Completed with Lab Partner: Josh Lake

# Objective

To understand the ARP mechanism and construct an ARP cache.

# Structural Overview

One computer will use the command `arping` to demonstrate regular usage of the address resolution protocol. The program developed in this lab will reply to the ARP request in addition to the reply the operating system generates (these two replies should be the same). The program will, responding to user input, send ARP requests for all IPs not in the cache and ARP replies to all IPs in the cache.

# Simulation

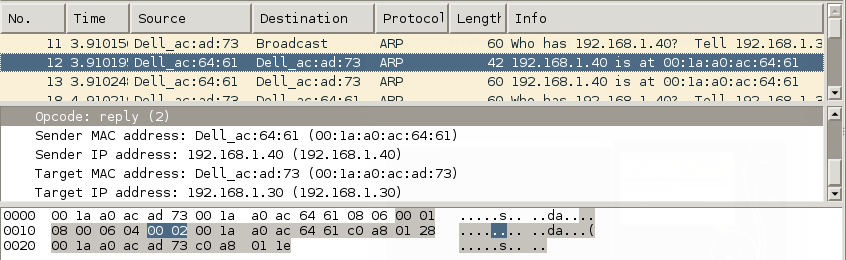


Figure - The Default OS-Generated Reply

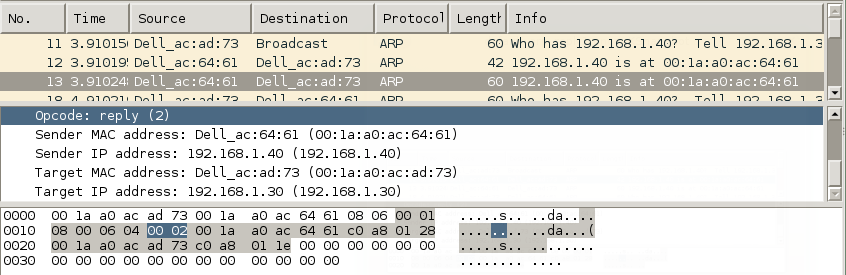


Figure - Our Code-Generated Reply

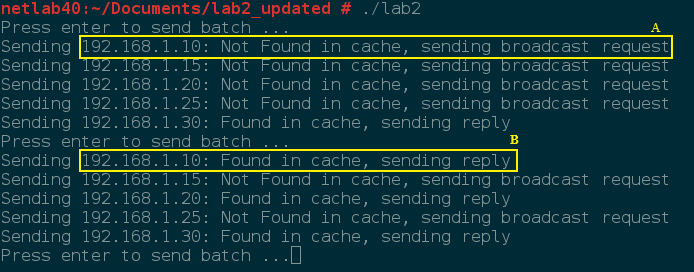


Figure - Console Output

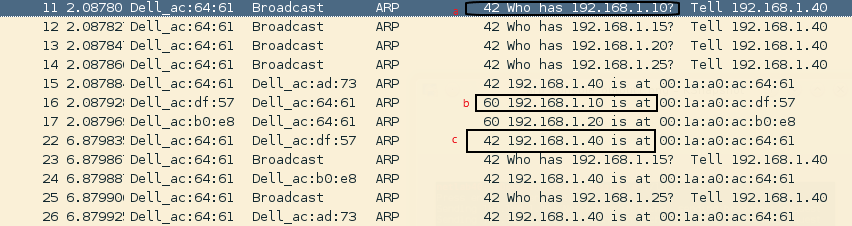


Figure - Wireshark Output

# Results

Figures 1 and 2 together demonstrate our program responding to an ARP request the same way the OS does. Wireshark omits the padding in OS-Generated messages for reasons undetermined. However, we demonstrated in lab session with the assistance of the TA’s that Wireshark on a different machine shows the padding for incoming messages whether OS-Generated or Code-Generated.

Figures 3 and 4 demonstrate the same period showing the following communication

* An IP address is not in the ARP cache, so the program sends a broadcast ARP request (annotated: A, a)
* The computer with that IP sends a reply ARP packet (annotated: b), that IP/MAC pair is added to the cache
* At a later point the program forms an ARP message to the IP address which is now cached (annotated B, c)

Some notes about our code. For our ARP cache we took the liberty of assuming we would only deal with messages in the same subnet, and the subnet was the first three octals; thus we could implement a hash table using the last octal of the IP address as a hash key for the cache (for the purposes of this lab, it was discussed with the TAs that this was acceptable).

Additionally, our code requires hard-coding the ip address of the machine running the code (near the top of main.cpp). The hardware address will be obtained via the `frameio` class provided by the instructor. The `message\_queue` class was also provided by the instructor, and is a simplified implementation of the sender/receiver multi-threading pattern.

# Makefile

lab2**:** main.cpp util.o frameio.o

g++ main.cpp util.o frameio.o -lpthread -g -O0 -o lab2

util.o**:** util.cpp util.h

g++ util.cpp -c -g -O0 -o util.o

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

g++ frameio.cpp -c -g -O0 -o frameio.o

clean**:**

rm \*.o

rm lab2

# main.cpp

#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 send\_queue**;** // message queue for the sending ether\_frames

message\_queue ip\_queue**;** // message queue for the IP protocol stack

message\_queue arp\_queue**;** // message queue for the ARP protocol stack

struct ipmac

**{**

octet mac**[**6**];**

octet ip**[**4**];**

**};**

ipmac me **=** **{** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 192**,** 168**,** 1**,** 40 **};**

ipmac**\*** arp\_cache**[**256**]** **=** **{** 0 **};**

struct ether\_header

**{**

octet dst\_mac**[**6**];** // destination MAC address

octet src\_mac**[**6**];** // source MAC address

octet prot**[**2**];** // protocol (or length)

**};**

struct ether\_frame // handy template for 802.3/DIX frames

**{**

ether\_header header**;**

octet data**[**1500**];** // payload

**};**

#define ETHER\_PROT\_IP 0x0800

#define ETHER\_PROT\_ARP 0x0806

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

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

**{**

ether\_frame buf**;**

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

**{**

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

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

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

**{**

**case** ETHER\_PROT\_IP**:**

ip\_queue**.**send**(**PACKET**,** buf**.**data**,** n **-** **sizeof(**ether\_header**));**

**break;**

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

arp\_queue**.**send**(**PACKET**,** buf**.**data**,** n **-** **sizeof(**ether\_header**));**

**break;**

**}**

**}**

**}**

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

**{**

int n**;**

ether\_frame buf**;**

event\_kind event**;**

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

**{**

n **=** send\_queue**.**recv**(&**event**,** **&**buf**,** **sizeof(**buf**));**

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

**}**

**}**

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

**{**

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

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

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

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

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

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

**return** out**;**

**}**

struct arp\_header

**{**

octet hwtype**[**2**];**

octet prottype**[**2**];**

octet hwlength**;**

octet protlength**;**

octet opcode**[**2**];**

**};**

struct arp\_frame

**{**

arp\_header header**;**

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

**};**

ipmac**\*** retrieveFromCache**(**ipmac**\*** value**)**

**{**

**return** arp\_cache**[**value**->**ip**[**3**]];**

**}**

void saveToCache**(**ipmac**\*** value**)**

**{**

**if** **(**retrieveFromCache**(**value**)** **==** **NULL)**

**{**

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

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

arp\_cache**[**value**->**ip**[**3**]]** **=** copy**;**

**}**

**}**

void**\*** arp\_protocol**(**void**\*** args**)**

**{**

int n**;**

arp\_frame buf**;**

event\_kind event**;**

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

**{**

n **=** arp\_queue**.**recv**(&**event**,** **&**buf**,** **sizeof(**buf**));**

**switch** **(**BUFF\_UINT16**(**buf**.**header**.**opcode**,** 0**))**

**{**

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

saveToCache**(((**ipmac**\*)**buf**.**data**)** **+** 0**);**

**if** **(**buf**.**data**[**16**]** **==** me**.**ip**[**0**]** **&&**

buf**.**data**[**17**]** **==** me**.**ip**[**1**]** **&&**

buf**.**data**[**18**]** **==** me**.**ip**[**2**]** **&&**

buf**.**data**[**19**]** **==** me**.**ip**[**3**])**

**{**

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

ether\_frame**\*** response **=** make\_frame**(**buf**.**data**,** ETHER\_PROT\_ARP**,** **(**octet**\*)&**buf**,** n**);**

arp\_frame**\*** response\_arp **=** **(**arp\_frame**\*)((**octet**\*)(**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

saveToCache**(((**ipmac**\*)**buf**.**data**)** **+** 0**);**

saveToCache**(((**ipmac**\*)**buf**.**data**)** **+** 1**);**

**break;**

**}**

**}**

**}**

// assuming value->mac = { ff, ff, ff, ff, ff, ff }

void sendARP**(**ipmac**\*** value**)**

**{**

ipmac**\*** found **=** retrieveFromCache**(**value**);**

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

**{**

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

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

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

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

**},**

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

**};**

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

**{**

printf**(**"Not Found in cache, sending broadcast request\n"**);**

message**.**header**.**opcode**[**1**]** **=** 1**;** // request

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

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

**}**

**else**

**{**

printf**(**"Found in cache, sending reply\n"**);**

message**.**header**.**opcode**[**1**]** **=** 2**;** // reply

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

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

**}**

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

ether\_frame**\*** frame **=** make\_frame**((**octet**\*)(((**ipmac**\*)(**message**.**data**))** **+** 1**),** ETHER\_PROT\_ARP**,** **(**octet**\*)(&**message**),** n**);**

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

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

**}**

int main**()**

**{**

// Open the shared resource before starting threads

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

const octet**\*** mymac **=** net**.**get\_mac**();**

me**.**mac**[**0**]** **=** mymac**[**0**];**

me**.**mac**[**1**]** **=** mymac**[**1**];**

me**.**mac**[**2**]** **=** mymac**[**2**];**

me**.**mac**[**3**]** **=** mymac**[**3**];**

me**.**mac**[**4**]** **=** mymac**[**4**];**

me**.**mac**[**5**]** **=** mymac**[**5**];**

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

int err**;**

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

pthread\_t arpthread**;**

// Create the threads

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

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

err **=** pthread\_create**(&**arpthread**,** **NULL,** arp\_protocol**,** **NULL);**

ipmac request **=** **{**

0xff**,** 0xff**,** 0xff**,** 0xff**,** 0xff**,** 0xff**,** 192**,** 168**,** 1**,** 0

**};**

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

printf**(**"Press enter to send batch ..."**);**

getchar**();**

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

**{**

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

printf**(**"Sending 192.168.1.%i: "**,** request**.**ip**[**3**]);**

sendARP**(&**request**);**

**}**

**}**

// Put main() to sleep until threads exit

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

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

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

**return** 0**;**

**}**