Completed with Lab Partner: Josh Lake

# Objective

To gain familiarity with programming Linux, the structure of Ethernet packets, and Wireshark.

# Structural Overview

One computer will ping another. The receiving end will run a program developed by the lab participants to dump the first 42 bytes of the packets to the console. These results will be compared with a data dump from Wireshark.

# Simulation

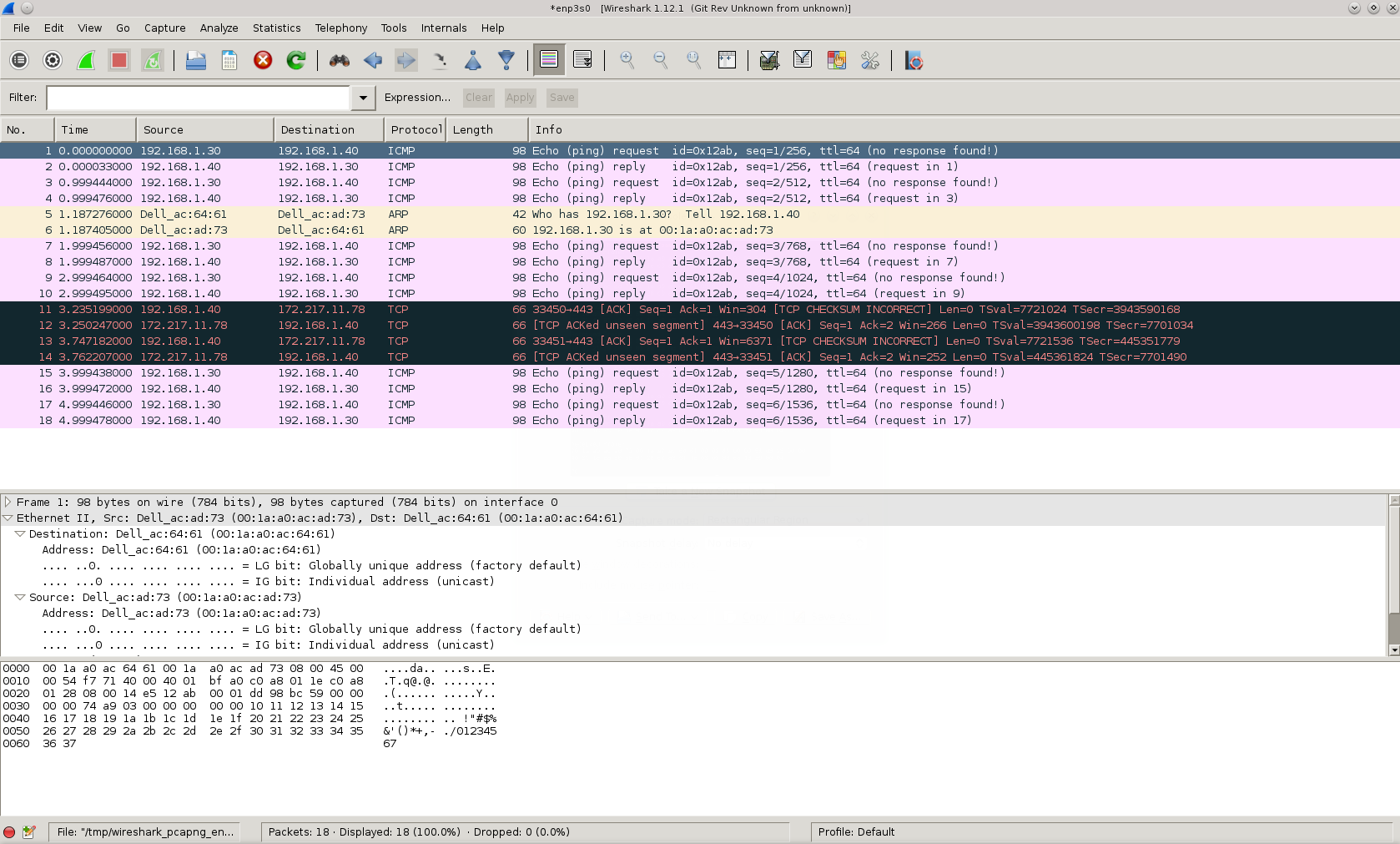


Figure 1 Wireshark Output

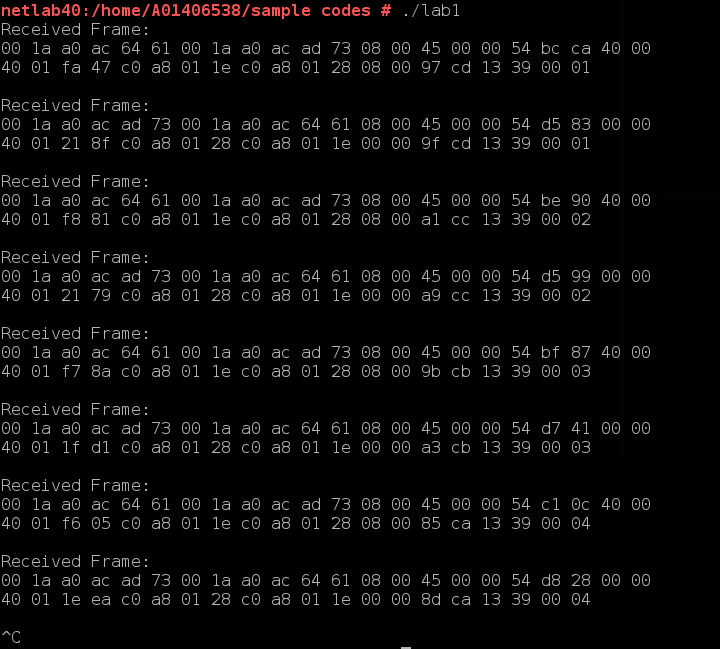


Figure 2 Program Output

# Results

It can be observed the Wireshark output and agree on the header. There was difficulty finding a complete match as it is believed there is a counter of some sort in part of the body of the ping request. However, the structure of the packets is the same, and it can be concluded that we are in fact seeing the ping requests and replies in the custom program.

# Makefile

lab1**:** lab1main.cpp frameio.o

g++ lab1main.cpp frameio.o -o lab1

frameio.o**:** frameio.cpp frameio.h

g++ frameio.cpp -c -o frameio.o

clean**:**

rm \*.o

rm lab1

# lab1main.cpp

#include "frameio.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.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

**};**

int main**()**

**{**

net**.**open\_net**(**"enp3s0"**);**

ether\_frame buf**;**

octet**\*** raw **=** **(**octet**\*)(&**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**:**

**case** 0x806**:**

printf**(**

"Received Frame: \n"

"%02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x \n"

"%02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x \n\n"**,**

raw**[**00**],** raw**[**01**],** raw**[**02**],** raw**[**03**],** raw**[**04**],** raw**[**05**],** raw**[**06**],** raw**[**07**],** raw**[**08**],** raw**[**09**],**

raw**[**10**],** raw**[**11**],** raw**[**12**],** raw**[**13**],** raw**[**14**],** raw**[**15**],** raw**[**16**],** raw**[**17**],** raw**[**18**],** raw**[**19**],**

raw**[**20**],** raw**[**21**],** raw**[**22**],** raw**[**23**],** raw**[**24**],** raw**[**25**],** raw**[**26**],** raw**[**27**],** raw**[**28**],** raw**[**29**],**

raw**[**30**],** raw**[**31**],** raw**[**32**],** raw**[**33**],** raw**[**34**],** raw**[**35**],** raw**[**36**],** raw**[**37**],** raw**[**38**],** raw**[**39**],**

raw**[**40**],** raw**[**41**],** raw**[**42**]**

**);**

**}**

**}**

**return** 0**;**

**}**

# frameio.h (code provided by instructor)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// frameio.h

//

// frameio.h and frameio.cpp provide convenient access to the ethernet

// using raw sockets. This provides the means to read and write frames

// directly to and from the ethernet interface.

//

// Before the frameio object can be used, you must specify which interface

// you are using. This is done via the member function open\_net(). For

// example, if you wish to communicate using inteface "eth1", you might

// write the code:

//

// frameio net;

//

// main(int argc, char \*argv[])

// {

// net.open\_net("eth1");

//

// After the net has been opened, the interface's 6-byte MAC address can be

// obtained by calling get\_mac(). Note that this function gives you a pointer

// to the object's internal storage - it is not recommended that you change

// the memory referenced by get\_mac.

//

// To read from the ethernet interface, call net.recv\_frame with a buffer

// address (and maximum size). The function waits for the next frame (unless

// one is already queued up) and copies it into the buffer (except for the

// CRC, which is handled by the interface). The function returns the actual

// number of bytes in the frame, but beware, it may not match the number of

// bytes in the logical frame (although it better not be smaller). Usually,

// you will want to dedicate a thread to reading the frame from the network

// and dispatching them to the protocol stack(s).

//

// To write a frame to the interface, call net.send\_frame with the address

// and size of the frame to send. Again, leave off the CRC - the interface

// handles that. send\_frame returns the number of bytes actually written,

// but you can usually ignore that.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef FRAMEIO\_H

#define FRAMEIO\_H

#include <sys/socket.h>

#include <unistd.h>

//

// it is a pain to declare unsigned chars everywhere, so we define

// "octet" to be unsigned char

//

**typedef** unsigned char octet**;**

//

// class frameio - see descripion in the file header.

//

class frameio

**{**

public**:**

//

// send a frame to the open interface, return number of bytes sent

//

int send\_frame**(**void **\***frame**,** int len**)**

**{**

**return** write**(**sock**,**frame**,**len**);**

**}**

//

// block, waiting for a frame. When it arrives, copy it into the buffer.

// return the number of bytes in the wire packet

//

int recv\_frame**(**void **\***frame**,** int max\_len**)**

**{**

**return** recvfrom**(**sock**,**frame**,**max\_len**,**0**,NULL,NULL);**

**}**

//

// open a socket on the specified interface and load my\_mac

//

int open\_net**(**const char **\***device**);** // e.g. "eth0"

//

// return this interface's MAC address

//

const octet **\***get\_mac**()**

**{**

**return** my\_mac**;**

**}**

//

// Constructor

//

frameio**()**

**{**

// make sure sock is not valid

sock **=** **-**1**;**

**}**

//

// Destructor

//

**~**frameio**()**

**{**

// if socket has been opened, close it

**if** **(** sock **>=** 0 **)** close**(**sock**);**

sock **=** **-**1**;**

**}**

private**:**

octet my\_mac**[**6**];** // this interface's 6-byte MAC address

int sock**;** // socket descriptor

**};**

#endif

# frameio.cpp (code provided by instructor)

#include "frameio.h"

#include <stdio.h>

#include <stdlib.h>

#include <sys/ioctl.h>

#include <net/if.h>

#include <netpacket/packet.h>

#include <net/ethernet.h>

#include <string.h>

#include <netinet/in.h>

//

// Open a raw socket on the interface and get the MAC address

//

int frameio**::**open\_net**(**const char **\***device**)**

**{**

const int LEN **=** 80**;**

struct sockaddr\_ll sll**;**

struct ifreq ifreq**;**

//

// set up the link-layer socket address

//

memset**(&**sll**,** 0**,** **sizeof** sll**);**

sll**.**sll\_family **=** PF\_PACKET**;**

sll**.**sll\_protocol **=** htons **(**ETH\_P\_ALL**);**

//

// open the socket, tell the OS we want all protocols

//

sock **=** socket **(**PF\_PACKET**,** SOCK\_RAW**,** htons **(**ETH\_P\_ALL**));**

**if** **(** sock **<** 0 **)** **return** sock**;**

//

// if all you want to do is read, we could stop here. But to

// write a frame, we have to bind this socket to an interface

// and to do that, we need its interface number (small int)

//

strcpy **(**ifreq**.**ifr\_name**,** device**);**

ioctl **(**sock**,** SIOCGIFINDEX**,** **&**ifreq**);**

sll**.**sll\_ifindex **=** ifreq**.**ifr\_ifindex**;**

//

// now let us bind...

//

bind **(**sock**,** **(**struct sockaddr **\*)** **&**sll**,** **sizeof** sll**);**

//

// get the mac address

//

struct ifreq ifr**;**

strncpy**(&**ifr**.**ifr\_name**[**0**],** device**,** IFNAMSIZ**);**

**if** **(**ioctl**(**sock**,** SIOCGIFHWADDR**,** **&**ifr**)** **>=** 0**)**

**{**

memcpy**(**my\_mac**,** **&**ifr**.**ifr\_hwaddr**.**sa\_data**,** 6**);**

**return** sock**;**

**}**

//

// could have failed three different ways, but failed nontheless...

//

close**(**sock**);**

**return** sock **=** **-**1**;**

**}**