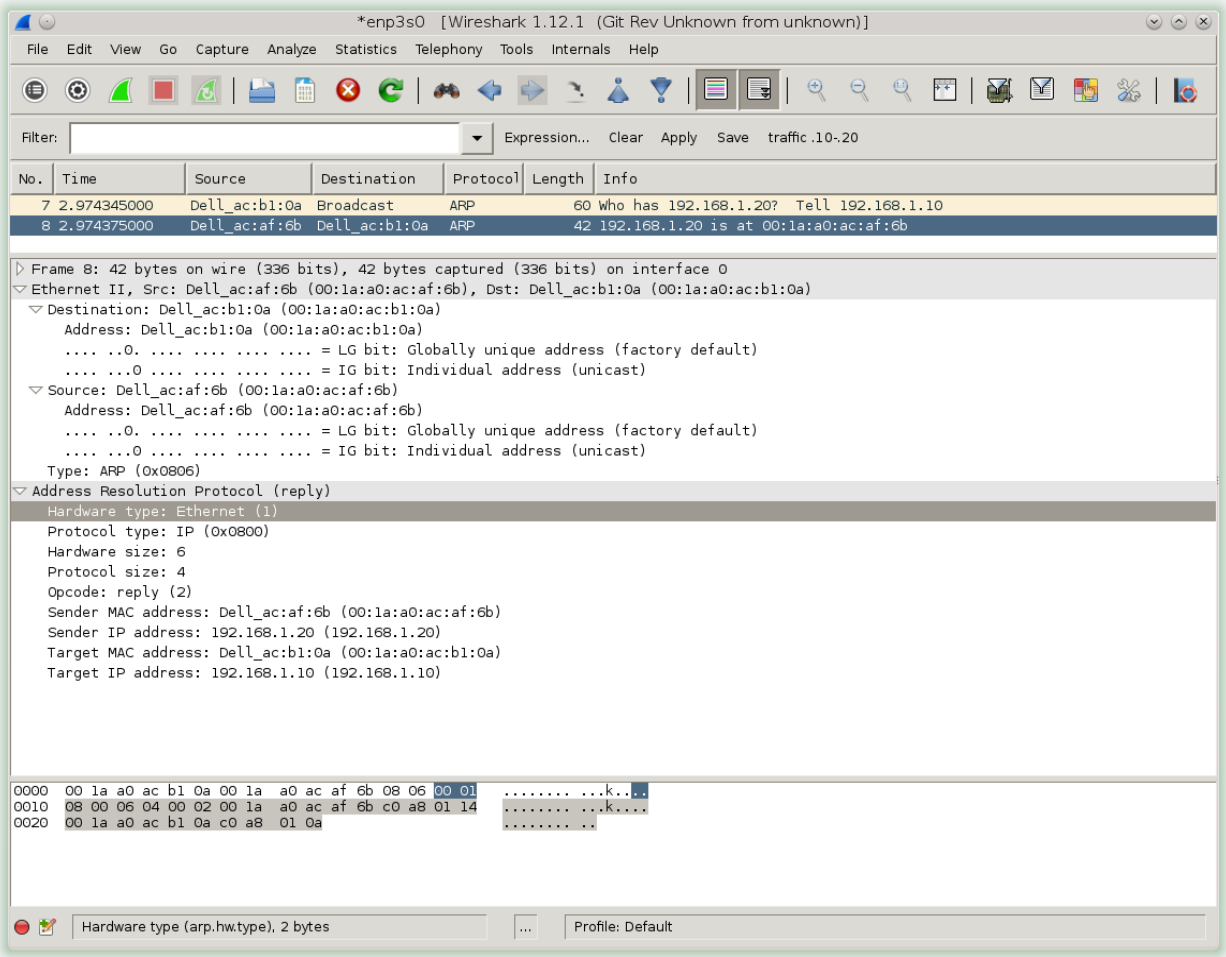
IP Reply from PC
Source IP Address = 192.168.1.20
Destination IP Address = 192.168.1.10



ARP Request from PC
Source MAC Address = 00:1A:A0:AC:B1:0A
Source IP Address = 192.168.1.10
Destination MAC Address = FF:FF:FF:FF:FF:FF
Destination IP Address = 192.168.1.20



ARP Reply from PC
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



---

4. Appendix

4.1 Program Code

```
#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
};

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

//
// Toy function to print something interesting when an IP frame arrives
//
void *ip_protocol_loop(void *arg)
{
   octet buf[1500];
   event_kind event;
   int ip_byte = 0;

   while ( 1 )
   {
      ip_queue.recv(&event, buf, sizeof(buf));
         printf("IP detected.\n");
      for ( ip_byte = 0; ip_byte < 42; ip_byte++) /* Read first 42 IP bytes */
      {
            printf("%02x ",buf[ip_byte]); /* Print IP byte */
            if ( ip_byte == 21 || ip_byte == 41 )
               printf("\n"); /* Add new line for the 22nd and 42nd IP byte. */
      }
   }
}

//
```

```
// Toy function to print something interesting when an ARP frame arrives
//
void *arp_protocol_loop(void *arg)
{
   octet buf[1500];
   event_kind event;
   int arp_byte = 0;

   while ( 1 )
   {
      arp_queue.recv(&event, buf, sizeof(buf));
      printf("ARP detected.\n");
         for ( arp_byte = 0; arp_byte < 42; arp_byte++) /* Read first 42 ARP bytes */
      {
            printf("%02x ",buf[arp_byte]); /* Print ARP byte */
            if ( arp_byte == 21 || arp_byte == 41 )
               printf("\n"); /* Add new line for the 22nd and 42nd ARP byte. */
      }
   }
}

//
// 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);
   pthread_create(&ip_thread,NULL,ip_protocol_loop,NULL);
   for ( ; ; )
      sleep(1);
}
```