Project 1 Report
ECE 5600

Nathan Tipton
A01207112
Partner: Erik Sargent

Sept 25, 2017

1 Objective

The purpose of this project is to familiarize ourselves with Wireshark and Linux OS. We will do this by capturing network traffic with a program written in C on the Linux system. We will also capture data frames from the network traffic using the program Wireshark. These data frames will also be analyzed with Wireshark.

2 Results

Frames on the network are captured by our program and stored in a frame buffer. The buffer includes the destination and source MAC addresses, the protocol and payload data. The protocol is used to to determine the payload data type, ARP or IP. If a frame is less than 42 bytes our program discards it and looks at the next frame. Using another computer we pinged the computer that was running our program. You can see the packets received and the program output in figure 1.

Wireshark was used to verify that our program was properly monitoring network traffic. In figure 1, our ARP packets were labeled in order to more easily find them in the program output. The program's message is output after the ARP packet.

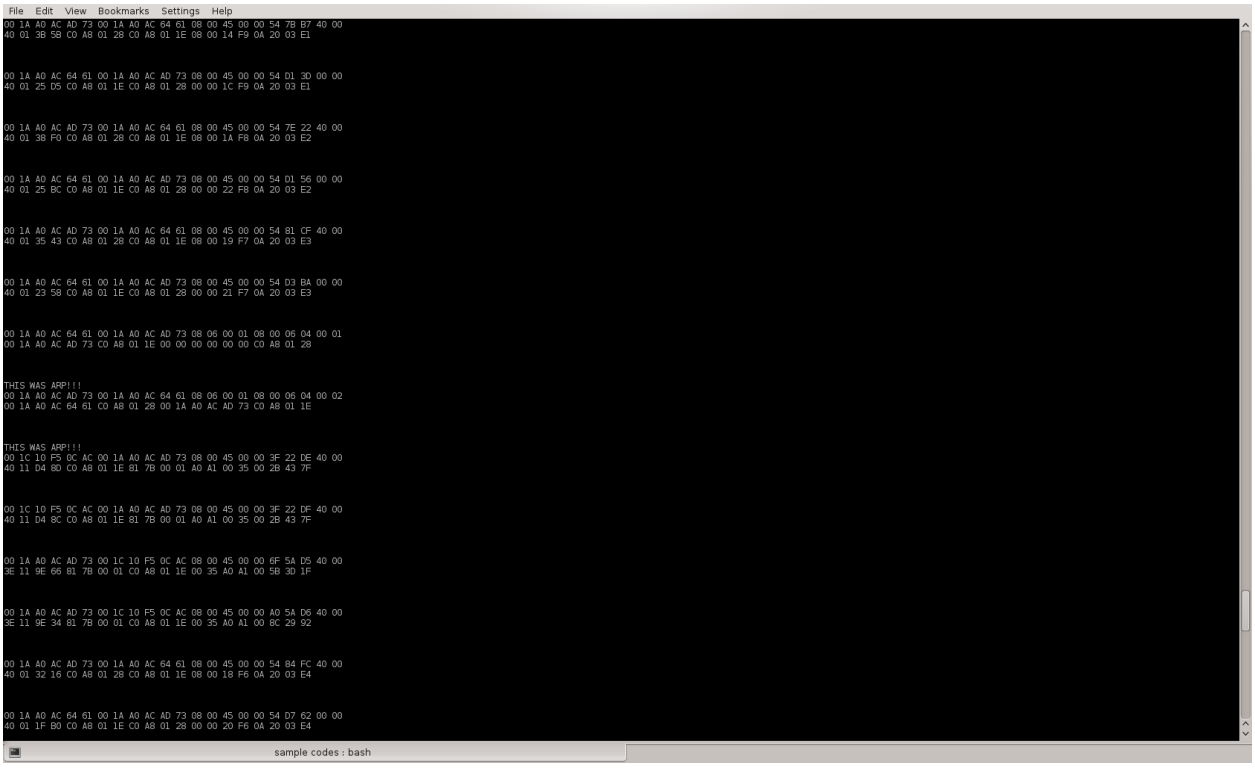

Figure 1: Data packets captured by our program

Figures 2 an 3 show the data captured by Wireshark. The first is the request from the second computer which is pinging the computer running our program. The second figure shows the response giving the MAC address of the program computer.

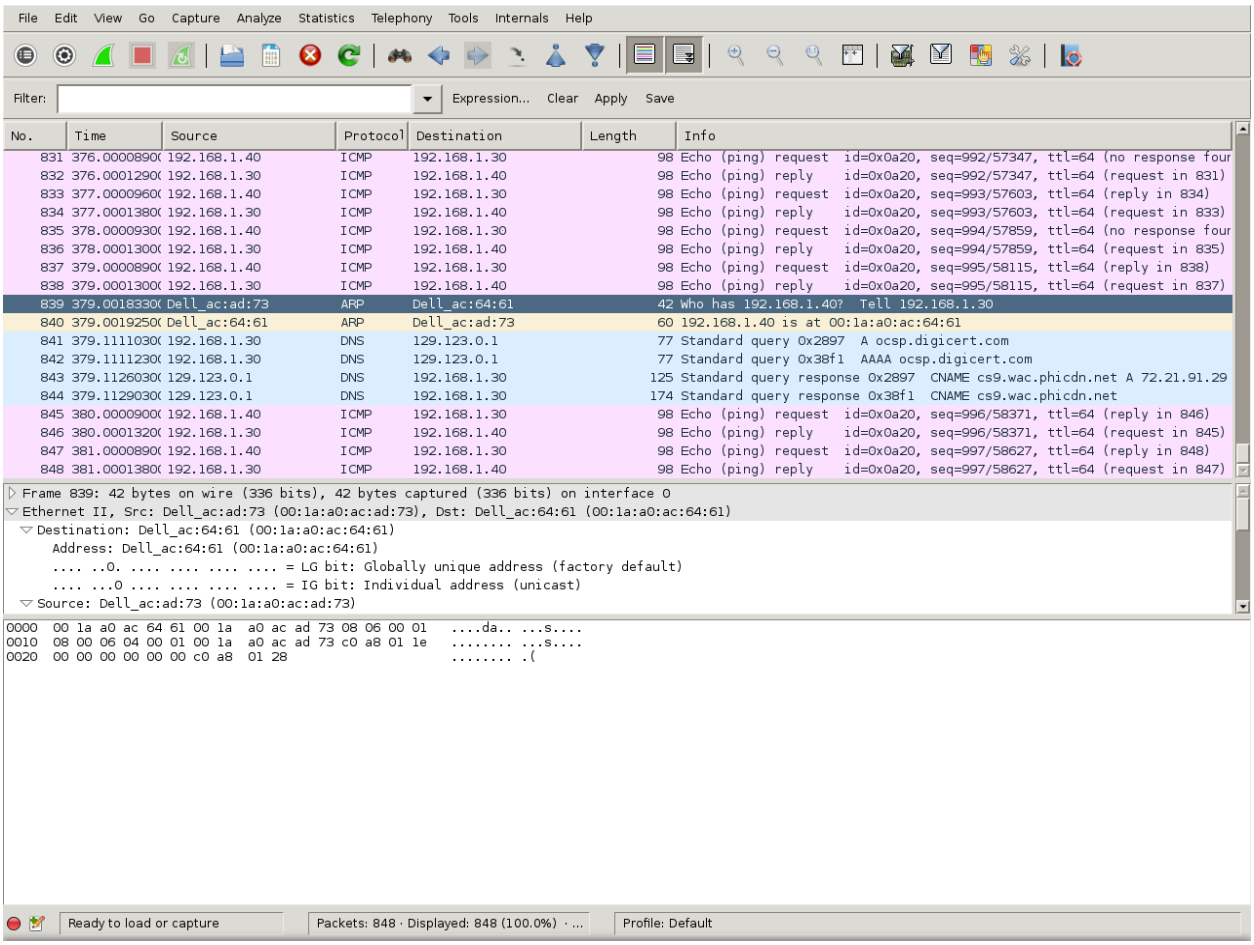

Figure 2: Wireshark ARP Request

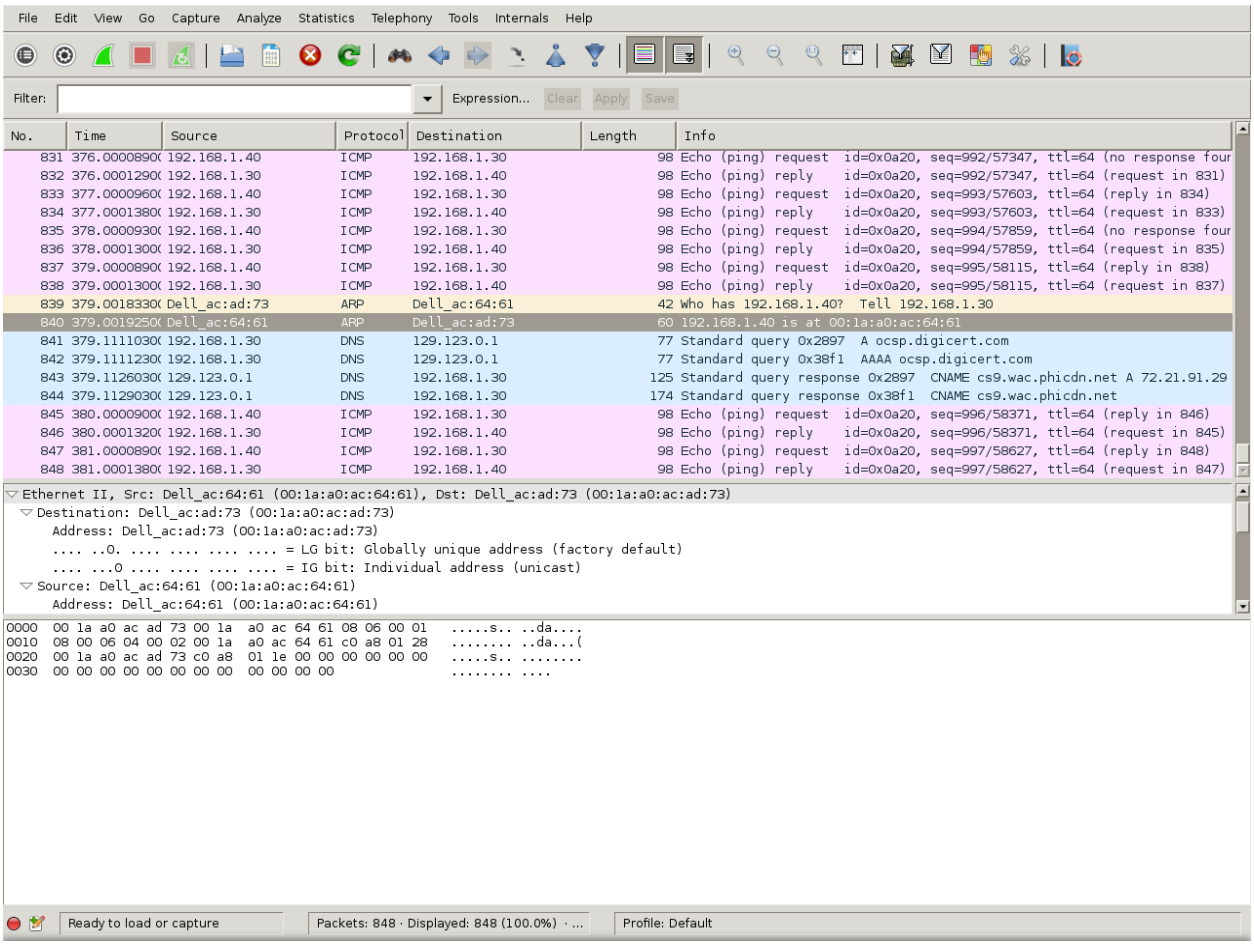

Figure 3: Wireshark ARP Response

Comparing the outputs of our program with the frames captured by Wireshark shows that our program sucessfully captured the data frames on the network.

output.txt

```
From Wireshark:
ARP Request:
00 1a a0 ac 64 61 00 1a  a0 ac ad 73 08 06 00 01
08 00 06 04 00 01 00 1a  a0 ac ad 73 c0 a8 01 1e
00 00 00 00 00 00 c0 a8  01 28

ARP Response:
00 1a a0 ac ad 73 00 1a  a0 ac 64 61 08 06 00 01
08 00 06 04 00 02 00 1a  a0 ac 64 61 c0 a8 01 28
00 1a a0 ac ad 73 c0 a8  01 1e 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00


From our program:
ARP Request:
00 1A A0 AC 64 61 00 1A A0 AC AD 73 08 06 00 01 08 00 06 04 00 01
00 1A A0 AC AD 73 C0 A8 01 1E 00 00 00 00 00 00 C0 A8 01 28

ARP Response:
00 1A A0 AC AD 73 00 1A A0 AC 64 61 08 06 00 01 08 00 06 04 00 02
00 1A A0 AC 64 61 C0 A8 01 28 00 1A A0 AC AD 73 C0 A8 01 1E
```

3 Conclusion

ARP or Address Resolution Protocol is used to map IP addresses with data link layer addresses. We can see how it works from our captured frames. A packet is broadcast on the network to see who has the destination IP address as seen in Figure 2. The machine with the right IP address will respond with it's Ethernet address. We can see this in the response of figure 3. Through this project I have learned how to monitor and filter network traffic using wireshark. I have also learned how a specific machine on an ethernet network is found and communicated with. I learned how to implement these in the Linux OS.

4 Appendix

```
#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

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!

      for (int i = 0; i < 6; i++)
        printf("%02X ", buf.dst_mac[i]);
      for (int i = 0; i < 6; i++)
        printf("%02X ", buf.src_mac[i]);
      for (int i = 0; i < 2; i++)
        printf("%02X ", buf.prot[i]);
      for (int i = 0; i < 8; i++)
        printf("%02X ", buf.data[i]);
      printf("\n");
      for (int i = 8; i < 28; i++)
        printf("%02X ", buf.data[i]);
      printf("\n\n");

   }
}

//
// if you're going to have pthreads, you'll need some thread descriptors
```

```
//
pthread_t loop_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);
    for ( ; ; )
        sleep(1);
}
```