Project 5 Report
ECE 5600

Nathan Tipton
A01207112
Partner: Erik Sargent

Dec 1, 2017

1 Objective

The purpose of this project is to familiarize ourselves with the User Datagram Protocol (UDP). We will also implement IP fragmentation and defragmentation to break up larger packets that are greater than the max packet size.

2 Results

We implemented the IP fragmentation and defragmentation in our code. This allows larger packets to be broken up and sent via UDP. Figure 1 shows a wireshark screenshot of our code working with UDP echo and IP fragmentation.

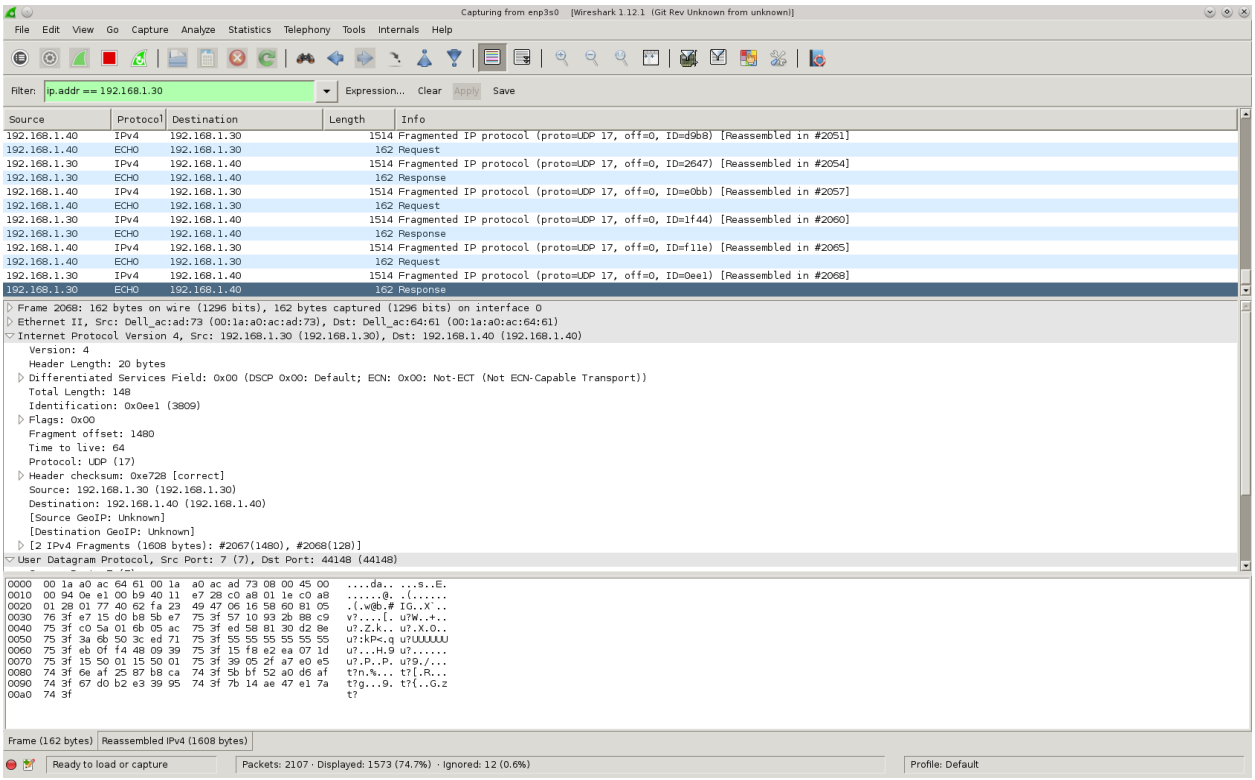

Figure 1: Wireshark screenshot of fragmentation and UDP echo

2.1 UDP Packet headers

REQUEST1
0000 00 1a a0 ac ad 73 00 1a a0 ac 64 61 08 00 45 00
0010 00 94 64 e9 00 b9 40 11 91 20 c0 a8 01 28 c0 a8
0020 01 1e 40 62 01 77 fa 23 76 3f 06 16 58 60 81 05

RESPONSE1
0000 00 1a a0 ac 64 61 00 1a a0 ac ad 73 08 00 45 00
0010 00 94 9b 16 00 b9 40 11 5a f3 c0 a8 01 1e c0 a8
0020 01 28 01 77 40 62 fa 23 49 47 06 16 58 60 81 05

REQUEST2
0000 00 1a a0 ac ad 73 00 1a a0 ac 64 61 08 00 45 00
0010 00 94 6e 38 00 b9 40 11 87 d1 c0 a8 01 28 c0 a8
0020 01 1e 40 62 01 77 fa 23 76 3f 06 16 58 60 81 05

RESPONSE2
0000 00 1a a0 ac 64 61 00 1a a0 ac ad 73 08 00 45 00
0010 00 94 91 c7 00 b9 40 11 64 42 c0 a8 01 1e c0 a8
0020 01 28 01 77 40 62 fa 23 49 47 06 16 58 60 81 05

3 Conclusion

UDP is a connectionless service. Packets of data, datagrams, are sent to ports on the computer. There is a destination port and source port. Sending to port 7 will use the UDP echo.

4 Appendix

i

```
#include "util.h"
#include "chksum.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 <array>
#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;
        }
    }
}

struct Arr {
    octet buf[1500];
    Arr(octet *b) {
        memcpy(buf, b, 1500);
    }
};

//
```

```
// 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;

    const octet *local_mac = net.get_mac();

    std::vector<Arr> bufs;

    // for fun, fire a timer each time we get a frame
    while ( 1 )
    {
        ip_queue.recv(&event, buf, sizeof(buf));

        // UDP
        if (buf[9] = 0x11) {
            Arr arr(buf);
            bufs.push_back(arr);
            printf("flags: %x\n", buf[6] >> 5);
            // std::cout << "Received UDP packet to port " << ((buf[22] << 8) | buf[23]) <<
    std::endl;
            // More data
            if ((buf[6] >> 5) == 0x01) {
                std::cout << "Waiting for more data" << std::endl;
            }
            //else if (buf[23] == 7) {
            if ((buf[6] >> 5) == 0x00) {
                std::cout << "No longer waiting for data" << std::endl;
                printf("Target packet id: %x %x\n", buf[4], buf[5]);
                bool port_confirmed = false;
                for (int i = 0; i < bufs.size(); i++) {
                    if (bufs[i].buf[23] == 7)
                        port_confirmed = true;
                }
                if (!port_confirmed) {
                    std::cout << "Data was not being sent to port 7" << std::endl;
                    continue;
                }

                std::cout << "Transmit everything, buf size: " << bufs.size() << std::endl;

                for (int i = 0; i < bufs.size(); i++) {
                    if (bufs[i].buf[4] != buf[4] || bufs[i].buf[5] != buf[5]) {
                        printf("ID Doesn't match %x %x\n", bufs[i].buf[4], bufs[i].buf[5]);
                        continue;
                    }

                    std::cout << "Sending packet" << std::endl;

                    octet *buf = bufs[i].buf;
                    ether_frame frame;

                    frame.prot[0] = 0x08;
                    frame.prot[1] = 0x00;

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

                        if (cache_table[i].is_ip(&buf[12])) {
                            found = true;

                            for (int i = 0; i < 6; i++)
                            {
                                frame.dst_mac[i] = target.mac_addr[i];
```

```
                frame.src_mac[i] = local_mac[i];
            }
        }
    }
    if (!found) {
        std::cout << "ARP entry not found" << std::endl;
        continue;
    }

    memcpy(&frame.data[0], &buf[0], 10);
//    printf("buf[0]: %x, data[0]: %x\n", buf[0], frame.data[0]);

    // Identification
    frame.data[4] = ~buf[4];
    frame.data[5] = ~buf[5];

    for (int i = 0; i < 4; i++)
    {
        // Sender's IP
        frame.data[12 + i] = local_addr[i];
        // Target IP
        frame.data[16 + i] = buf[12 + i];
    }

    int hc = chksum(frame.data, 10, 0);
    hc = chksum(&frame.data[12], 8, hc);
    hc = ~hc;

    // Checksum
    frame.data[10] = (hc >> 8) & 0xFF;
    frame.data[11] = hc & 0xFF;

    // UDP Specific
    // Source port
    frame.data[20] = buf[22];
    frame.data[21] = buf[23];

    // Destination port
    frame.data[22] = buf[20];
    frame.data[23] = buf[21];

    // Length
    frame.data[24] = buf[24];
    frame.data[25] = buf[25];

    memcpy(&frame.data[28], &buf[28], 1472);

    int dc = chksum(&frame.data[12], 8, 0);

    frame.data[8] = 0x00;
    dc = chksum(&frame.data[8], 2, dc);
    frame.data[8] = 0x40;

    dc = chksum(&frame.data[24], 2, dc);
    dc = chksum(&frame.data[20], 6, dc);

    dc = chksum(&frame.data[28], 1472, dc);

    dc = chksum(&frame.data[28], 280, dc);

    dc = ~dc;

    // Checksum
    frame.data[26] = (dc >> 8) & 0xFF;
    frame.data[27] = dc & 0xFF;

    for (int i = 0; i < 28; i++)
        printf("%d: %x\n", i, frame.data[i]);
```

```
                    std::cout << "Sending bytes: " << ((buf[2] << 8) | buf[3]) << std::endl;

                    net.send_frame(&frame, ((buf[2] << 8) | buf[3]) + 14);

                    std::cout << "--SEND UDP request, seq: " << i << "--" << std::endl;

                }

                //bufs.clear();
            }
        }
    }
}

//
// 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();

    FILE *ph = popen("ifconfig enp3s0 | sed -rn '2s/ .*:(.*)$/\\1/p'", "r");
    char mask_string[15];
    fgets(mask_string, sizeof(mask_string) - 1, ph);
    mask_string[14] = 0;
    pclose(ph);

    octet mask_addr[4];
    char *mask_str = mask_string;
    char *end = mask_str;
    mask_addr[0] = strtol(mask_str, &end, 10);
    while (*end == '.') end++;
    mask_str = end;
    mask_addr[1] = strtol(mask_str, &end, 10);
    while (*end == '.') end++;
    mask_str = end;
    mask_addr[2] = strtol(mask_str, &end, 10);
    while (*end == '.') end++;
    mask_str = end;
    mask_addr[3] = strtol(mask_str, &end, 10);
    std::cout << (int)mask_addr[0] << "." << (int)mask_addr[1] << "." << (int)mask_addr[2]
    << "." << (int)mask_addr[3] << std::endl;


    std::cout << "Enter the target IP address: " << std::endl;
    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;
    std::cout << (int)(input[0] & mask_addr[0]) << "." << (int)(input[1] & mask_addr[1]) <<
    "." << (int)(input[2] & mask_addr[2]) << "." << (int)(input[3] & mask_addr[3]) << std::
    endl;

    bool local_network = false;
    if (((input[0] & mask_addr[0]) == (local_addr[0] & mask_addr[0])) &&
            ((input[1] & mask_addr[1]) == (local_addr[1] & mask_addr[1])) &&
            ((input[2] & mask_addr[2]) == (local_addr[2] & mask_addr[2])) &&
            ((input[3] & mask_addr[3]) == (local_addr[3] & mask_addr[3]))) {
        local_network = true;
    }
    std::cout << (local_network ? "Local" : "Not Local") << std::endl;

    octet gateway[4] = { 192, 168, 1, 1 };
    octet *dest_addr;
    if (local_network)
        dest_addr = input;
```

```
else
    dest_addr = gateway;

int seq = 1;
int id = rand() & 0xFFFF;

while(1) {
    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(dest_addr)) {
            std::cout << "Found input in the table" << std::endl;
            //Send reply frame
            found_entry = true;

            ether_frame frame;

            frame.prot[0] = 0x08;
            frame.prot[1] = 0x00;

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

            // IP Version + IHL
            frame.data[0] = 0x45;

            // Diff serivces
            frame.data[1] = 0x00;

            // Total length
            if (seq != 0) {
                frame.data[2] = 0x05;
                frame.data[3] = 0xDC;
            }
            else {
                frame.data[2] = 0x01;
                frame.data[3] = 0x2C;
            }

            // Identification
            frame.data[4] = id >> 8;
            frame.data[5] = id;

            // Fragment
            if (seq == 0) {
                frame.data[6] = 0x00;
                frame.data[7] = 0xB9;
            }
            else {
                frame.data[6] = 0x20;
                frame.data[7] = 0x00;
            }

            // TTL
            frame.data[8] = 0x40;

            // Protocol (UDP)
            frame.data[9] = 0x11;

            for (int i = 0; i < 4; i++)
```

```
            {
                // Sender's IP
                frame.data[12 + i] = local_addr[i];
                // Target IP
                frame.data[16 + i] = input[i];
            }

            int hc = chksum(frame.data, 10, 0);
            hc = chksum(&frame.data[12], 8, hc);
            hc = ~hc;

            // Checksum
            frame.data[10] = (hc >> 8) & 0xFF;
            frame.data[11] = hc & 0xFF;

            // UDP Specific
            if (seq != 0) {
                // Source port
                frame.data[20] = 0x19;
                frame.data[21] = 0x64;

                // Destination port
                frame.data[22] = 0x00;
                frame.data[23] = 0x07;

                // Length
                frame.data[24] = 0x06;
                frame.data[25] = 0xF4;

                for (int i = 28; i < 1500; i++)
                    frame.data[i] = 0x55;

                int dc = chksum(&frame.data[12], 8, 0);

                frame.data[8] = 0x00;
                dc = chksum(&frame.data[8], 2, dc);
                frame.data[8] = 0x40;

                dc = chksum(&frame.data[24], 2, dc);
                dc = chksum(&frame.data[20], 6, dc);

                dc = chksum(&frame.data[28], 288, dc);
                dc = chksum(&frame.data[28], 1472, dc);

                dc = ~dc;

                // Checksum
                frame.data[26] = (dc >> 8) & 0xFF;
                frame.data[27] = dc & 0xFF;

                net.send_frame(&frame, 1500);
            }
            else {
                for (int i = 20; i < 300; i++)
                    frame.data[i] = 0x55;

                net.send_frame(&frame, 300);
            }

            std::cout << "--SEND UDP request, seq: " << seq << "--" << std::endl;

            seq--;

            if (seq < 0)
                return NULL;
        }
    }
```

```
        if (!found_entry)
        {
            std::cout << "Did not find input in the table, request address" << std::endl;

            ether_frame resp;

            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] = dest_addr[i];
            }

            net.send_frame(&resp, 42);
            std::cout << "--SEND FRAME--" << std::endl;
            sleep(1);
        }

    }
}

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 > 200) {
                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);
}
```