# Objective

To understand the IP packet structure and implement ICMP echo.

# Structural Overview

The machine will run code that initiates an ICMP echo request to two IP addresses: one in the same subnet and one outside the subnet. It is expected that the program will determine the destination MAC address via ARP developed in the previous lab, directing IP addresses out of the subnet towards a hard-coded default gateway. Another machine will ping the host running the program, to which it is expected to respond appropriately.

# Simulation

|  |  |
| --- | --- |
| Figure 1 - Parsing ICMP Requests | Figure 2 - Console output ICMP echo request with ARP  Figure 3 - Wireshark output ICMP echo request with ARP  Figure 4 - Console output ICMP echo out of subnet |

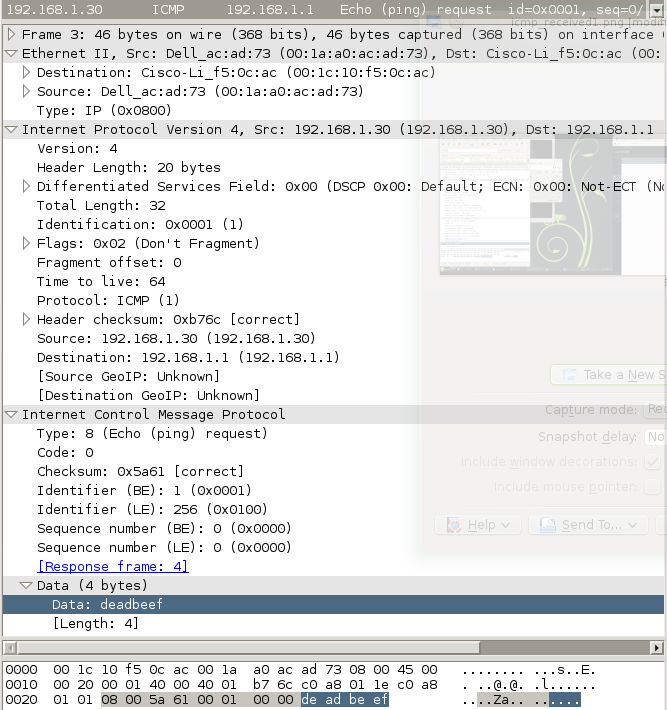


Figure 5 - Custom echo request within the subnet

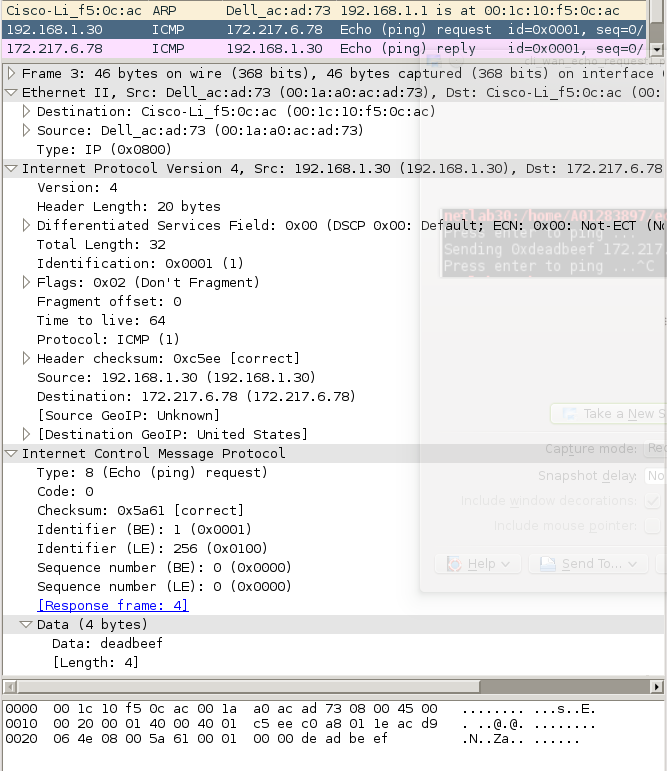


Figure 6 - Custom echo request out of subnet

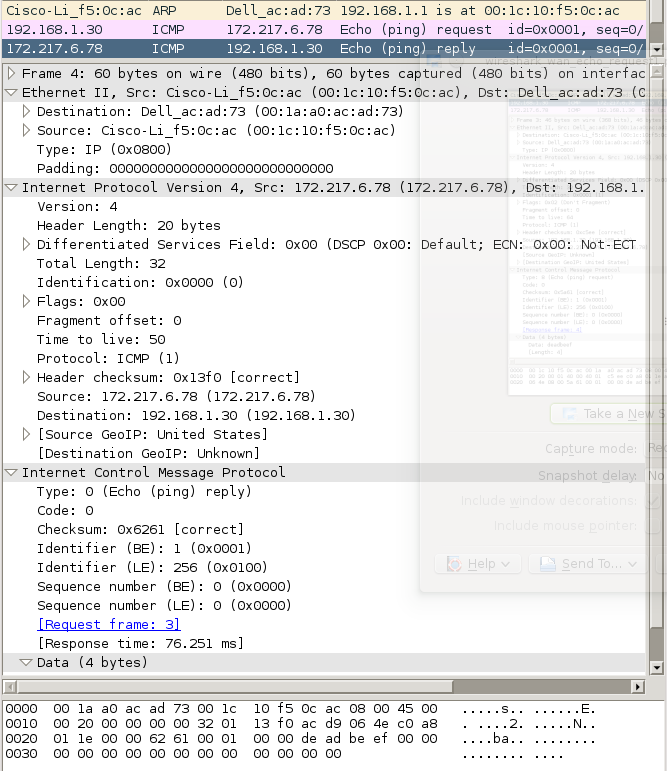


Figure 7 - Google echoes deadbeef

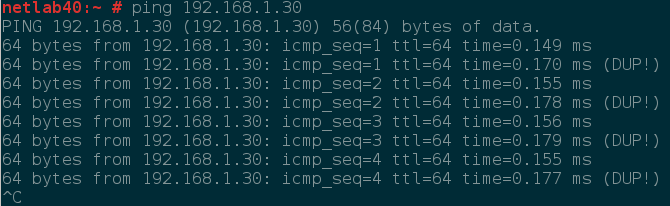


Figure 8 - Console ping request from remote device

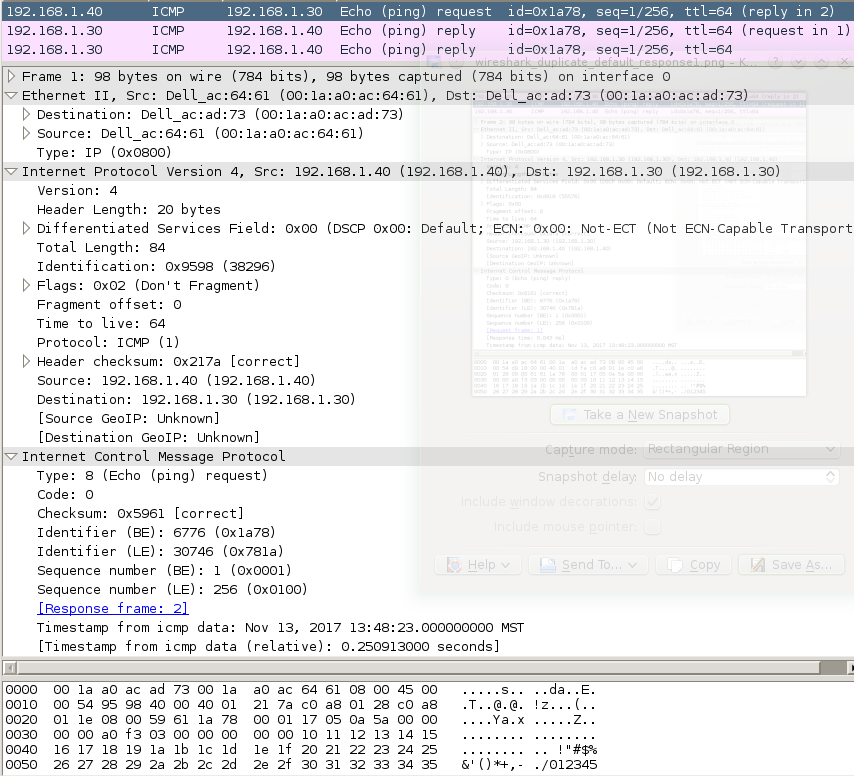


Figure 9 - An incoming ping request

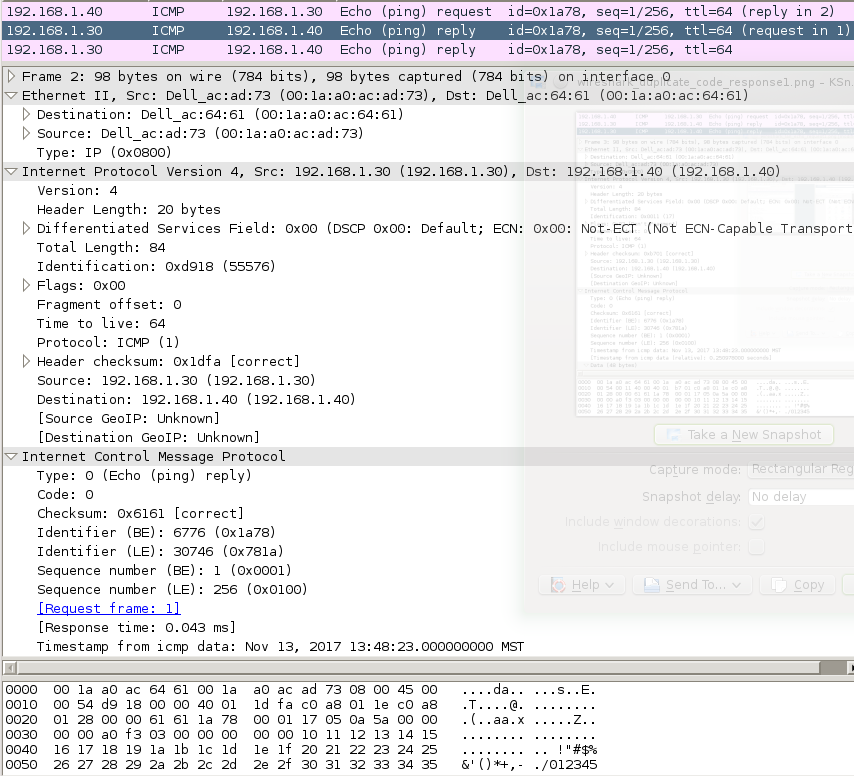


Figure 10 - Automatic echo response from machine

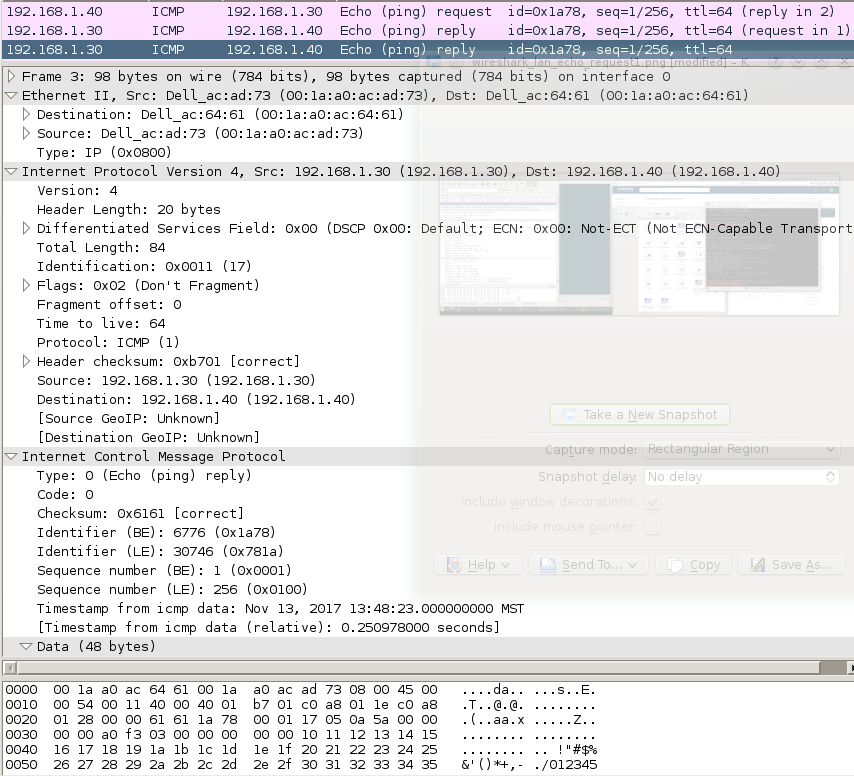


Figure 11 - Custom echo response from program

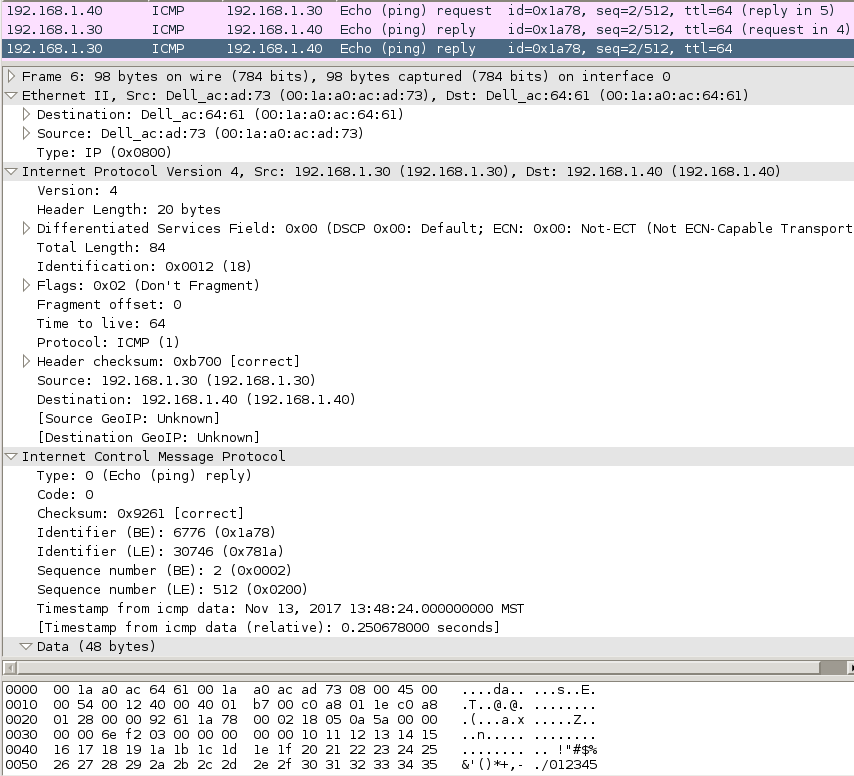


Figure 12 - Custom response for next request in the ICMP sequence

# Results

The results you can see in figure one are a result of commented out code that would print to the console upon reception of any ICMP message. For custom ICMP echo requests used later, the message deadbeef was used.

Figures 2 and 3 correspond to sending an ICMP echo request before and after a cached ARP entry to demonstrate the automatic use of ARP upon requesting an unknown IP within the subnet. Figure 5 shows the packet used for the ARP request within the subnet.

Figures 4 and 6 demonstrate the output for an ICMP message request out of the subnet. As it happens with Figures 2, 3, and 5, I pinged the default gateway instead of another machine in the subnet (I promise it works for other devices as well…I left the lab before realizing this error and am unable to update with corrective screen captures)

Figure 7 is partially for fun, I got Google to say deadbeef. However, the fact it responded demonstrates correct implementation.

Figure 8 shows the output from a machine used in the final procedure for this lab: responding to an ICMP echo request. Note the “(DUP)” appearing as a result of our code responding in addition to the machine’s default response.

Figure 9 shows the packet data for the ping request from the remote machine.

Figure 10 shows the packet data for the machine’s default response.

Figure 11 shows the packet data for our custom code’s response.

Figure 12 demonstrates an increasing sequence number in the echo requests, to which our program correctly responds.

# Notes

I renamed given code samples from the instructor to match their language (.hpp instead of .h) and util to message\_queue as a more descriptive name (at some point I was considering adding my own util.hpp). Additionally I renamed the type ‘octet’ to ‘byte’

# makefile

lab3**:** main.cpp net.hpp chksum.o message\_queue.o frameio.o

g++ -std=c++11 main.cpp chksum.o message\_queue.o frameio.o -lpthread -g -O0 -o lab3

chksum.o**:** chksum.c

g++ chksum.c -c -o chksum.o

message\_queue.o**:** message\_queue.cpp message\_queue.hpp

g++ message\_queue.cpp -c -o message\_queue.o

frameio.o**:** frameio.cpp frameio.hpp

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

clean**:**

rm \*.o

rm lab3

# net.hpp

#pragma once

**typedef** unsigned char byte**;**

extern int chksum**(**byte**\*** s**,** int n**,** int i**);**

// macro converts byte[] into ushort, uint

#define BUFF\_UINT16(buff, i) (buff[i + 0] << 8 | buff[i + 1] << 0)

#define BUFF\_UINT32(buff, i) (buff[i + 0] << 24 | buff[i + 1] << 16 | buff[i + 2] << 8 | buff[i + 3] << 0)

struct ipmac

**{**

byte mac**[**6**];**

byte ip**[**4**];**

**};**

struct net\_device

**{**

union

**{**

ipmac arp\_cache\_self**;**

struct

**{**

byte mac**[**6**];**

byte ip**[**4**];**

**};**

**};**

byte subnet\_mask**[**4**];**

byte default\_gateway**[**4**];**

**};**

//----------------------------------------------------------------------------+

// Ethernet 802.3/DIX frames |

struct ether\_header

**{**

byte dst**[**6**];**

byte src**[**6**];**

union

**{**

byte len**[**2**];**

byte prot**[**2**];**

**};**

**};**

struct ether\_frame

**{**

ether\_header header**;**

byte data**[**1500**];**

**};**

ether\_frame**\*** make\_frame**(**byte**\*** dst**,** unsigned short prot**,** byte**\*** data**,** int n**);**

//----------------------------------------------------------------------------+

//----------------------------------------------------------------------------+

// ARP |

struct arp\_header

**{**

byte hwtype**[**2**];**

byte prottype**[**2**];**

byte hwlength**;**

byte protlength**;**

byte opcode**[**2**];**

**};**

struct arp\_frame

**{**

arp\_header header**;**

byte data**[**1500 **-** **sizeof(**arp\_header**)];**

**};**

ipmac**\*** retrieveArpCache**(**byte**\*** value**);**

void saveArpCache**(**ipmac**\*** value**);**

void pingARP**(**byte**\*** ip**);**

//----------------------------------------------------------------------------+

//----------------------------------------------------------------------------+

// IP |

struct ip\_header

**{**

byte ver\_ihl**;**

byte dscp**;**

byte length**[**2**];**

byte ident**[**2**];**

byte frag**[**2**];**

byte ttl**;**

byte prot**;**

byte crc**[**2**];**

byte src**[**4**];**

byte dst**[**4**];**

**};**

struct ip\_frame

**{**

ip\_header header**;**

byte data**[**1500 **-** **sizeof(**ip\_header**)];**

**};**

void sendIPv4Packet**(**byte**\*** ip**,** byte prot**,** byte**\*** payload**,** int n**);**

//----------------------------------------------------------------------------+

//----------------------------------------------------------------------------+

// ICMP |

struct icmp\_header

**{**

byte type**;**

byte code**;**

byte crc**[**2**];**

union

{

byte header[4];

struct

{

byte ident[2];

byte seqno[2];

} echo;

};

};

struct icmp\_frame

{

icmp\_header header;

byte data[1500 - sizeof(ip\_header) - sizeof(icmp\_header)];

};

void pingICMP(byte\* ip, byte\* data, int n);

//----------------------------------------------------------------------------+

#define ETHER\_PROT\_IPV4 0x0800

#define ETHER\_PROT\_ARP 0x0806

void arp\_handler(byte\* packet, int n, ether\_header\* header);

void ip\_handler(byte\* packet, int n, ether\_header\* header);

#define IPV4\_PROT\_ICMP 0x01

void icmp\_handler(byte\* packet, int n, ip\_header\* header);

# main.cpp

#include "frameio.hpp"

#include "message\_queue.hpp"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <pthread.h>

#include <unordered\_map>

#include "net.hpp"

// device name must be hard-coded

frameio net**(**"enp3s0"**);**

net\_device me **=**

**{**

0**,** 0**,** 0**,** 0**,** 0**,** 0**,** // mac copied at start of main()

192**,** 168**,** 1**,** 30**,** // ip must be hard-coded

255**,** 255**,** 255**,** 0**,** // subnet mask must be hard-coded

192**,** 168**,** 1**,** 1**,** // default gateway must be hard-coded

**};**

std**::**unordered\_map**<**int**,** ipmac**\*>** arp\_cache**;**

inline int hash\_ip**(**byte**\*** ip**)**

**{**

static const int hash\_mask **=** **~**BUFF\_UINT32**(**me**.**subnet\_mask**,** 0**);**

int ip4 **=** BUFF\_UINT32**(**ip**,** 0**);**

int key **=** ip4 **&** hash\_mask**;**

**return** key**;**

**}**

ipmac**\*** retrieveArpCache**(**byte**\*** ip**)**

**{**

int key **=** hash\_ip**(**ip**);**

auto search **=** arp\_cache**.**find**(**key**);**

**if** **(**search **!=** arp\_cache**.**end**())** **{**

**return** search**->**second**;**

**}**

**return** **NULL;**

**}**

void saveArpCache**(**ipmac**\*** value**)**

**{**

ipmac**\*** found **=** retrieveArpCache**(**value**->**ip**);**

**if** **(**found **==** **NULL)**

**{**

// insert

ipmac**\*** copy **=** **(**ipmac**\*)**malloc**(sizeof(**ipmac**));**

memcpy**(**copy**,** value**,** **sizeof(**ipmac**));**

int key **=** hash\_ip**(**copy**->**ip**);**

arp\_cache**.**insert**({**key**,** copy**});**

**}**

**else**

**{**

// update

memcpy**(**found**,** value**,** **sizeof(**ipmac**));**

**}**

**}**

// message queue for the sending ether\_frames

message\_queue send\_queue**;**

void**\*** send\_thread**(**void**\*** args**)**

**{**

int n**;**

ether\_frame frame**;**

event\_kind event**;**

**while(**1**)**

**{**

n **=** send\_queue**.**recv**(&**event**,** **&**frame**,** **sizeof(**ether\_frame**));**

net**.**send\_frame**(&**frame**,** n**);**

**}**

**}**

void**\*** receive\_thread**(**void**\*** args**)**

**{**

ether\_frame frame**;**

**while(**1**)**

**{**

int n **=** net**.**recv\_frame**(&**frame**,** **sizeof(**ether\_frame**));**

**if** **(**n **<** 42**)** **continue;** // bad frame!

**switch** **(**BUFF\_UINT16**(**frame**.**header**.**prot**,** 0**))**

**{**

**case** ETHER\_PROT\_IPV4**:**

ip\_handler**(**frame**.**data**,** n **-** **sizeof(**ether\_header**),** **&(**frame**.**header**));**

**break;**

**case** ETHER\_PROT\_ARP**:**

arp\_handler**(**frame**.**data**,** n **-** **sizeof(**ether\_header**),** **&(**frame**.**header**));**

**break;**

**}**

**}**

**}**

ether\_frame**\*** make\_frame**(**byte**\*** dst**,** unsigned short prot**,** byte**\*** data**,** int n**)**

**{**

ether\_frame**\*** out **=** **(**ether\_frame**\*)**malloc**(**n **+** **sizeof(**ether\_header**));**

memcpy**(**out**->**header**.**dst**,** dst**,** 6**);**

memcpy**(**out**->**header**.**src**,** me**.**mac**,** 6**);**

out**->**header**.**prot**[**0**]** **=** **(**prot **&** 0xFF00**)** **>>** 8**;**

out**->**header**.**prot**[**1**]** **=** **(**prot **&** 0x00FF**)** **>>** 0**;**

memcpy**(**out**->**data**,** data**,** n**);**

**return** out**;**

**}**

void arp\_handler**(**byte**\*** packet**,** int n**,** ether\_header**\*** header**)**

**{**

arp\_frame**\*** frame **=** **(**arp\_frame**\*)**packet**;**

**switch** **(**BUFF\_UINT16**(**frame**->**header**.**opcode**,** 0**))**

**{**

**case** 1**:** // Request

saveArpCache**(((**ipmac**\*)**frame**->**data**)** **+** 0**);**

**if** **(**frame**->**data**[**16**]** **==** me**.**ip**[**0**]** **&&**

frame**->**data**[**17**]** **==** me**.**ip**[**1**]** **&&**

frame**->**data**[**18**]** **==** me**.**ip**[**2**]** **&&**

frame**->**data**[**19**]** **==** me**.**ip**[**3**])**

**{**

// Start with a response frame that has a payload exactly matching what we received

ether\_frame**\*** response **=** make\_frame**(**frame**->**data**,** ETHER\_PROT\_ARP**,** **(**byte**\*)&**frame**,** n**);**

arp\_frame**\*** response\_arp **=** **(**arp\_frame**\*)((**byte**\*)(**response**)** **+** **sizeof(**ether\_header**));**

// Convert to reply opcode

response\_arp**->**header**.**opcode**[**1**]** **=** 2**;**

// Move the sender info the the target info

memcpy**(**response\_arp**->**data **+** **sizeof(**ipmac**),** response\_arp**->**data **+** 0**,** **sizeof(**ipmac**));**

// Fill the sender info with our info

memcpy**(**response\_arp**->**data **+** 0**,** **&**me**,** **sizeof(**ipmac**));**

send\_queue**.**send**(**PACKET**,** response**,** n **+** **sizeof(**ether\_header**));**

free**(**response**);**

**}**

**break;**

**case** 2**:** // Reply

saveArpCache**(((**ipmac**\*)**frame**->**data**)** **+** 0**);**

saveArpCache**(((**ipmac**\*)**frame**->**data**)** **+** 1**);**

**break;**

**}**

**}**

void ip\_handler**(**byte**\*** packet**,** int n**,** ether\_header**\*** header**)**

**{**

static const int this\_ip **=** BUFF\_UINT32**(**me**.**ip**,** 0**);**

ip\_frame**\*** frame **=** **(**ip\_frame**\*)**packet**;**

// Validate the checksum

**if** **(**chksum**(**packet**,** **sizeof(**ip\_header**),** 0**)** **!=** 0xffff**)**

**{**

printf**(**"IP message received with bad checksum\n"**);**

**return;**

**}**

// Ignore packets meant for others

**if** **(**BUFF\_UINT32**(**frame**->**header**.**dst**,** 0**)** **!=** this\_ip**)**

**{**

**return;**

**}**

// Don't include any padding in ip packet

int len **=** BUFF\_UINT16**(**frame**->**header**.**length**,** 0**);**

**if** **(**n **>** len**)** **{** n **=** len**;** **}**

// This should be a rare error condition

**if** **(**n **<** len**)**

**{**

printf**(**"IP message received with missing data\n"**);**

**return;**

**}**

// Find the payload

byte**\*** payload **=** frame**->**data**;**

int option\_bytes **=** 4 **\*** **((**frame**->**header**.**ver\_ihl **&** 0x0f**)** **-** 5**);**

payload **=** payload **+** option\_bytes**;**

int payload\_n **=** n **-** option\_bytes **-** **sizeof(**ip\_header**);**

//printf("IP message received, protocol: %i\n", frame->header.prot);

**switch** **(**frame**->**header**.**prot**)**

**{**

**case** IPV4\_PROT\_ICMP**:**

icmp\_handler**(**payload**,** payload\_n**,** **&(**frame**->**header**));**

**break;**

**}**

**}**

void icmp\_handler**(**byte**\*** packet**,** int n**,** ip\_header**\*** header**)**

**{**

icmp\_frame**\*** frame **=** **(**icmp\_frame**\*)**packet**;**

// Validate the checksum

**if** **(**chksum**(**packet**,** n**,** 0**)** **!=** 0xffff**)**

**{**

printf**(**"ICMP message received with bad checksum\n"**);**

**return;**

**}**

//printf("ICMP message received\n");

**switch** **(**frame**->**header**.**type**)**

**{**

**case** 0x08**:** // echo (ping) request

frame**->**header**.**type **=** 0x00**;** // echo (ping) reply

frame**->**header**.**crc**[**0**]** **=** 0**;**

frame**->**header**.**crc**[**1**]** **=** 0**;**

int crc **=** **~**chksum**((**byte**\*)**frame**,** n**,** 0**);**

frame**->**header**.**crc**[**0**]** **=** **(**crc **&** 0xff00**)** **>>** 8**;**

frame**->**header**.**crc**[**1**]** **=** **(**crc **&** 0x00ff**)** **>>** 0**;**

sendIPv4Packet**(**header**->**src**,** IPV4\_PROT\_ICMP**,** packet**,** n**);**

**break;**

**}**

**}**

void pingARP**(**byte**\*** ip**)**

**{**

static arp\_frame message **=** **{**

**{**

**{** 0**,** 1 **},**

**{** 8**,** 0 **},**

6**,** 4**,**

**{** 0**,** 1 **},**

**},**

**{** 0 **},**

**};**

static const int n **=** **sizeof(**arp\_header**)** **+** **(**2 **\*** **sizeof(**ipmac**));**

**if** **(**message**.**data**[**0**]** **==** 0**)**

**{**

memcpy**(**message**.**data**,** **&**me**,** **sizeof(**ipmac**));**

**}**

ipmac**\*** found **=** retrieveArpCache**(**ip**);**

**if(**found **==** **NULL)**

**{**

ipmac value **=** **{** 0xff**,** 0xff**,** 0xff**,** 0xff**,** 0xff**,** 0xff**,** 0**,** 0**,** 0**,** 0 **};**

memcpy**(**value**.**ip**,** ip**,** 4**);**

memcpy**(((**ipmac**\*)(**message**.**data**))** **+** 1**,** **&**value**,** **sizeof(**ipmac**));**

**}**

**else**

**{**

memcpy**(((**ipmac**\*)(**message**.**data**))** **+** 1**,** found**,** **sizeof(**ipmac**));**

**}**

byte**\*** dest\_mac = **(**byte**\*)(((**ipmac**\*)(**message**.**data**))** **+** 1**)**;

ether\_frame**\*** frame **=** make\_frame**(**dest\_mac**,** ETHER\_PROT\_ARP**,** **(**byte**\*)(&**message**),** n**);**

send\_queue**.**send**(**PACKET**,** frame**,** n **+** **sizeof(**ether\_header**));**

free**(**frame**);**

**}**

inline byte**\*** hop\_ip**(**byte**\*** ip**)**

**{**

static const int gateway **=** BUFF\_UINT32**(**me**.**default\_gateway**,** 0**);**

static const int subnet\_mask **=** BUFF\_UINT32**(**me**.**subnet\_mask**,** 0**);**

static const int subnet **=** subnet\_mask **&** gateway**;**

int ip4 **=** BUFF\_UINT32**(**ip**,** 0**);**

**if** **((**ip4 **&** subnet\_mask**)** **==** subnet**)**

**{**

**return** ip**;**

**}**

**return** me**.**default\_gateway**;**

**}**

byte**\*** get\_mac**(**byte**\*** ip**)**

**{**

byte**\*** dst\_ip **=** hop\_ip**(**ip**);**

ipmac**\*** dst **=** retrieveArpCache**(**dst\_ip**);**

int attempts **=** 4**;**

**while** **(**dst **==** **NULL** **&&** **--**attempts **>=** 0**)**

**{**

pingARP**(**dst\_ip**);**

sleep**(**1**);**

dst **=** retrieveArpCache**(**dst\_ip**);**

**}**

**if** **(**dst **==** **NULL)**

**{**

printf**(**"Unable to resolve ip address: %i.%i.%i.%i\n"**,** ip**[**0**],** ip**[**1**],** ip**[**2**],** ip**[**3**]);**

**return** **NULL;**

**}**

**return** dst**->**mac**;**

**}**

void sendIPv4Packet**(**byte**\*** ip**,** byte prot**,** byte**\*** payload**,** int n**)**

**{**

static unsigned short identifier **=** 0**;**

static ip\_frame request **=** **{** 0 **};**

/\*

{

{

{ 4, 5 }, // 0x45 // ipv4 optionless header

{ 0, 0 }, // default dscp

{ 0x00, 0x00 }, // length (calculated at each call)

{ 0x00, 0x00 }, // id (calculated at each call)

{ 2, 0 }, // 0x4000, // no fragmentation

64, // ttl 64 (seems common for a default)

0, // protocol: (copied at each call)

{ 0 }, // checksum (0 to start)

{ 0 }, // source (0 for now, copied on first call)

{ 0 }, // destination (copied at each call)

},

{ 0 }, // payload (copied at each call)

};

\*/

// static initializer for request

**if** **(**request**.**header**.**ver\_ihl **==** 0**)**

**{**

//request.header.version = 4;

//request.header.ihl = 5; // no options

request**.**header**.**ver\_ihl **=** 0x45**;**

//request.header.flags = 2; // no fragmentation

request**.**header**.**frag**[**0**]** **=** 0x40**;**

request**.**header**.**ttl **=** 64**;**

memcpy**(**request**.**header**.**src**,** me**.**ip**,** 4**);** // copy source ip

**}**

byte**\*** dst\_mac **=** get\_mac**(**ip**);**

**if** **(**dst\_mac **==** **NULL)** **{** **return;** **}**

**++**identifier**;**

int N **=** **sizeof(**ip\_header**)** **+** n**;**

ether\_frame**\*** frame **=** make\_frame**(**dst\_mac**,** ETHER\_PROT\_IPV4**,** **(**byte**\*)(&**request**),** N**);**

ip\_frame**\*** packet **=** **(**ip\_frame**\*)(**frame**->**data**);**

memcpy**(**packet**->**data**,** payload**,** n**);**

memcpy**(**packet**->**header**.**dst**,** ip**,** 4**);**

packet**->**header**.**length**[**0**]** **=** **(**N **&** 0xff00**)** **>>** 8**;**

packet**->**header**.**length**[**1**]** **=** **(**N **&** 0x00ff**)** **>>** 0**;**

packet**->**header**.**ident**[**0**]** **=** **(**identifier **&** 0xff00**)** **>>** 8**;**

packet**->**header**.**ident**[**1**]** **=** **(**identifier **&** 0x00ff**)** **>>** 0**;**

packet**->**header**.**prot **=** prot**;**

int crc **=** **~**chksum**((**byte**\*)**packet**,** **sizeof(**ip\_header**),** 0**);**

packet**->**header**.**crc**[**0**]** **=** **(**crc **&** 0xff00**)** **>>** 8**;**

packet**->**header**.**crc**[**1**]** **=** **(**crc **&** 0x00ff**)** **>>** 0**;**

send\_queue**.**send**(**PACKET**,** frame**,** N **+** **sizeof(**ether\_header**));**

free**(**frame**);**

**}**

void pingICMP**(**byte**\*** ip**,** byte**\*** data**,** int n**)**

**{**

static unsigned short identifier **=** 0**;**

static icmp\_frame request **=**

**{**

**{**

0x08**,** // echo (ping) request

0x00**,** // code

**{** 0 **},** // checksum (computed every call)

**{** 0 **},** // header (computed every call)

**},**

**{** 0 **},**

**};**

**++**identifier**;**

unsigned short sequence **=** 0**;**

memcpy**(**request**.**data**,** data**,** n**);**

int N **=** n **+** **sizeof(**icmp\_header**);**

request**.**header**.**crc**[**0**]** **=** 0**;**

request**.**header**.**crc**[**1**]** **=** 0**;**

request**.**header**.**echo**.**ident**[**0**]** **=** **(**identifier **&** 0xff00**)** **>>** 8**;**

request**.**header**.**echo**.**ident**[**1**]** **=** **(**identifier **&** 0x00ff**)** **>>** 0**;**

request**.**header**.**echo**.**seqno**[**0**]** **=** **(**sequence **&** 0xff00**)** **>>** 8**;**

request**.**header**.**echo**.**seqno**[**1**]** **=** **(**sequence **&** 0x00ff**)** **>>** 0**;**

int crc **=** **~**chksum**((**byte**\*)(&**request**),** N**,** 0**);**

request**.**header**.**crc**[**0**]** **=** **(**crc **&** 0xff00**)** **>>** 8**;**

request**.**header**.**crc**[**1**]** **=** **(**crc **&** 0x00ff**)** **>>** 0**;**

sendIPv4Packet**(**ip**,** IPV4\_PROT\_ICMP**,** **(**byte**\*)(&**request**),** N**);**

**}**

int main**()**

**{**

memcpy**(**me**.**mac**,** net**.**get\_mac**(),** 6**);**

arp\_cache**[**me**.**ip**[**3**]]** **=** **&(**me**.**arp\_cache\_self**);**

int err**;**

pthread\_t rthread**,** sthread**;**

// Create the threads

err **=** pthread\_create**(&**rthread**,** **NULL,** receive\_thread**,** **NULL);**

err **=** pthread\_create**(&**sthread**,** **NULL,** send\_thread**,** **NULL);**

//------------------------------------------------------------------------+

// main application routine |

byte request**[**4**]** **=** **{** 192**,** 168**,** 1**,** 30 **};**

byte payload**[**4**]** **=** **{** 0xde**,** 0xad**,** 0xbe**,** 0xef **};**

**while(**1**)** **{**

printf**(**"Press enter to ping ..."**);**

getchar**();**

//for(int i = 0; i < 5; ++i)

**{**

//request[3] = 10 + i \* 5;

printf**(**"Sending 0xdeadbeef %i.%i.%i.%i\n"**,** request**[**0**],** request**[**1**],** request**[**2**],** request**[**3**]);**

pingICMP**(**request**,** payload**,** 4**);**

**}**

**}**

// main application routine |

//------------------------------------------------------------------------+

// Put main() to sleep until threads exit

err **=** pthread\_join**(**rthread**,** **NULL);**

err **=** pthread\_join**(**sthread**,** **NULL);**

**return** 0**;**

**}**