**Report for Stripping IP Packets**

2016311410 컴퓨터공학과 권준희

이번 과제를 통해서 교안으로만 배웠던 IP 패킷을 직접 분석해보았다. Wireshark 프로그램을 통해 특정 와이파이를 이용하여 전송된 패킷들을 수집할 수 있었다. 다양한 프로그램으로부터 TCP, UDP 패킷을 캡쳐하기 위해서 스마트폰을 PC와 같은 와이파이에 연결해서 패킷을 분석해보았다.

1. iPhone에서 Instagram을 가동하고 타임라인을 새로고침할 때 UDP 패킷, 다양한 사람들의 계정에 들어갔을 때 TCP 패킷을 캡쳐할 수 있었다.
2. 카카오톡으로 메시지를 전송했을 때는 TCP 패킷을 캡쳐할 수 있었다.
3. Window PC의 크롬 브라우저로 구글에 접속했을 때, UDP 패킷을 대량 캡쳐했다.
4. iPhone에서 Netflix의 동영상을 재생하고, 재생을 종료하는 순간, MDNS(UDP)라는 프로토콜을 사용하는 패킷을 캡쳐했고, 동영상이 재생되는 동안에는 다량의 TCP 패킷이 발생되었다. MDNS 프로토콜에 대해 찾아보니 애플사에서 사용하는 DNS와 비슷한 프로토콜이라는 것을 알 수 있었다.
5. Twitter 어플리케이션에서 글과 사진을 보고 리트윗 등을 하는 경우 TCP패킷을 전송하는 것을 확인할 수 있었다.
6. 네이버 파파고 어플리케이션도 TCP 패킷을 전송했다.
7. 패킷의 stream을 확인하다보니 거실 TV에 연결되어있는 크롬캐스트의 패킷도 캡쳐할 수 있었다.크롬 캐스트는 UDP 프로토콜을 사용했다.
8. Youtube music은 TCP를 사용했다.
9. 학교의 인의예지 클러스터에 접속한 경우에도 TCP 패킷을 얻을 수 있었다.

패킷을 수집한 후에 (udp || tcp || icmp) && !ipv6 라는 필터를 사용해서 원하는 패킷만 따로 파일로 저장할 수 있었다. 캡쳐한 패킷 파일을 분석하기 위한 소스코드는 아래와 같다.

Source code

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <math.h>

#include <WinSock2.h>

#include <time.h>

#pragma comment(lib,"ws2\_32")

#define MAX\_PACKET 500

#define MAC\_ADDR 6

#define DF(frag) (frag & 0x40)

#define MF(frag) (frag & 0x20)

#define FRAG\_OFFSET(frag) (ntohs(frag) & (~0x6000))

//Structs

typedef struct pcapHeader {

int magic;

short major;

short minor;

int time\_zone;

int time\_stamp;

int snap\_len;

int link\_type;

}pcap\_H;

typedef struct Timeval\_ {

long val\_sec;

long val\_usec;

}Timeval;

typedef struct pktHeader\_ {

Timeval time;

unsigned int caplen;

unsigned int len;

}pkt\_H;

typedef struct Mac {

unsigned char MAC\_DST[MAC\_ADDR];

unsigned char MAC\_SRC[MAC\_ADDR];

unsigned short type;

}Mac;

typedef struct IP {

unsigned char ver\_hlen;

unsigned char ecn;

unsigned short tot;

unsigned short id;

unsigned short frag;

unsigned char ttl;

unsigned char protocol;

unsigned short checksum;

unsigned char src\_ip[4];

unsigned char dst\_ip[4];

}IP;

//Functions

void Parse(FILE\* fp);

void Parse\_strt(FILE\* fp);

void showFileHeader(pkt\_H\* ph);

void showMac(Mac mac);

unsigned short ntohs\_(unsigned short value);

void showIP(IP ip);

//전역

pkt\_H packetHeader[MAX\_PACKET];

int packetCount = 0;

int main() {

FILE\* fp;

fp = fopen("packet3.pcap", "rb");

Parse\_strt(fp);

fclose(fp);

return 0;

}

void Parse\_strt(FILE\* fp) {

//pcap의 쓸모없는 부분(24byte) 제거 -> 각 패킷의 TCP 헤더(MAC, Type) 읽기

pcap\_H noUse;

fread(&noUse, sizeof(pcap\_H), 1, fp);

Parse(fp);

}

void Parse(FILE\* fp) {

pkt\_H\* ph = packetHeader;

//fp가 끝날 때까지 읽기

while (feof(fp) == 0) {

if ((fread(ph, sizeof(pkt\_H), 1, fp) != 1))

break;

if (packetCount == MAX\_PACKET)

break;

//패킷이 아직 있는 경우 -> time, caplen, actual len 출력

showFileHeader(ph);

Mac mac;

fread(&mac, sizeof(mac), 1, fp);

showMac(mac);

char tmpIP[65536];

//caplen-(mac address and type which are 14 bytes)

fread(tmpIP, ph->caplen - 14, 1, fp);

IP\* ip = (IP\*)tmpIP;

showIP(\*ip);

}

}

void showFileHeader(pkt\_H\* ph) {

packetCount++;

time\_t rawtime = ph->time.val\_sec;

//time\_t rawtime2 = ph->time.val\_usec;

struct tm ts;

//struct tm ts2;

char buf[80];

//char buf2[80];

// Format time, "ddd yyyy-mm-dd hh:mm:ss zzz"

ts = \*localtime(&rawtime);

strftime(buf, sizeof(buf), "%a %Y-%m-%d %H:%M:%S", &ts);

//printf("%s\n\n", buf);

printf("\n\n<Packet %d> \nLocal Time: %s.%08d\nCaptured Packet Length: %u bytes, Actual Packet Length: %u bytes\n",

packetCount,buf, ph->time.val\_usec, ph->caplen, ph->len);

}

void showMac(Mac mac) {

int s, d;

//src

printf("SRC MAC address: ");

for (s = 0; s < MAC\_ADDR - 1; s++)

printf("%02x:", mac.MAC\_SRC[s]);

printf("%02x -> ", mac.MAC\_SRC[s]);

//dst

printf("DST MAC address: ");

for (d = 0; d < MAC\_ADDR - 1; d++)

printf("%02x:", mac.MAC\_DST[d]);

printf("%02x\n", mac.MAC\_DST[d]);

}

void showIPaddr(IP ip) {

//src

int s, d;

printf("SRC IP address : ");

for (s = 0; s < 3; s++)

printf("%u.", ip.src\_ip[s]);

printf("%u -> ", ip.src\_ip[s]);

//dst

printf("DST IP address : ");

for (d = 0; d < 3; d++)

printf("%u.", ip.dst\_ip[d]);

printf("%u\n", ip.dst\_ip[d]);

}

void ver\_hlen(unsigned char ver\_hlen) {

printf("Ver: %x, ", ver\_hlen >> 4);

unsigned char tmp = ver\_hlen << 4;//하위 4비트를 상위 4비트로 올렸다가

printf("HLEN in header: %d byte\n", (tmp >> 4) \* 4);//상위 4비트를 다시 하위 4비트로 내림 0000xxxx

}

void showIP(IP ip) {

showIPaddr(ip);//IP address

printf("Total LEN: %u byte, ", ntohs(ip.tot)); //Total Length

printf(" TTL: %d\n", ip.ttl); //Time to Live

ver\_hlen(ip.ver\_hlen);//Version and HLEN

printf("Id: %d, ", ntohs(ip.id)); //Identification

//Flag

if (DF(ip.frag))

printf("DF=1, ");

else {

if (MF(ip.frag) == 0)

printf("DF=0 and MF=0, ");

else

printf("DF=0 and MF=1, ");

}

printf("Fragment Offset: %d\n", 8 \* (ntohs(ip.frag) & 0x1fff));

//Protocol

switch (ip.protocol) {

case 1: printf("Protocol: ICMP. "); break;

case 2: printf("Protocol: IGMP. "); break;

case 6: printf("Protocol: TCP. "); break;

case 17: printf("Protocol: UDP. "); break;

case 89: printf("Protocol: OSPF. "); break;

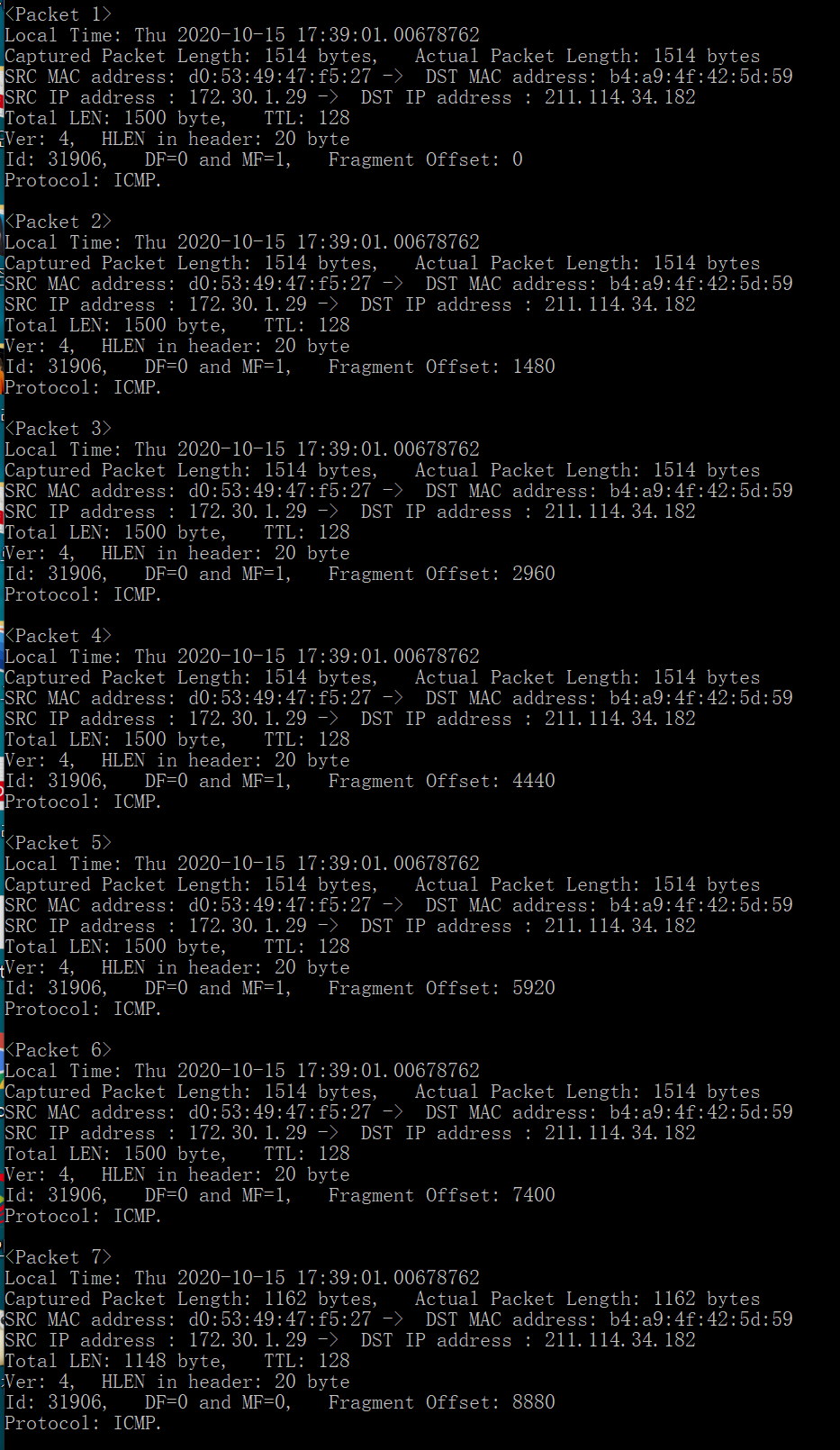
default: printf("This protocol is not supported. "); break;

}

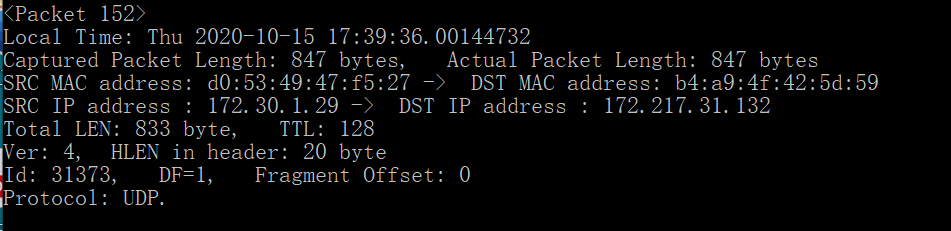
}

Output in terminal

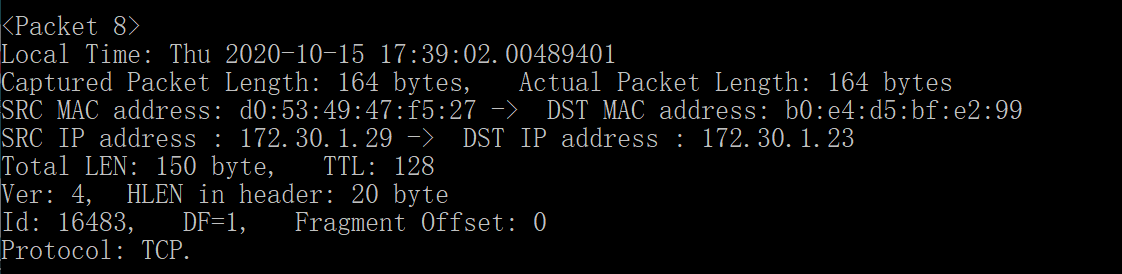
<Frame 1-7 : ICMP packet>



<Frame 152:UDP packet>

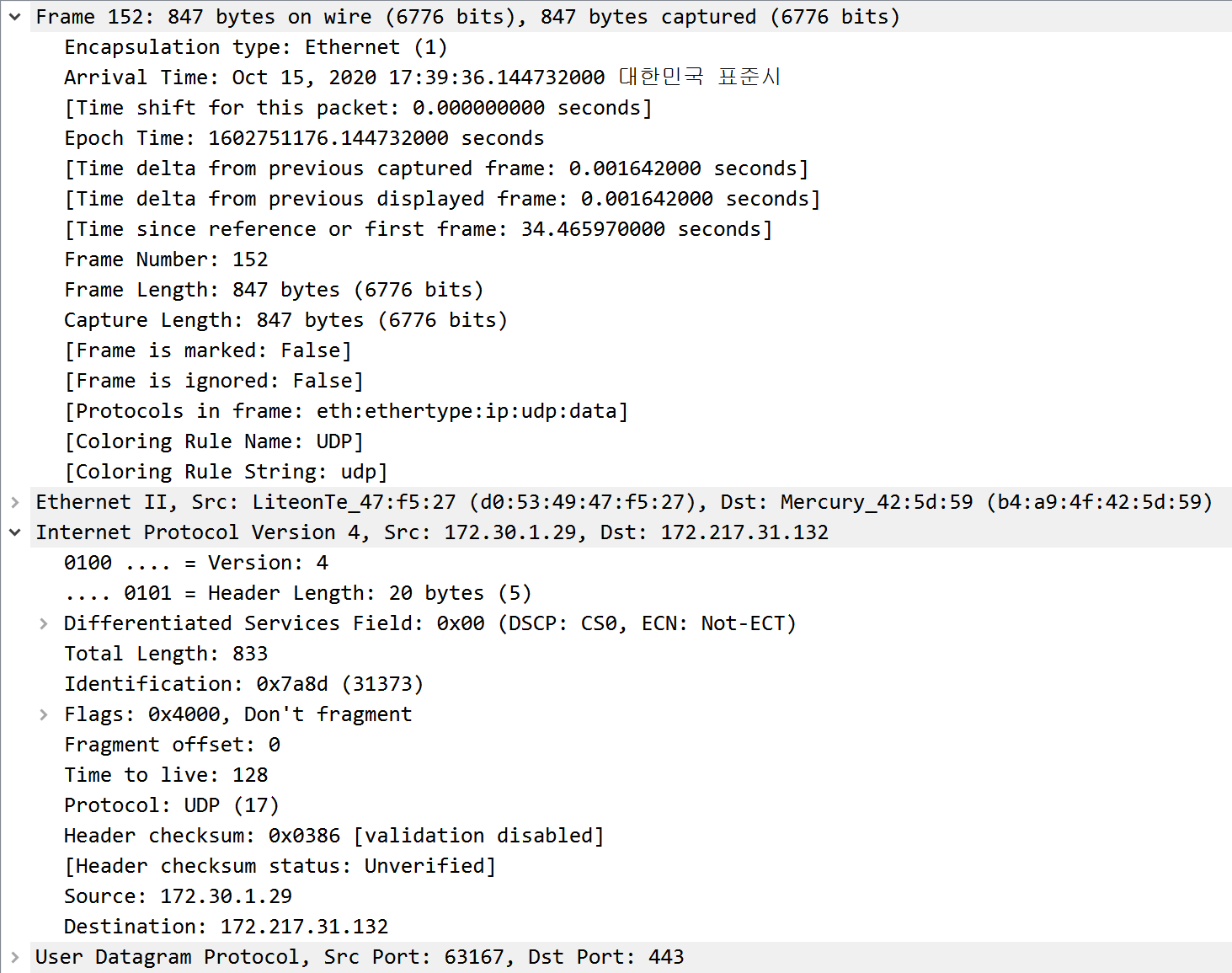


<Frame 8 : TCP packet>

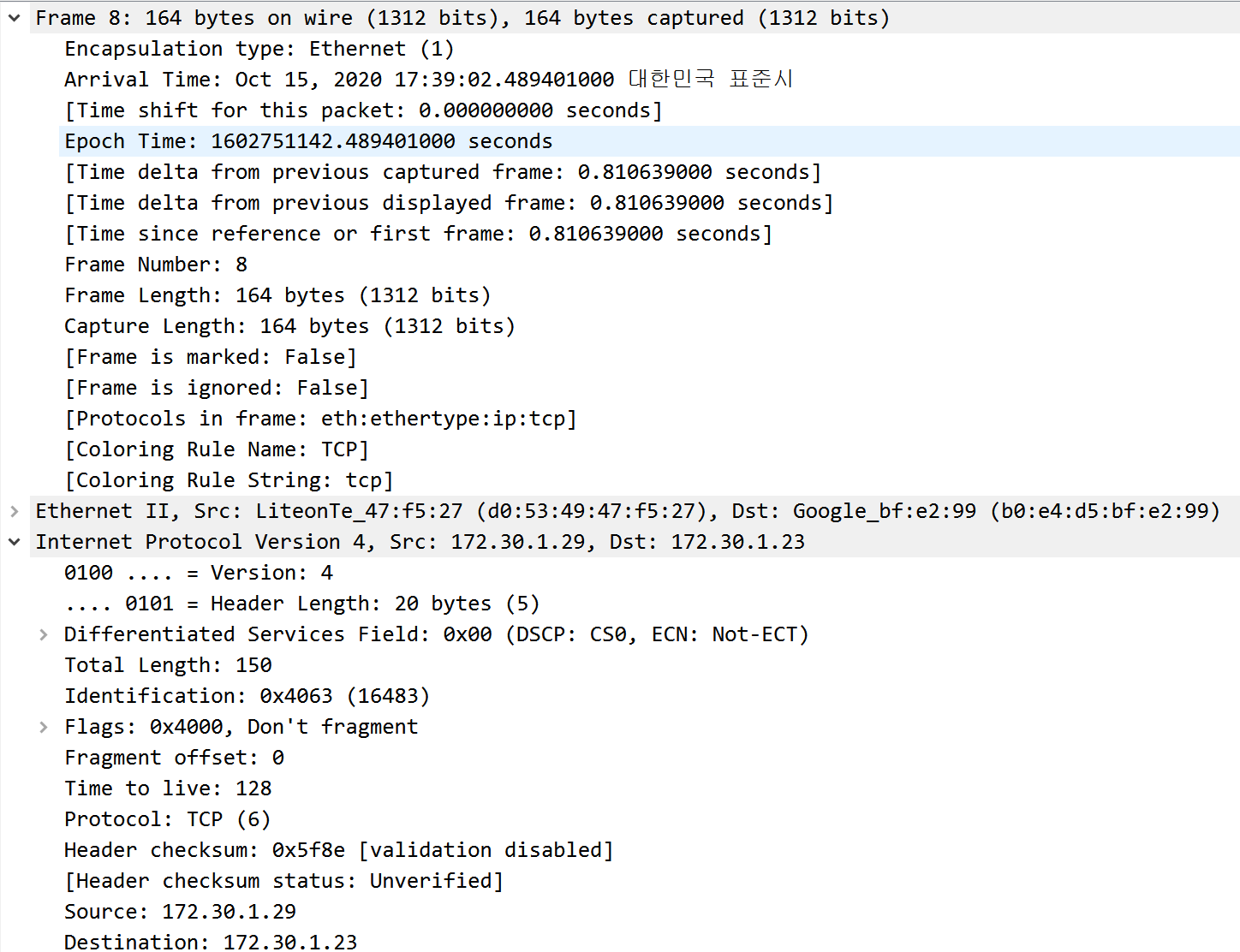


Verification with Wireshark

<Packet 152 :UDP packet>

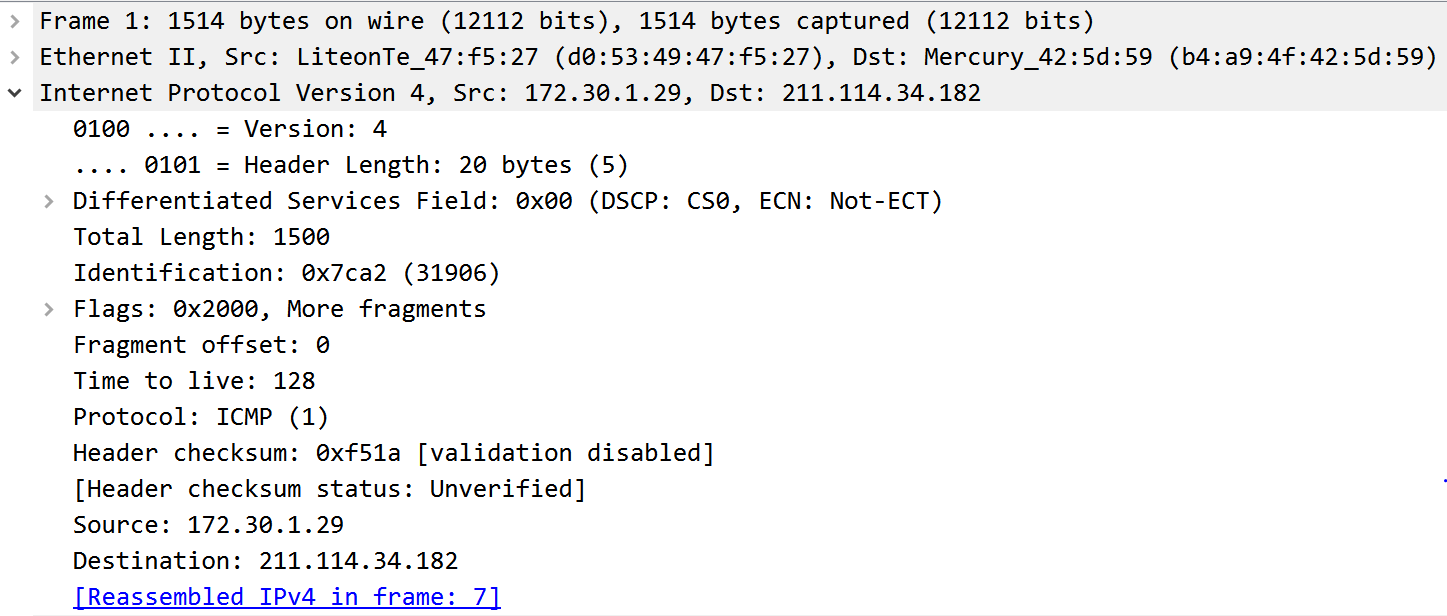


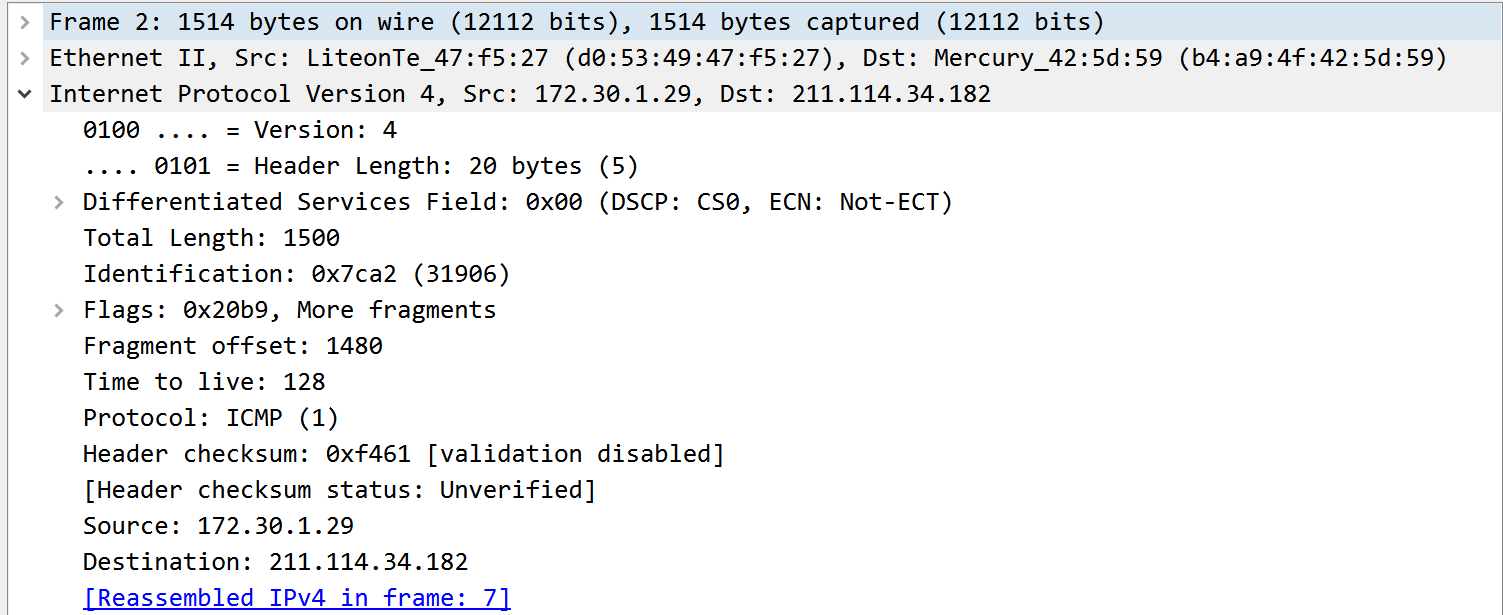
<Packet 8 : TCP packet>

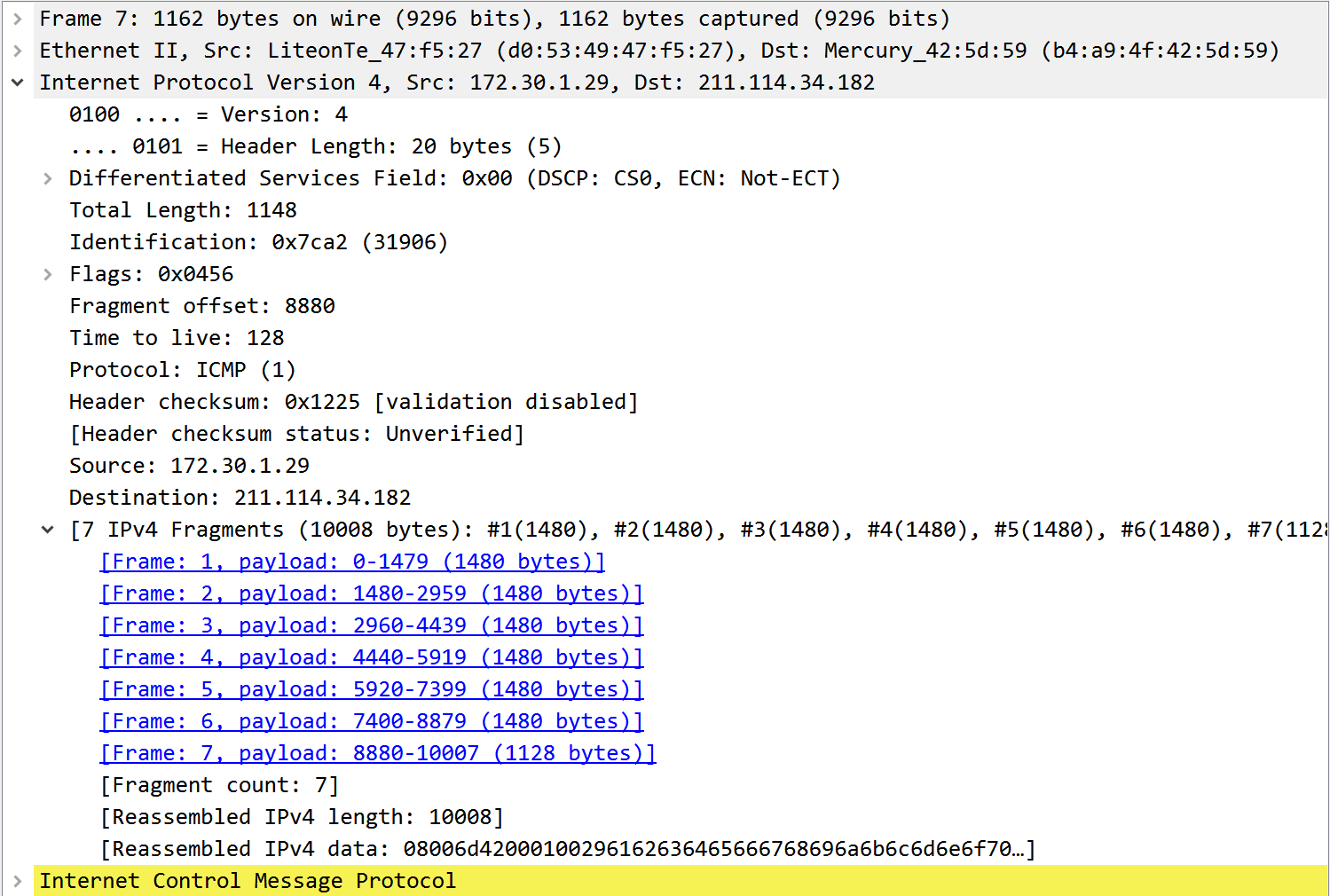


위의 소스를 통해서 packet 152와 packet8를 분석한 결과와 Wireshark를 비교해보았을 때, 같은 결과를 보여주는 것을 확인할 수 있었다.

<Packet 1-7 : ICMP packet>







Packet 1-7을 하나로 묶은 것은 7개의 패킷들이 하나의 Data chunk에서 조각난 패킷들이기 때문이다. 4개 패킷의 ID를 확인했을 때, 모두 동일하게 0x7ca2(31906)임을 확인할 수 있다. 프로젝트를 진행한 로컬 컴퓨터의 라우팅 테이블에서 IP주소 211.114.34.182로 10000 byte짜리 ping을 보냈을 때의 결과이다. MTU의 사이즈는 1500byte이기 때문에 4개의 패킷에 걸쳐서 10000byte를 전송했다. 7개의 패킷을 살펴보면 앞의 2개의 패킷의 Flag는 DF=0, MF=1이고, 마지막 패킷은 MF=0으로 해당 Data chunk의 마지막 패킷임을 표현하고 있다. 그리고 offset을 살펴보았을 때, 1480byte씩 떨어져 있는 것을 알 수 있다. 그것은 Fragment당 1500 byte이고, 패킷 헤더가 20byte이기 때문에 실제 전송하려던 데이터는 1480 byte씩 하나의 패킷에 담겼기 때문이다.

쪼개진 패킷들은 오직 Destination에서만 다시 조립된다. Packet 1-6의 가장 아래의 메시지를 보면 *‘Reassembled IPv4 in frame : 7’* 라고 적혀 있는 것을 확인할 수 있다. 그리고 packet 7의 마지막을 살펴보면 #1 #2 #3 #4 #5 #6 #7 패킷들이 다시 조립되어 10008byte 길이의 데이터를 얻었다는 문구를 확인할 수 있다.

Question

패킷 분석 결과를 확인하면서 궁금했던 점은 ‘왜 마지막 패킷의 데이터 크기가 1128 byte인지’ 이다. CMD를 통해서 분명히 10000byte를 전송했는데 reassemble하고 난 후의 데이터 크기는 10008이 되었다. 10000을 MTU가 1500인 7개의 패킷을 통해서 전송하면 각각 1480, 1480, 1480, 1480, 1480, 1480, 1120 byte이고, reassemble하면 10000이 되어야 한다고 생각했는데 결과가 그렇지 않았다. 다시 Wireshark를 통해 capture을 진행했을 때도 동일한 결과였다.

처음엔 각 패킷에 새롭게 추가된 헤더의 size 때문인가 생각해보았지만, 그것은 아니었다. 왜냐하면 10008byte라는 최종 크기는 각 패킷의 헤더를 제외한 데이터인 1480\*6 + 1128의 결과이기 때문이다.