Project 2 Report
ECE 5600

Nathan Tipton
A01207112
Partner: Erik Sargent

Oct 14, 2017

1 Objective

The purpose of this project is to familiarize ourselves with the structure of an Ethernet frame. We will also develop our understanding of the ARP mechanism. Using Wireshark we will analyze captured frames and locate the MAC or IP addresses. We will implement the ARP cache.

2 Results

Frames on the network are captured by our program and stored in a frame buffer. Our program determines if the ARP frame is either a request or reply. The computer's MAC address and IP address are determined and stored. When the program receives an ARP request it compares the destination IP address with the computer's IP address. If there is a match, the program creates and sends an ARP reply. Figure 1 shows that in Wireshark we are able to see the reply from the computer and the duplicate reply from our implementation.

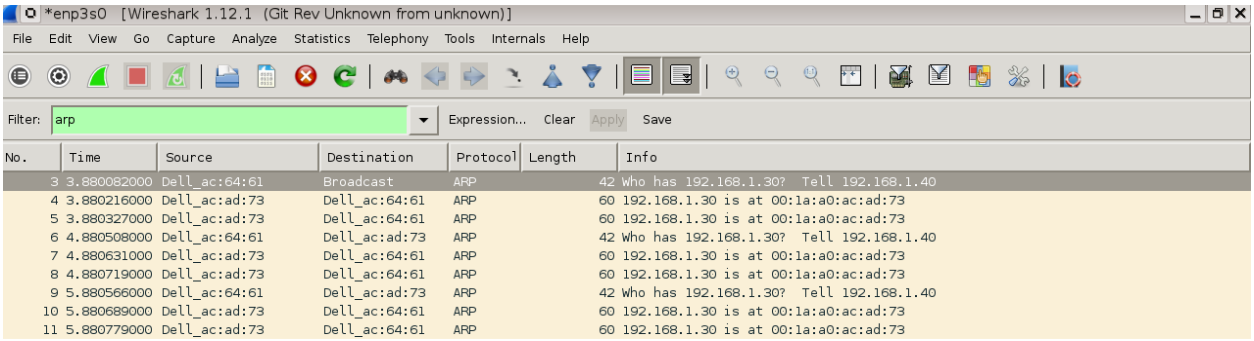

Figure 1: Wireshark screenshot of duplicate ARP reply

Next we implemented a cache for our ARP implementation. All ARP frames received have the source MAC and IP address parsed and cached. When an IP address is input on the console the cache is checked. If the IP address is not cached, an ARP request is sent and the resulting reply is then cached for future use. If the IP is found an ARP reply is immediately sent to the destination MAC address from the cache. Every cached MAC and IP pair is timestamped as they are parsed into the cache. After approximately 20 seconds the pair expires and is removed from the cache. If a message is attempted to an expired pair the request will be sent again. The test of our cache mechanism is shown in figure 2. First, we put in the desired IP address and the cache was checked. The IP was not found so the ARP request was sent. The reply was received and cached as the second frame shown in Wireshark. The third captured frame is the second time the IP address was entered into the console. This time the IP was found in the cache so an ARP reply was sent.

Captured frames 6 and 7 are after the 20 second expiration, the IP was no longer in the cache. Therefore, the ARP request was sent again.

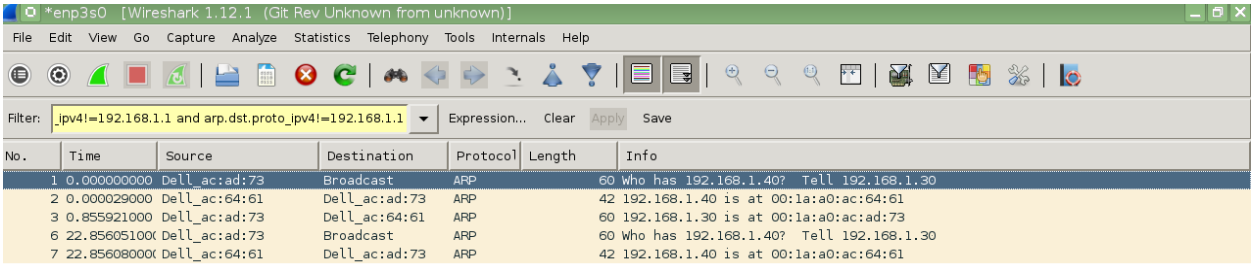

Figure 2: Wireshark screenshot of cache test

Comparing the reply from the computer and the reply from our program we can see that both replies are exactly the same.

output.txt

```
ï»¿Request

0000   00 1a a0 ac ad 73 00 1a a0 ac 64 61 08 06 00 01
0010   08 00 06 04 00 01 00 1a a0 ac 64 61 c0 a8 01 28
0020   00 1a a0 ac ad 73 c0 a8 01 1e

Reply 1

0000   00 1a a0 ac 64 61 00 1a a0 ac ad 73 08 06 00 01
0010   08 00 06 04 00 02 00 1a a0 ac ad 73 c0 a8 01 1e
0020   00 1a a0 ac 64 61 c0 a8 01 28 00 00 00 00 00 00


Reply 2

0000   00 1a a0 ac 64 61 00 1a a0 ac ad 73 08 06 00 01
0010   08 00 06 04 00 02 00 1a a0 ac ad 73 c0 a8 01 1e
0020   00 1a a0 ac 64 61 c0 a8 01 28 00 00 00 00 00 00




Request from program

0000   ff ff ff ff ff ff 00 1a a0 ac ad 73 08 06 00 01
0010   08 00 06 04 00 01 00 1a a0 ac ad 73 c0 a8 01 1e
0020   00 00 00 00 00 00 c0 a8 01 28 00 00 00 00 00 00


Reply

0000   00 1a a0 ac ad 73 00 1a a0 ac 64 61 08 06 00 01
0010   08 00 06 04 00 02 00 1a a0 ac 64 61 c0 a8 01 28
0020   00 1a a0 ac ad 73 c0 a8 01 1e

Found in table, send reply

0000   00 1a a0 ac 64 61 00 1a a0 ac ad 73 08 06 00 01
0010   08 00 06 04 00 02 00 1a a0 ac ad 73 c0 a8 01 1e
```

```
0020   00 1a a0 ac 64 61 c0 a8 01 28 00 00 00 00 00 00
```

---

3 Conclusion

ARP or Address Resolution Protocol is used to map IP addresses with data link layer addresses. A cache can be used to store this mapping and reduce the need to broadcast requests. This can speed up network communication. The cache can be cleaned out periodically so as to reduce memory usage. IP addresses used often will be cached repeatedly while IP addresses unused will be removed.

4 Appendix

```
#include "frameio.h"
#include "util.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <iostream>
#include <fstream>
#include <string>
#include <time.h>
#include <vector>
#include <fstream>

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
};


class ARP_Table
{
public:
  octet ip_addr[4];
  octet mac_addr[6];
  time_t timer;

  ARP_Table(octet ip[4], octet mac[6]) {
    memcpy(ip_addr, ip, 4);
    memcpy(mac_addr, mac, 6);
    time(&timer);
    //std::cout << "Timer value: " << timer << std::endl;
    //printf("Cached IP: %d.%d.%d.%d\n", ip_addr[0], ip_addr[1], ip_addr[2], ip_addr[3]);
    //printf("Cached MAC: %x.%x.%x.%x\n", mac_addr[0], mac_addr[1], mac_addr[2], mac_addr
    [3]);
  };

  bool is_ip(octet ip[4]) {
    //printf("Cached IP: %d.%d.%d.%d\n", ip_addr[0], ip_addr[1], ip_addr[2], ip_addr[3]);
```

```
    //printf("Cached MAC: %x.%x.%x.%x.%x.%x\n", mac_addr[0], mac_addr[1], mac_addr[2],
    mac_addr[3], mac_addr[4], mac_addr[5]);
    for (int i = 0; i < 4; i++) {
      if (ip_addr[i] != ip[i]) {
        return false;
      }
    }
    return true;
  }
};

std::vector<ARP_Table>cache_table;
octet local_addr[4];

//
// 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 timer_no = 1;

   // for fun, fire a timer each time we get a frame
   while ( 1 )
   {
      ip_queue.recv(&event, buf, sizeof(buf));
      if ( event != TIMER )
      {
         //printf("got an IP frame from %d.%d.%d.%d, queued timer %d\n",
         //         buf[12],buf[13],buf[14],buf[15],timer_no);
         ip_queue.timer(10,timer_no);
         timer_no++;
      }
      else
      {
         //printf("timer %d fired\n",*(int *)buf);
      }
   }
}

//
// Toy function to print something interesting when an ARP frame arrives
//
void *arp_protocol_loop(void *arg)
```

```
{
    octet buf[1500];
    event_kind event;

    const octet *local_mac = net.get_mac();
    //for (int i = 0; i < 6; i++)
    //   printf("%02x ", mac[i]);


    FILE *ph = popen("ifconfig enp3s0 | grep 'inet addr' | cut -d':' -f2 | cut -d' ' -f1", "r
    ");
    char local_addr_string[15];
    fgets(local_addr_string, sizeof(local_addr_string) - 1, ph);
    local_addr_string[14] = 0;
    pclose(ph);

    char *str = local_addr_string;
    char *end = str;
    local_addr[0] = strtol(str, &end, 10);
    while (*end == '.') end++;
    str = end;
    local_addr[1] = strtol(str, &end, 10);
    while (*end == '.') end++;
    str = end;
    local_addr[2] = strtol(str, &end, 10);
    while (*end == '.') end++;
    str = end;
    local_addr[3] = strtol(str, &end, 10);
    //printf("Local IP Address: %d.%d.%d.%d\n", local_addr[0], local_addr[1], local_addr[2],
     local_addr[3]);

    freopen("project2_output.txt", "w+", stdout);

    while ( 1 )
    {
        arp_queue.recv(&event, buf, sizeof(buf));
        printf("got an ARP %s\n", buf[7]==1? "request":"reply");

        octet ip[4];
        octet mac[6];
        memcpy(ip, &buf[14], 4);
        memcpy(mac, &buf[8], 6);

        //std::cout << "Prev table size: " << cache_table.size() << std::endl;
        bool found = false;
        for (int i = 0; i < cache_table.size(); i++) {
          if (cache_table[i].is_ip(ip)) {
            std::cout << "Found in table" << std::endl;
            found = true;
            break;
          }
        }
        if (!found) {
          std::cout << "Not found in table, adding for ip: " << (int)ip[3] << ", last mac: "
     << mac[5] << std::endl;
          ARP_Table entry = ARP_Table(ip, mac);
          cache_table.push_back(entry);
        }
        std::cout << "New table size: " << cache_table.size() << std::endl;

        if (buf[7] == 1)
        {
          printf("ARP Target IP: %d.%d.%d.%d\n", buf[24], buf[25], buf[26], buf[27]);
          printf("ARP Sender IP: %d.%d.%d.%d\n", buf[14], buf[15], buf[16], buf[17]);

          bool is_me = true;
    for (int i = 0; i < 4; i++) {
            if (buf[24 + i] != local_addr[i]) {
```

```
            is_me = false;
            break;
          }
        }

        ether_frame resp;

        if (!is_me) {
          continue;
        }
        else {
          //printf("Looking for me!\n");

          for (int i = 0; i < 6; i++)
          {
            resp.dst_mac[i] = buf[8 + i];
            resp.src_mac[i] = local_mac[i];
            // Sender's hardware address
            resp.data[8 + i] = local_mac[i];
            // Target hardware address
            resp.data[18 + i] = buf[8 + i];
          }
        }
        resp.prot[0] = 0x08;
        resp.prot[1] = 0x06;
        // hardware type (ethernet)
        resp.data[0] = 0x00;
        resp.data[1] = 0x01;
        // Protocol type (IPv4)
        resp.data[2] = 0x08;
        resp.data[3] = 0x00;
        // Hardware address length
        resp.data[4] = 0x06;
        // Protocol address length
        resp.data[5] = 0x04;
        // Opcode (2 = reply)
        resp.data[6] = 0x00;
        resp.data[7] = 0x02;

        for (int i = 0; i < 4; i++)
        {
          // Sender's IP
          resp.data[14 + i] = buf[24 + i];
          // Target IP
          resp.data[24 + i] = buf[14 + i];
        }

        net.send_frame(&resp, 42);



        //for (int i = 1; i < 60; i++)
          //printf("\t index: %d, value: 0x%x - %d\n", i, buf[i], buf[i]);

        //Is this me?
        //Find source address
        //Send response
      }
    }
}

void *cin_loop(void *arg) {
  const octet *mac = net.get_mac();

  std::cout << "Enter the target IP address: " << std::endl;
  while(1) {

    int read = 0;
```

```
    std::string str;
    octet input[4];
    while (read < 3 && std::getline(std::cin, str, '.') || read < 4 && std::getline(std::cin
    , str)) {
      input[read] = std::stoi(str);
      read++;
    }

    //std::cin >> (int)input[0] >> (int)input[1] >> (int)input[2] >> (int)input[3];
    std::cout << (int)input[0] << "." << (int)input[1] << "." << (int)input[2] << "." << (
    int)input[3] << std::endl;

    ether_frame resp;
    bool found_entry = false;
    for (int i = 0; i < cache_table.size(); i++) {
      ARP_Table target = cache_table[i];

      if (cache_table[i].is_ip(input)) {
        std::cout << "Found input in the table" << std::endl;
        //Send reply frame
        found_entry = true;

        for (int i = 0; i < 6; i++)
        {
          resp.dst_mac[i] = target.mac_addr[i];
          resp.src_mac[i] = mac[i];
          // Sender's hardware address
          resp.data[8 + i] = mac[i];
          // Target hardware address
          resp.data[18 + i] = target.mac_addr[i];
        }

        // Opcode (2 = reply)
        resp.data[6] = 0x00;
        resp.data[7] = 0x02;
      }
    }

    if (!found_entry)
    {
      std::cout << "Did not find input in the table, request address" << std::endl;
      for (int i = 0; i < 6; i++)
      {
        resp.dst_mac[i] = 0xFF;
        resp.src_mac[i] = mac[i];
        // Sender's hardware address
        resp.data[8 + i] = mac[i];
        // Target hardware address
        resp.data[18 + i] = 0;
      }

      //Send request frame
      // Opcode (1 = request)
      resp.data[6] = 0x00;
      resp.data[7] = 0x01;
    }

    resp.prot[0] = 0x08;
    resp.prot[1] = 0x06;
    // hardware type (ethernet)
    resp.data[0] = 0x00;
    resp.data[1] = 0x01;
    // Protocol type (IPv4)
    resp.data[2] = 0x08;
    resp.data[3] = 0x00;
    // Hardware address length
    resp.data[4] = 0x06;
    // Protocol address length
```

```
    resp.data[5] = 0x04;

    for (int i = 0; i < 4; i++)
    {
      // Sender's IP
      resp.data[14 + i] = local_addr[i];
      // Target IP
      resp.data[24 + i] = input[i];
    }

    net.send_frame(&resp, 42);
    std::cout << "--SEND FRAME--" << std::endl;
  }
}

void *time_loop(void *arg)
{
  while(1)
  {
    sleep(1);

    time_t timer;
    time(&timer);
    int i = 0;
    while (i < cache_table.size()) {
      if (timer - cache_table[i].timer > 20) {
        std::cout << "Timer removed an item from the cache table" << std::endl;
        std::cout << "Removed item with IP: " << (int)cache_table[i].ip_addr[0] << "." << (
    int)cache_table[i].ip_addr[1] << "." << (int)cache_table[i].ip_addr[2] << "." << (int)
    cache_table[i].ip_addr[3] << std::endl;

        cache_table.erase(cache_table.begin() + i);
      }
      else {
        i++;
      }
    }
  }
}

//
// if you're going to have pthreads, you'll need some thread descriptors
//
pthread_t loop_thread, cin_thread, arp_thread, ip_thread, timer_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);
   pthread_create(&arp_thread,NULL,arp_protocol_loop,NULL);
   pthread_create(&cin_thread,NULL,cin_loop,NULL);
   pthread_create(&ip_thread,NULL,ip_protocol_loop,NULL);
   pthread_create(&timer_thread,NULL,time_loop,NULL);
   for ( ; ; )
      sleep(1);
}
```