**Computer Network HW1**

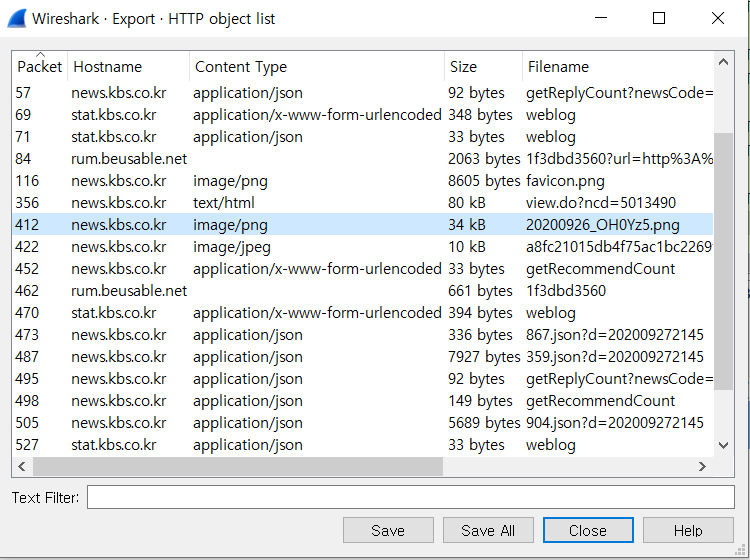
2016147008 조성현

**1. Wireshark를 이용한 Packet Capturing**

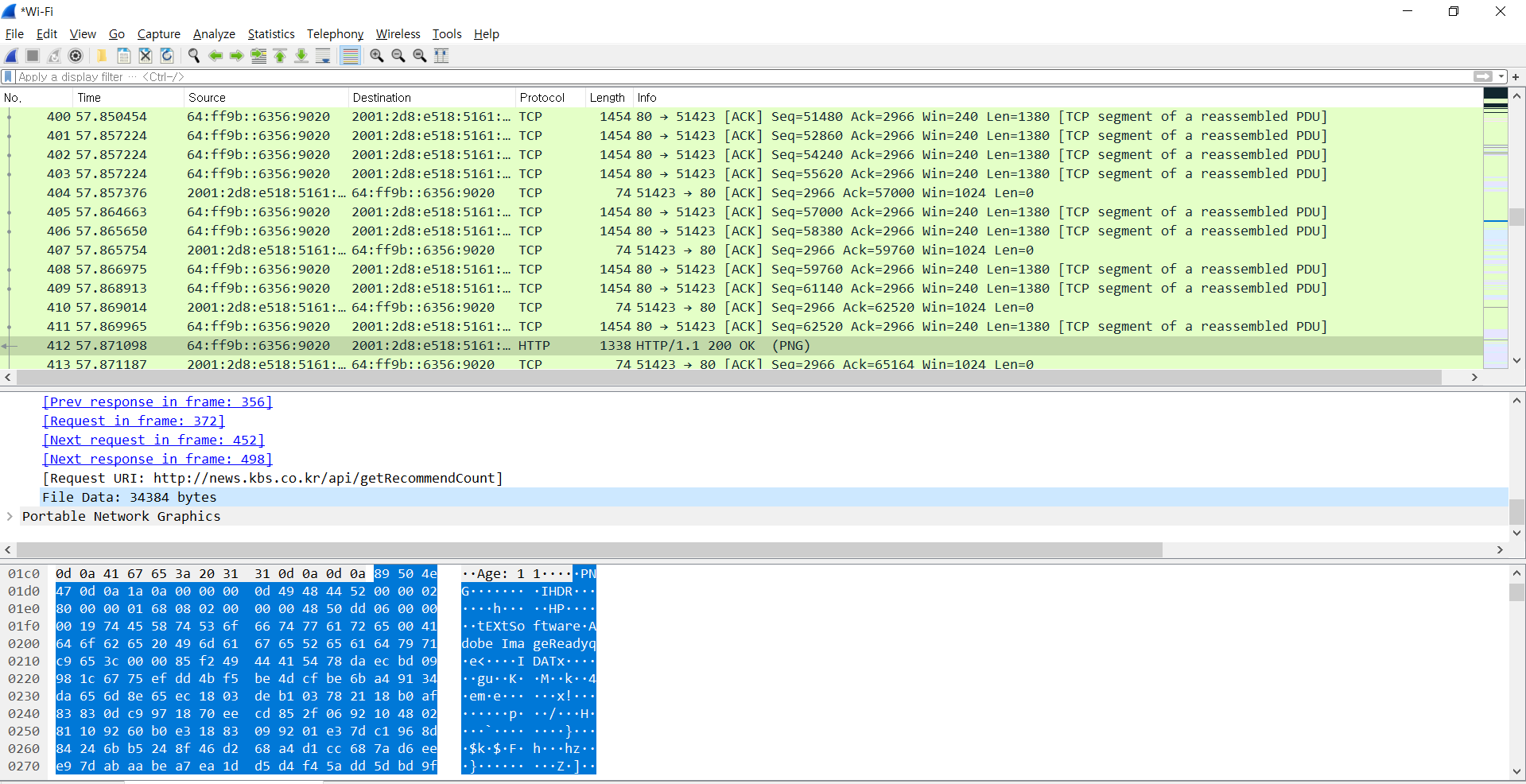
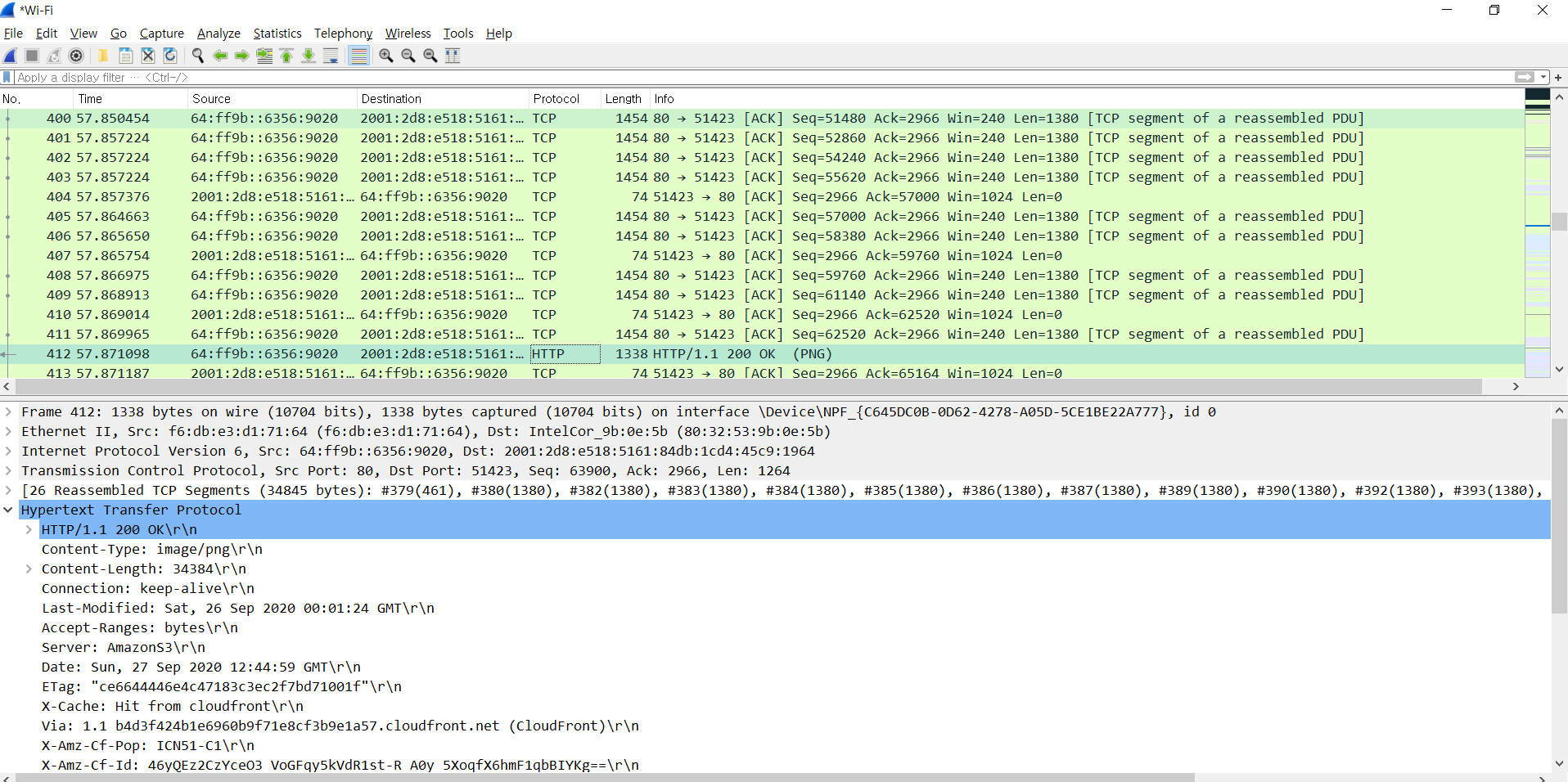


<http://news.kbs.co.kr/news/view.do?ncd=5013490>

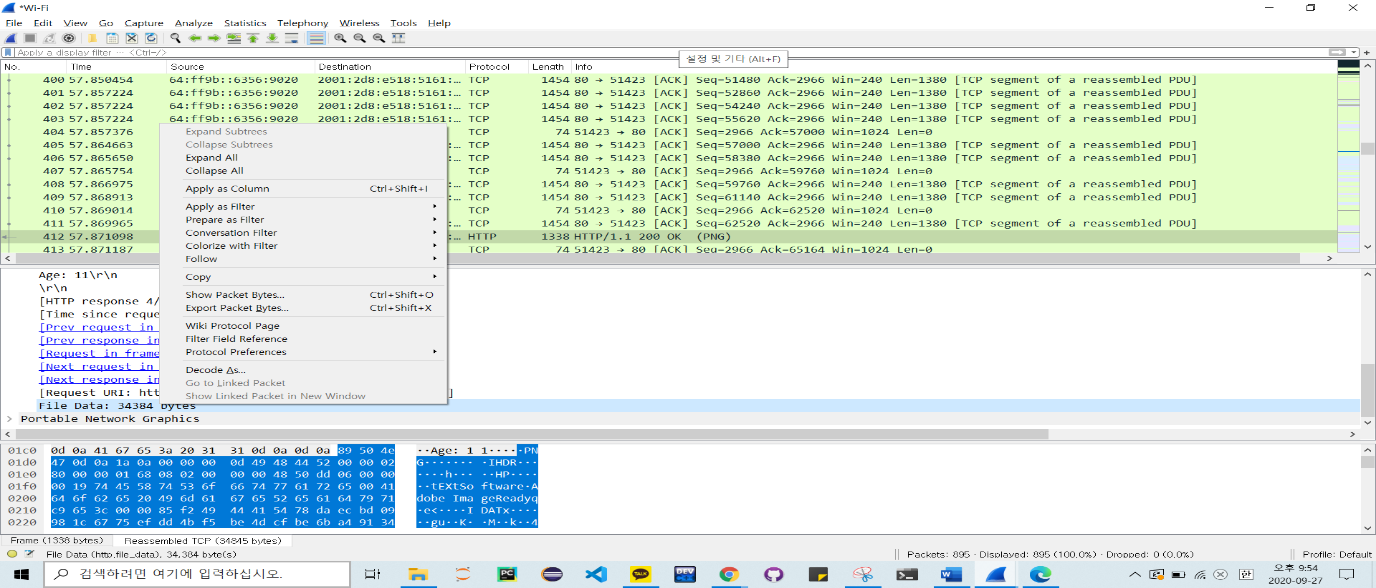
**Wireshark를 이용하여 위의 Http 주소에 있는 그림 이미지를 Export하려고 한다.**

****

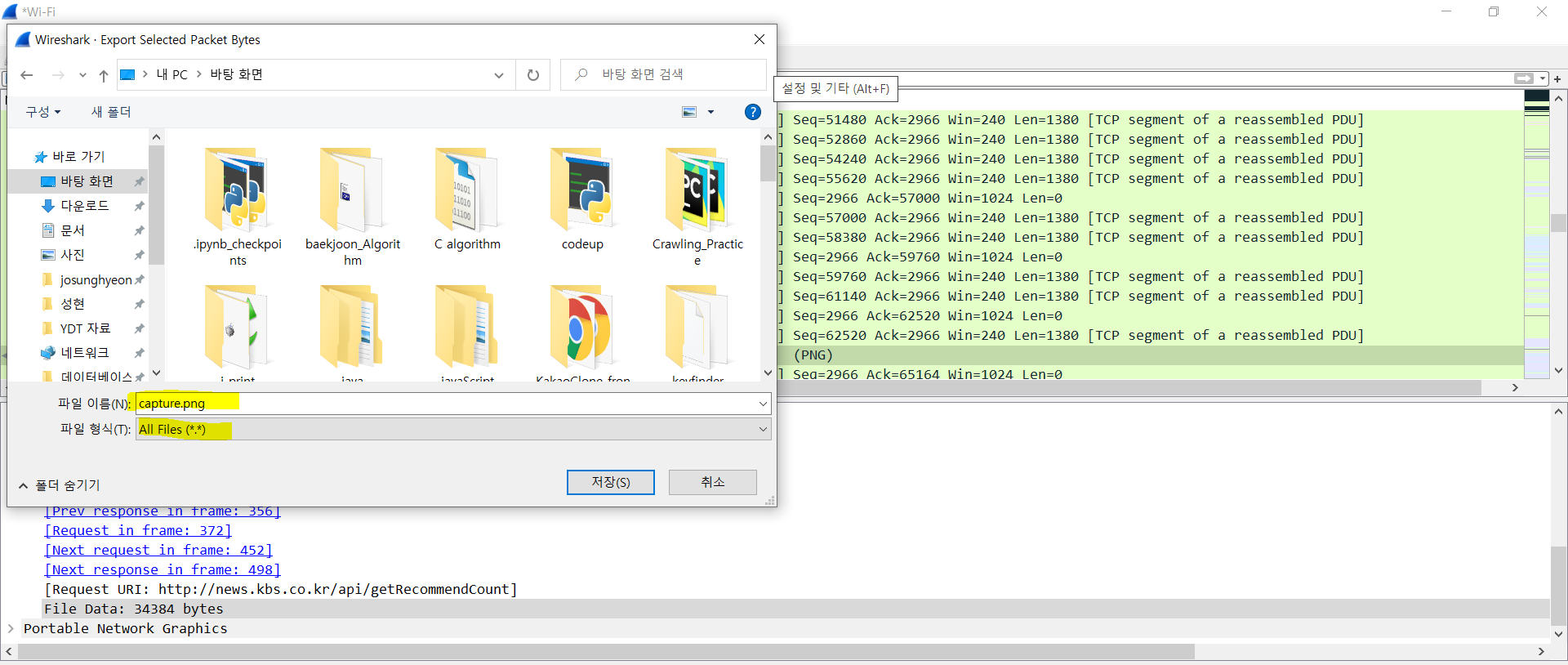
HTTP PNG Packet번호를 빠르게 찾기 위해 HTTP Object list를 참고하였다.



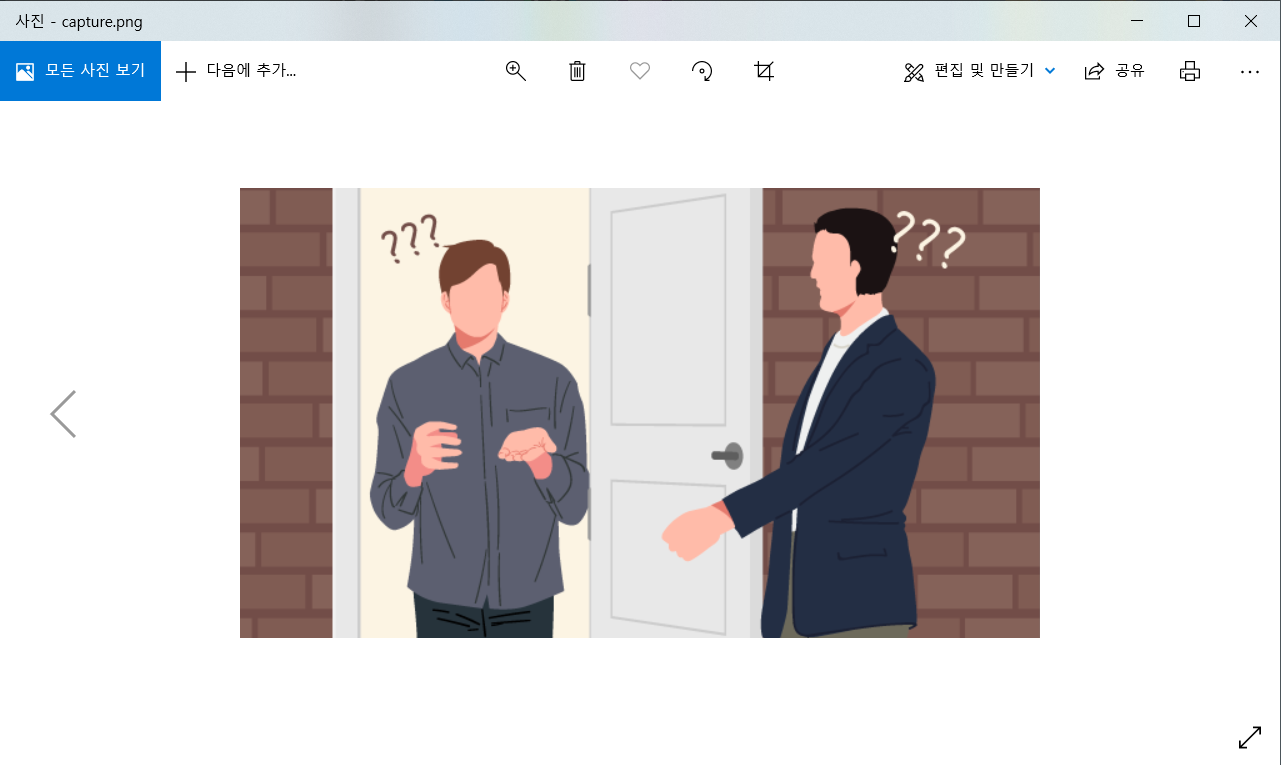
위에서 Packet No.412번, HTTP Protocol, Packet Length = 1338인 PNG 파일의 Packet을 Capture하여 Export하기로 한다. 또한 위의 그림에서 확인할 수 있듯이, File Data는 34384 bytes이다.

****

위 그림에서 File Data 부분에서 마우스를 우클릭하면 Export Packet Bytes가 있다. 이 부분을 클릭하면 아래와 같이 PNG 파일을 저장할 수 있다.

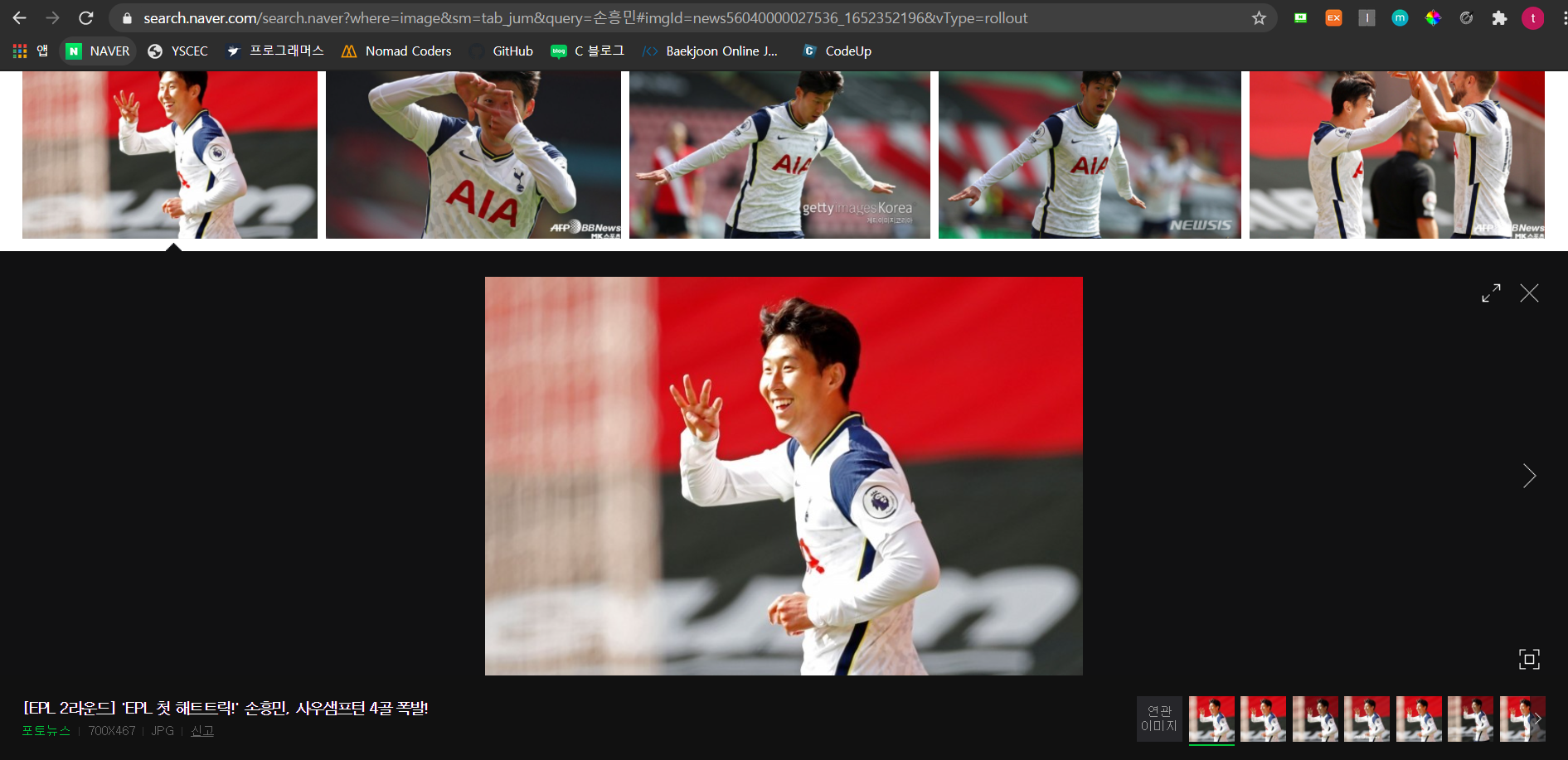


Capture.png라고 파일 이름을 입력하고, 파일 형식은 All files로 설정한다.



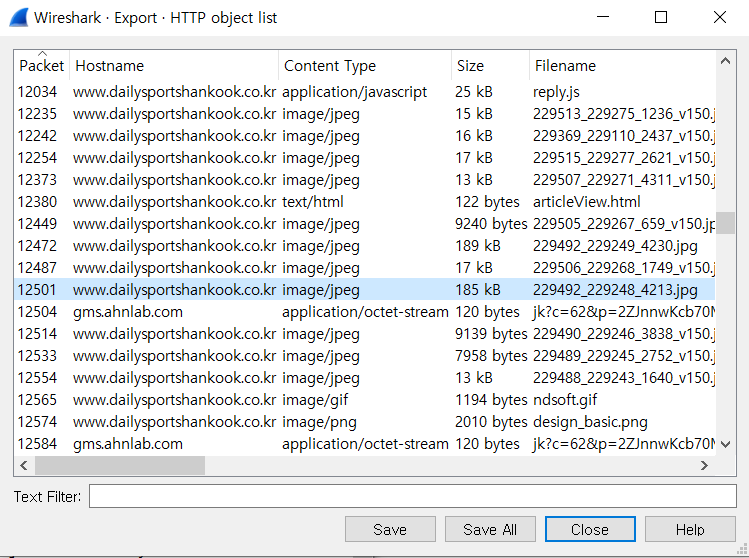
위와 같이 PNG 파일을 Packet을 Capture하여 Export하였다.

1-2 Wireshark를 이용한 Packet Capturing 방법2



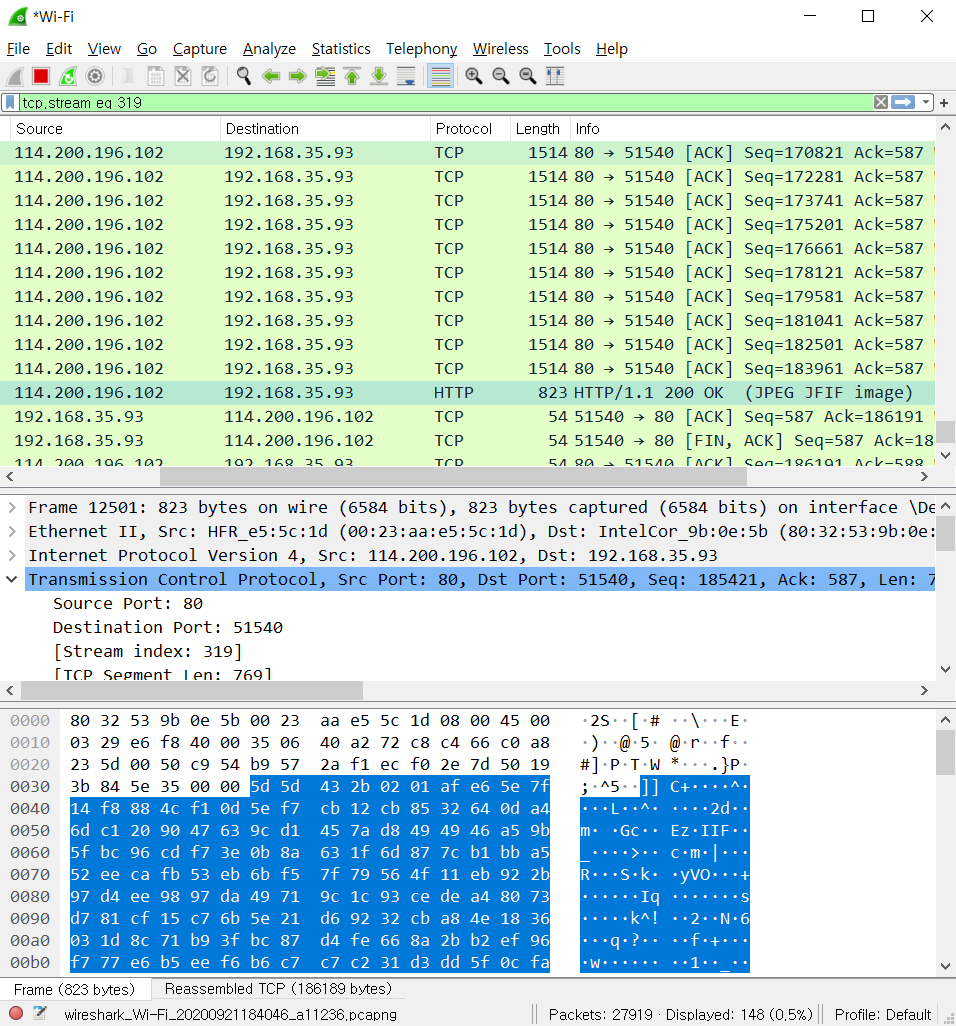
**http://search.naver.com/search.naver?where=image&sm=tab\_jum&query=%EC%86%90%ED%9D%A5%EB%AF%BC#imgId=news56040000027536\_1652352196&vType=rollout**

**Wireshark를 이용하여 위의 Http 주소에 있는 손흥민 선수의 이미지를 Export하려고 한다.**

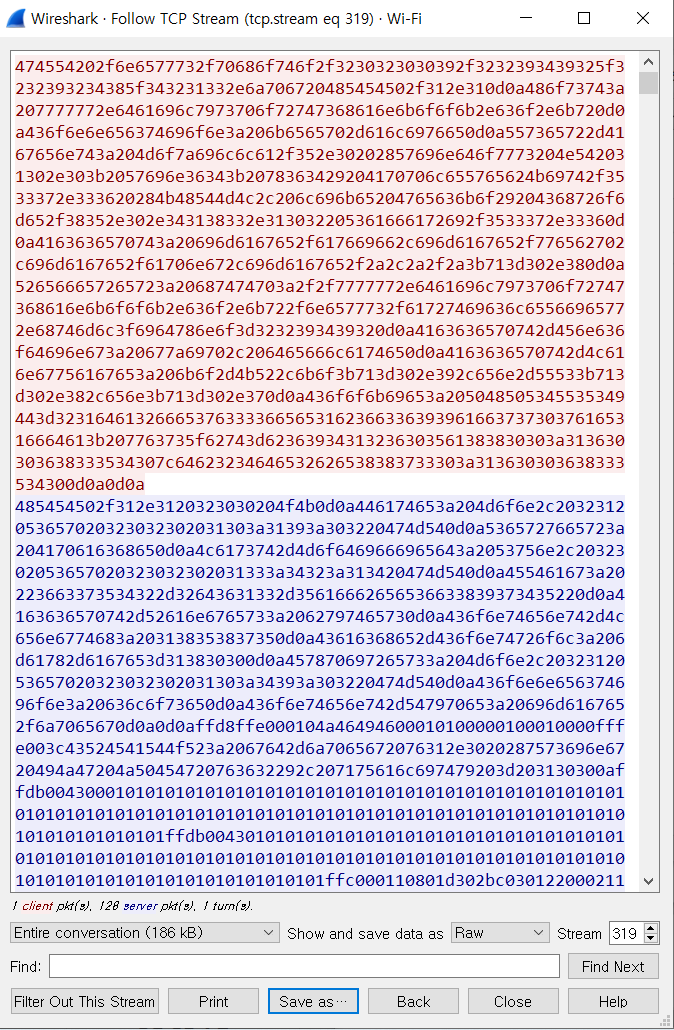
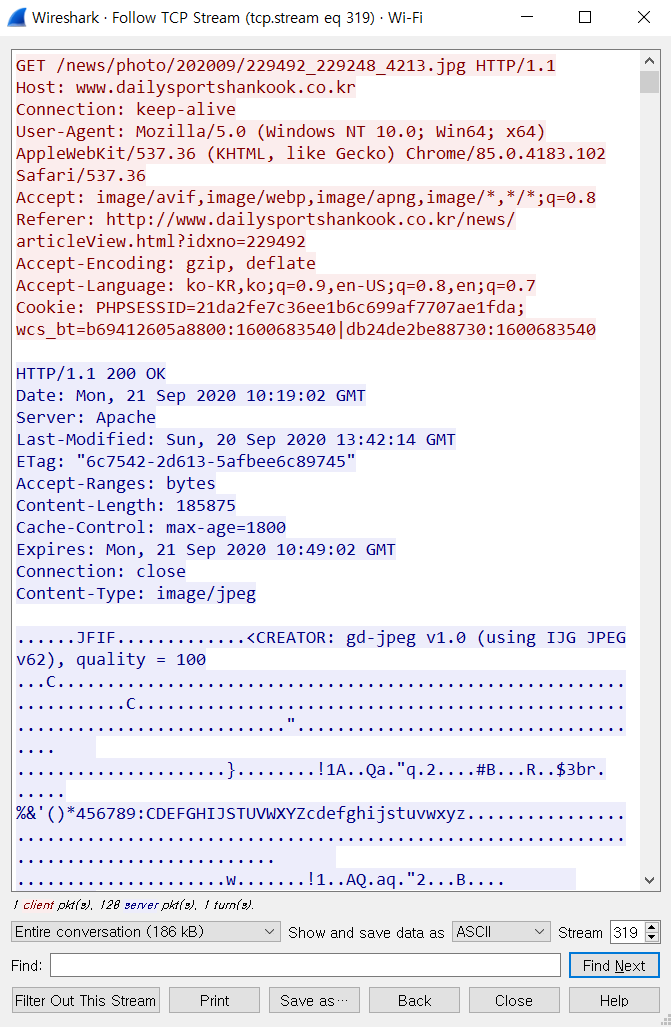


해당 이미지 패킷의 Packet Number를 확인하기 위해 Wireshark의 HTTP object를 확인하였다.

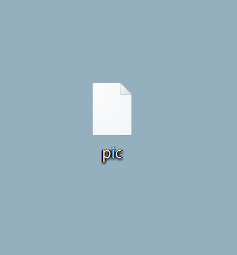
해당 이미지의 Packet number는 ‘12501’이다.



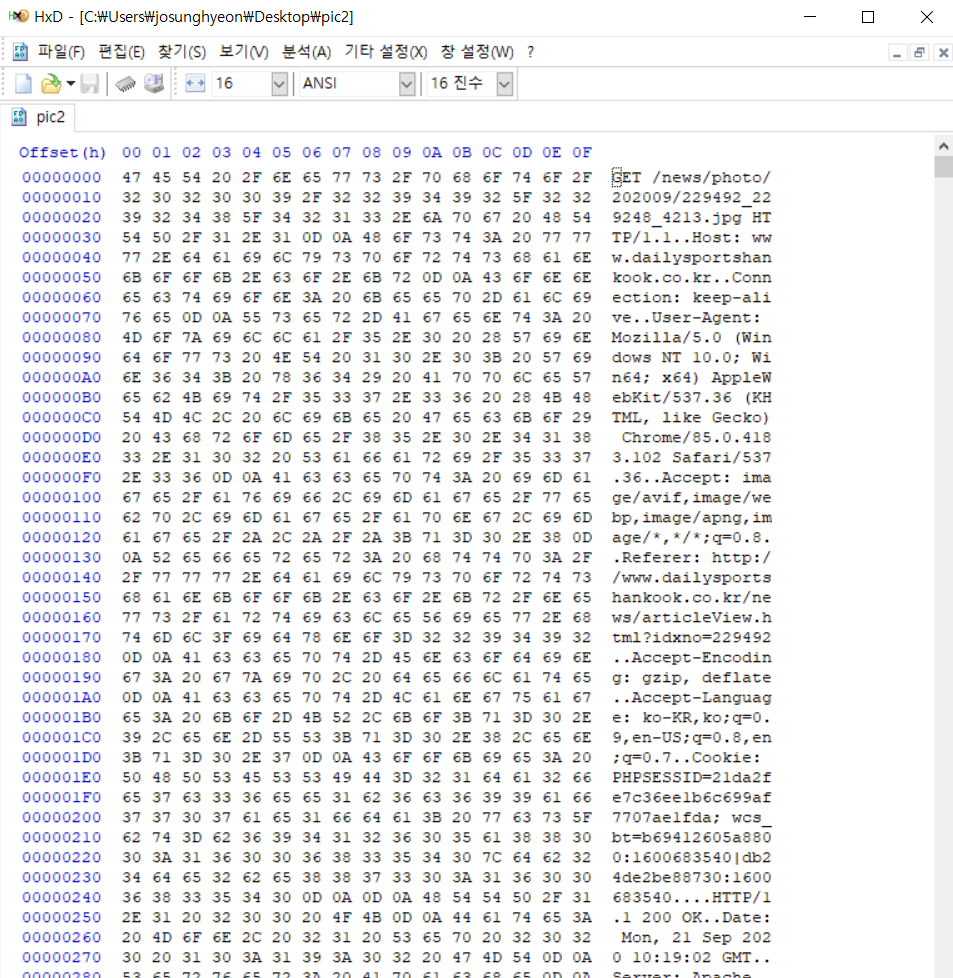
해당 이미지 패킷의 TCP Stream을 모아왔다.



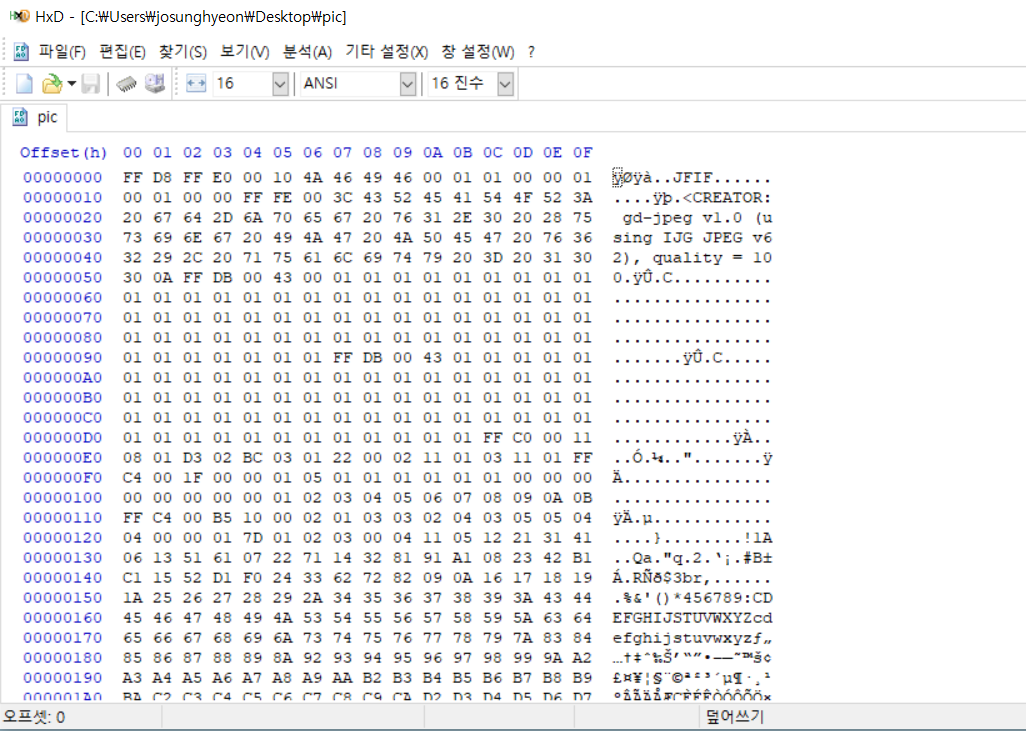
빨간색으로 색칠된 부분은 이미지를 서버에 요청한 내용이며 보라색 부분은 이미지를 보내서 응답한 부분을 나타낸다. 요청 부분과 응답부분에서 이미지 파일의 내용만 잘라서 추출하기 위해 Raw 형식으로 패킷 내용을 바꿔서 출력했다.



패킷의 내용을 Text 형식으로 저장하고, HxD 파일의 형태로 열어보았다.



Raw로 변경한 패킷 내용을 HxD 파일을 이용해 열어보았다. 하지만 이 파일 전체를 저장하면 이미지 파일의 형태로 저장할 수 없다. 그 까닭은 패킷 내용안의 파일 시그니쳐를 확인해서 그 앞의 요청 패킷 내용은 지워야 하기 때문이다. 다시 말해서, 이 패킷이 담고 있는 파일이 이미지 형식이라는 것을 Window에게 알려 이를 인식하도록 해야 하는 것이다.

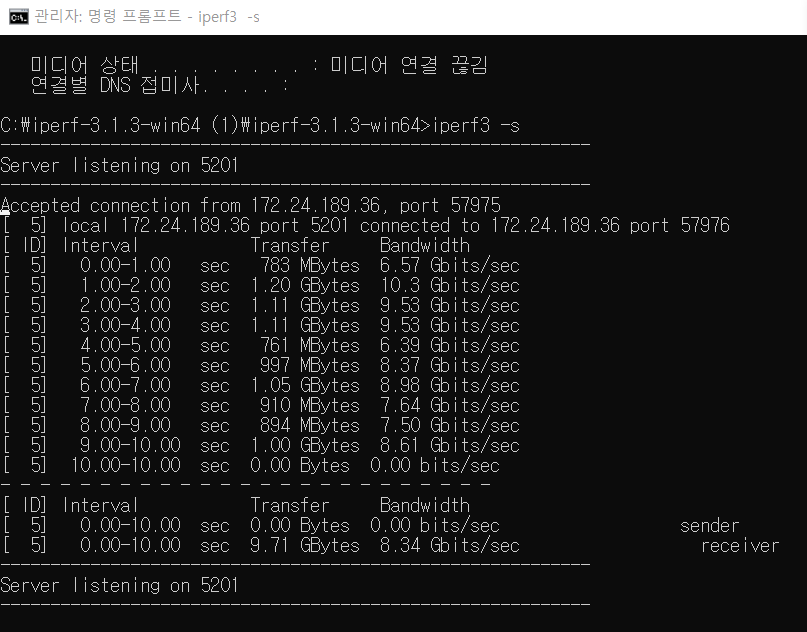


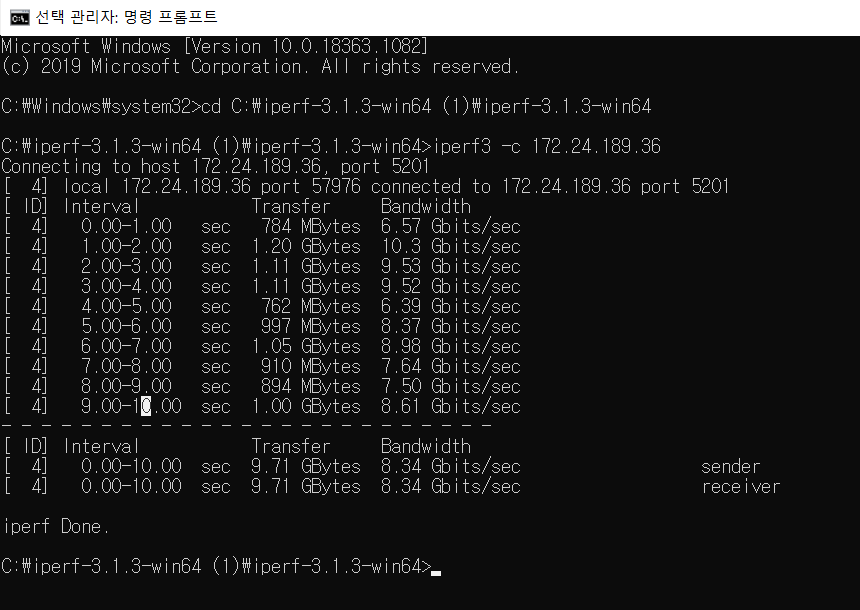
파일 시그니처(FF D8 FF E0) 부분 앞의 패킷 내용을 지우고 남은 부분을 다시 이미지 형태로 저장하면 패킷을 통한 이미지 추출이 완료된다. 완성된 그림은 아래와 같다.



**2. iPerf3 Report**

Iperf3를 컴퓨터 내에서 사용하기 위해 두 개의 명령프롬프트를 사용하였다. 첫번째 이미지는 서버, 두번째 이미지는 클라이언트의 Iperf3 실행 결과이다. OS체제는 Window10 환경에서 수행했다.





\* 우선 Server의 옵션을 실행시켜주고, 반대 쪽 Client PC에서 Packet을 전송한다.

우선, Window 환경에서 Bitrate와 Cwnd Column값이 나오지 않는다.

따라서, Bitrate를 대신하여 Bandwidth를 사용하고, Cwnd 값은 그 정의와 실행에서의 기본값에 대해 설명하고자 한다.

1) Transfer

Transfer 값은 네트워크 환경뿐만 아니라, 목적지인 Server에 접근할 수 있는 속도에 제한받는다.따라서, 큰 데이터를 보낼 때 작은 set로 나누어서 보내는 것이다.

현재 결과창을 살펴보면 10초동안 Transfer값은 9.71GBytes이다.

2) Bitrate

Window10 환경에서 Bitrate는 Bandwidth와 거의 비슷하다.

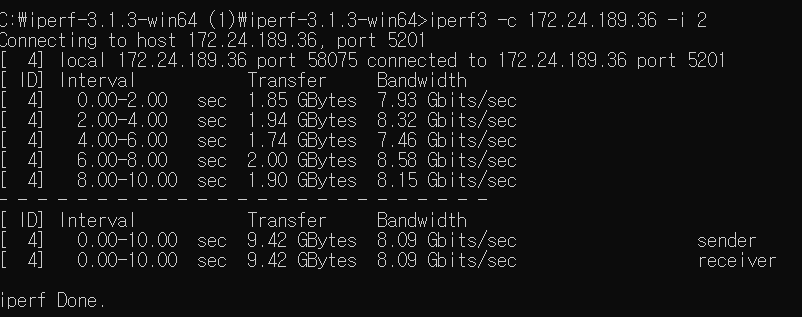
Bandwidth는 두 서버 간 대역폭을 의미한다. 위에서 8.34Gbit/sec라는 결과를 보면 7기가비트의 포트의 속도가 보장되고 있다고 볼 수 있다. 따라서, 서버와 클라이언트 간 통신 속도는 8.34Gbit/sec임을 알 수 있다.

3) CWND

TCP의 흐름 제어를 할 때 송신 측의 윈도우 크기는 수신 측이 보내준 윈도우 크기와 네트워크 상황을 고려해서 정해진다. 송신 측은 자신의 최종 윈도우 크기를 보내줄 때, 수신 측이 보내준 윈도우 크기인 RWND, 그리고 자신이 네트워크의 상황을 고려해서 정한 윈도우 크기인 CWND 중에서 더 작은 값을 사용한다. 즉, CWND는 서버와 클라이언트 간 데이터를 전송할 때, 송신 측의 네트환경을 고려한 윈도우 크기이다.

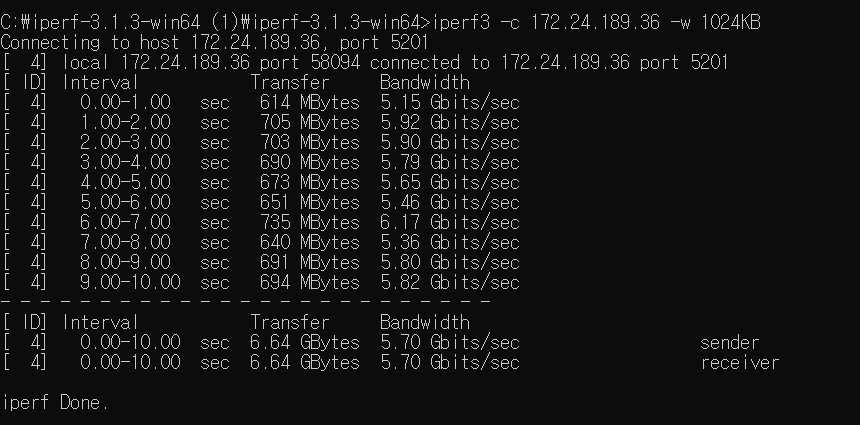
4) -i 옵션

Iperf의 결과값은 일정 시간동안 측정한 평균값을 보여준다. -i 옵션을 이용하여 초 단위를 제어할 수 있다. 예를 들어 i값을 2로 설정하면, 2초 단위로 측정값이 표시되고, 그 결과 1초보다 2초동안 측정한 Transfer값은 커지게 된다. 하지만 전체 시간동안 Transfer값은 크게 변하지 않는다.



5) -w 옵션

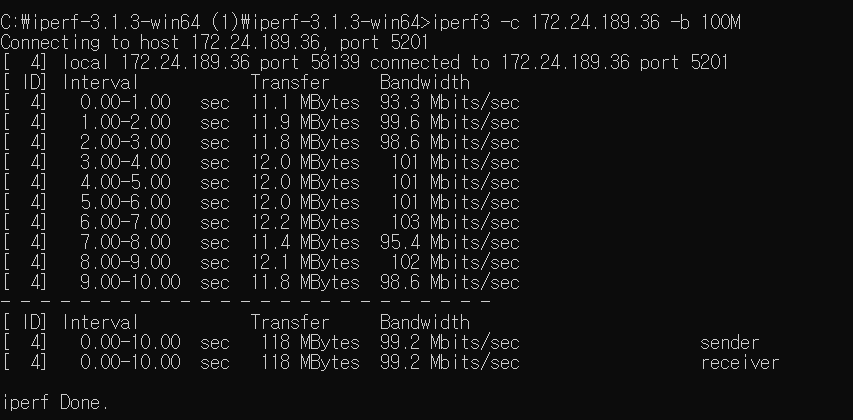
-w 옵션은 TCP 윈도우 사이즈를 조정하는 옵션으로, 원하는 크기의 덩어리 단위로 데이터를 보내 측정할 수 있다. 예를 들어 -w 1024KB라고 입력하면, TCP 윈도우 사이즈가 1Mbyte로 바뀌어 측정된다.



6) -b 옵션

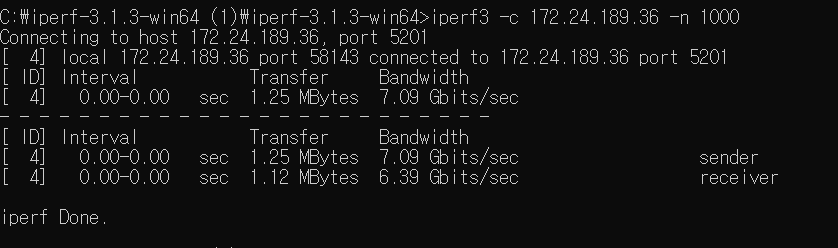
-b 옵션은 UDP 프로토콜일 때 대역폭을 지정한다.

예를 들어 -b100M을 옵션으로 지정하게 되면, 서버에 접속해서 100Mbps의 대역폭으로 무선 구간의 성능을 측정하게 된다.



7) -n 옵션

Iperf는 기본값으로 10초의 시간동안 서버의 성능을 측정한다. 하지만 -n옵션을 주면 전송할 bytes의 숫자를 time 대신에 입력받는다. 예를 들어 -n 1000bytes라고 하면 1000bytes를 전송하는 동안 서버의 성능을 측정하게 되어 1초도 되지 않는 시간에 전송이 완료되는 것을 확인할 수 있다.



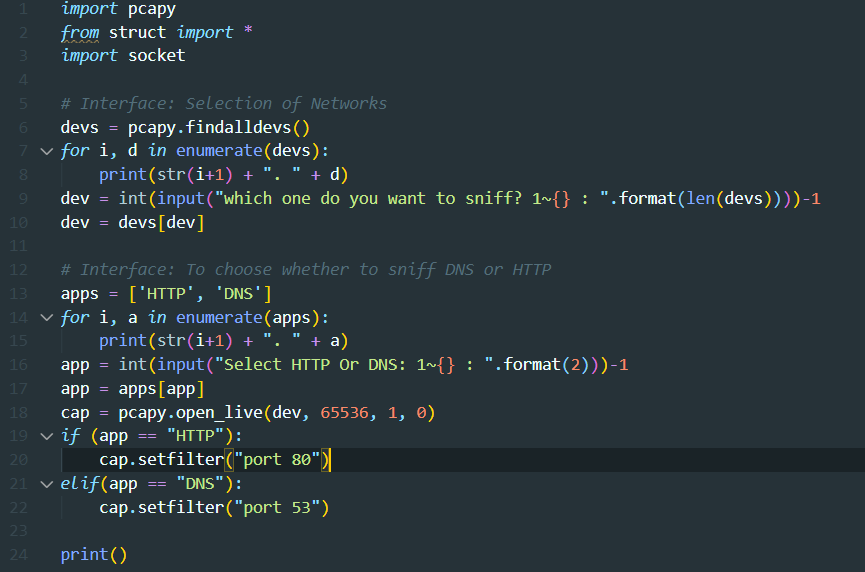
[아래에 1-3 계속]

1-3 Simple HTTP & DNS Sniffer

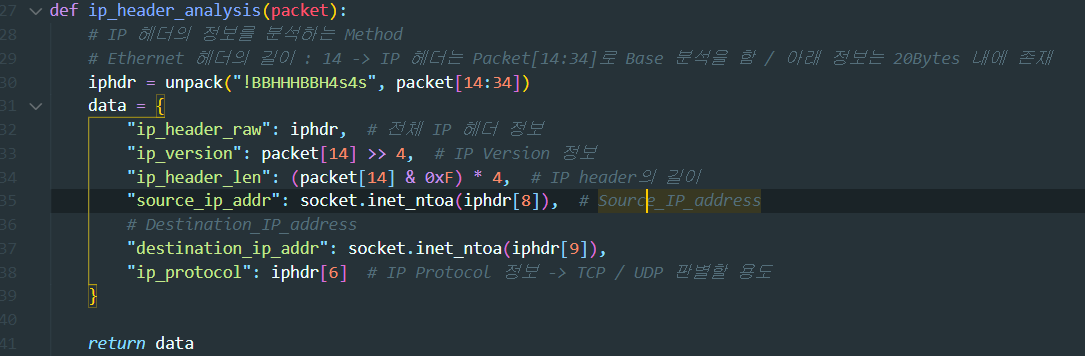
1) introduction

Python3 환경에서 Ubuntu18.04를 이용하여 프로젝트를 진행하였다. Pcapy라이브러리를 사용하여 패킷을 분석하였고, 그 과정에서 Struct, Socket 표준 라이브러리를 이용하여 패킷을 Parsing하였다.

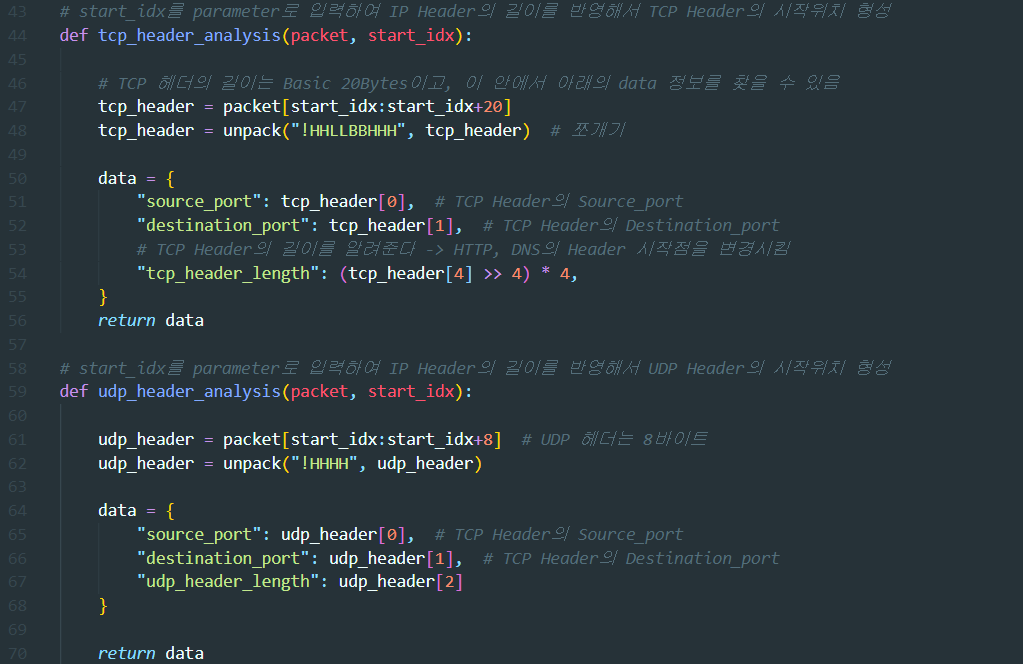
2) Code / Comments / Flow chart



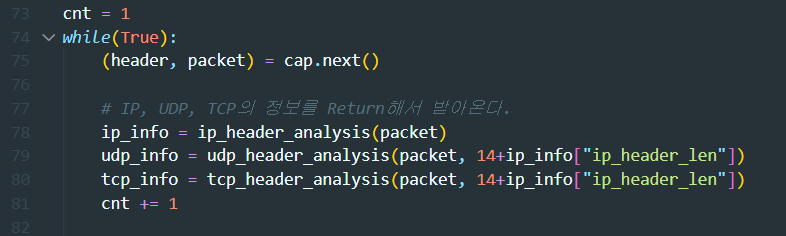
기본 Interface를 구축하기 위한 코드이다. dev(=device)와 app(=Application)을 각각 순서대로 선택할 수 있도록 하였다. HTTP의 경우에는 Port 번호를 80으로 설정했으며, DNS의 경우 Port 번호를 53으로 설정하는 조건문을 이용해 Filtering을 진행하였다.



Ethernet 헤더의 길이는 14로 고정되어 있기 때문에, IP Header에 대한 분석을 하였다. 기본적으로 20Bytes 크기의 IP Header를 Parsing하여 정보를 읽었다. 정보에 대한 설명은 Code comment에 제시하였다.

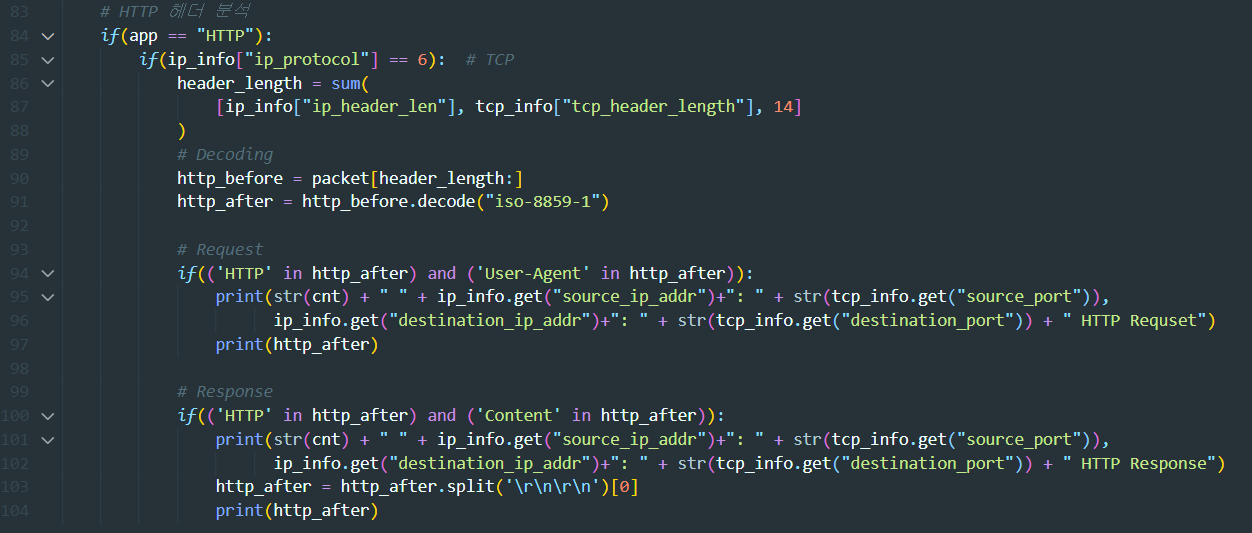


IP header의 길이에 따라 TCP, UDP 헤더의 패킷 내 시작위치가 달라진다. 따라서 ‘start\_idx’라는 매개변수를 통해 시작위치를 설정할 수 있도록 하였다. TCP 헤더의 경우 기본 20Bytes의 헤더 길이를 기본값으로 가지고 있다. 따라서 그 범위 내에서 헤더를 분석하였다. TCP 헤더의 구조에 따라 Bytes 단위로 패킷을 Parsing하였고, 해당 인덱스에 따라 Source port 번호, Destination port번호, TCP 헤더의 길이를 구하였다. TCP 헤더의 길이에 따라 HTTP, UDP 헤더의 시작위치가 영향을 받기 때문에 이를 뒷부분에서 다시 사용하도록 하였다. UDP 헤더의 분석 역시 TCP와 비슷한 방식으로 진행하였고 TCP, UDP헤더에서 필요한 정보는 위의 Code Comment를 통해 제시하였다.



본격적으로 HTTP, DNS Header를 Sniffing 하기 위한 코드이다. NO = cnt = 1로 시작하였다.

IP, UDP, TCP의 정보는 위에서 설명한 함수를 통해 각각 Data를 Return 받도록 하였다. 특히 TCP, UDP 헤더의 경우 IP header의 길이에 영향을 받기 때문에 이를 고려하여 매개변수를 입력했다. 14는 기본적인 Ethernet Header의 길이다.

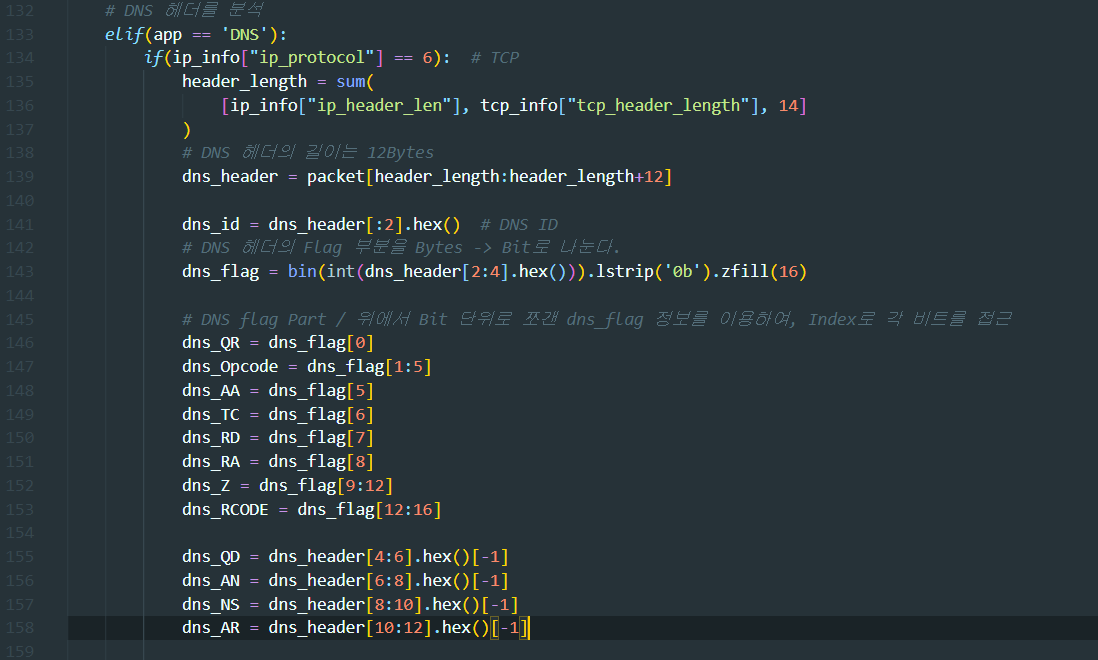


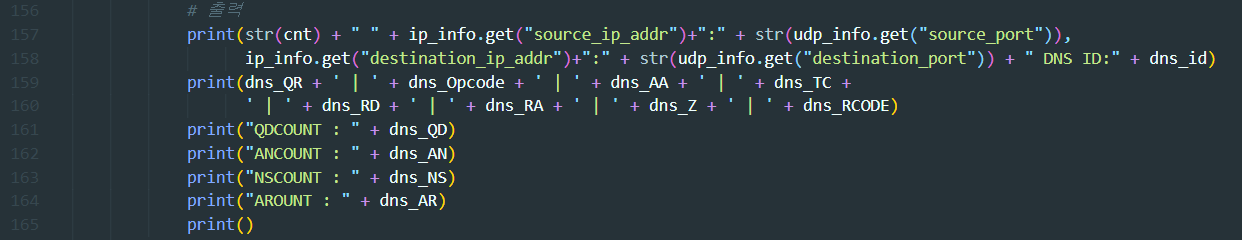


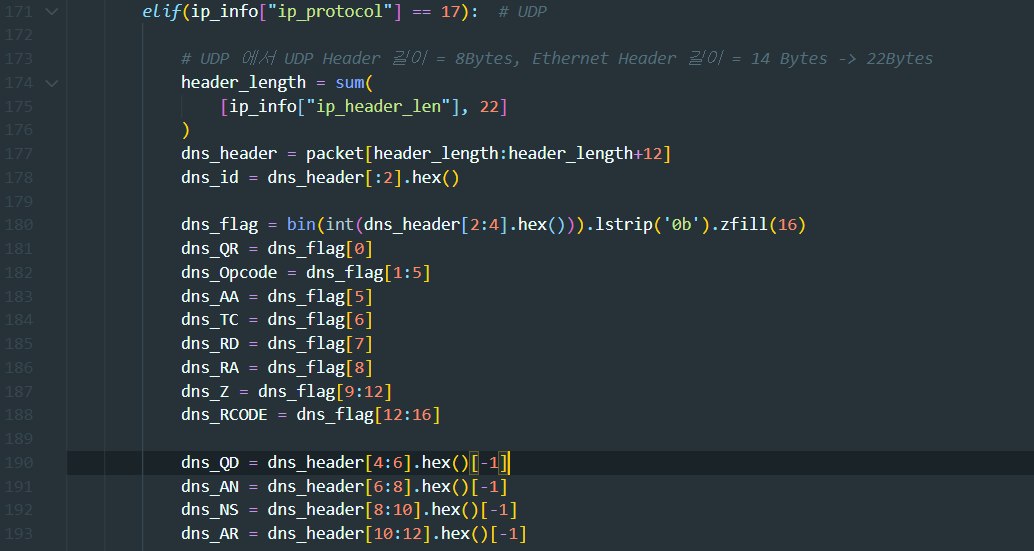
HTTP Header Sniffing 과정이다. IP Protocol의 값에 따라 TCP, UDP 방식을 달리 설정하였다.

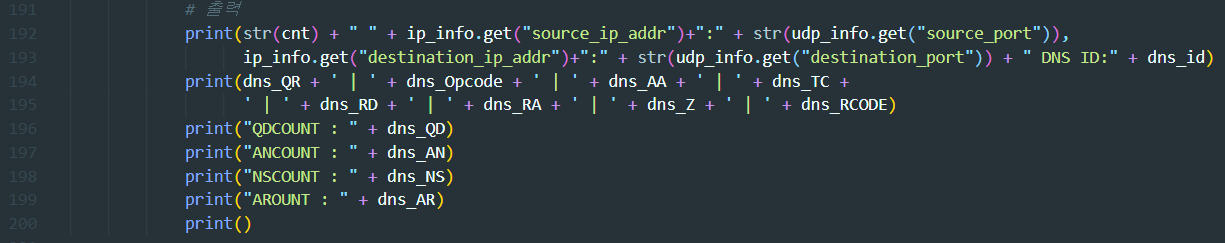
TCP의 경우 HTTP header의 시작 위치는 14(Ethernet) + IP Header Length + TCP Header Length이며, UDP의 경우 HTTP header 시작 위치는 14(Ethernet) + 8(UDP header length) + IP Header Length이다.

“ISO-8859-1”을 이용하여 Packet을 Decoding하였고, Request는 ‘HTTP’, ‘Host’ 키워드를 이용하여 추출했다. 또한 Response는 ‘HTTP’, ‘Content’ 키워드를 이용하였다. Header와 Body의 구분은 ‘\r\n\r\n’을 기준으로 하여 Split()함수를 사용했다.









DNS Header를 분석하는 것 역시 IP Protocol을 통해 TCP, UDP 방식을 분류하였고 Header의 시작위치를 각기 다르게 설정해주었다. 한편 DNS의 Header 길이는 12로 고정하였고 DNS Header는 ID part, Flag Part, Count Part 4가지를 각각 2Bytes 단위로 잘라서 분석하였다. 특히 Flag Part(QR, OPCODE, AA, TC, RD, RA, Z, RCODE)는 비트단위로 분석하여 Index를 이용해서 분석했다.

3) Flow Chart

<HTTP Header Sniffing>

Dev(eth0), App(Http) 선택 🡪 Packet 추출 🡪 IP, TCP, UPD 헤더의 정보를 각각 Parsing 🡪 HTTP header의 시작 위치 설정(IP Protocol 정보 이용) 🡪 HTTP Header의 Request, Response를 Keyword를 이용하여 분류 🡪 Header와 Body를 분리하기 위해 ‘\r\n\r\n’ 키워드를 사용 🡪 Format에 맞추어 출력

<DNS Header Sniffing>

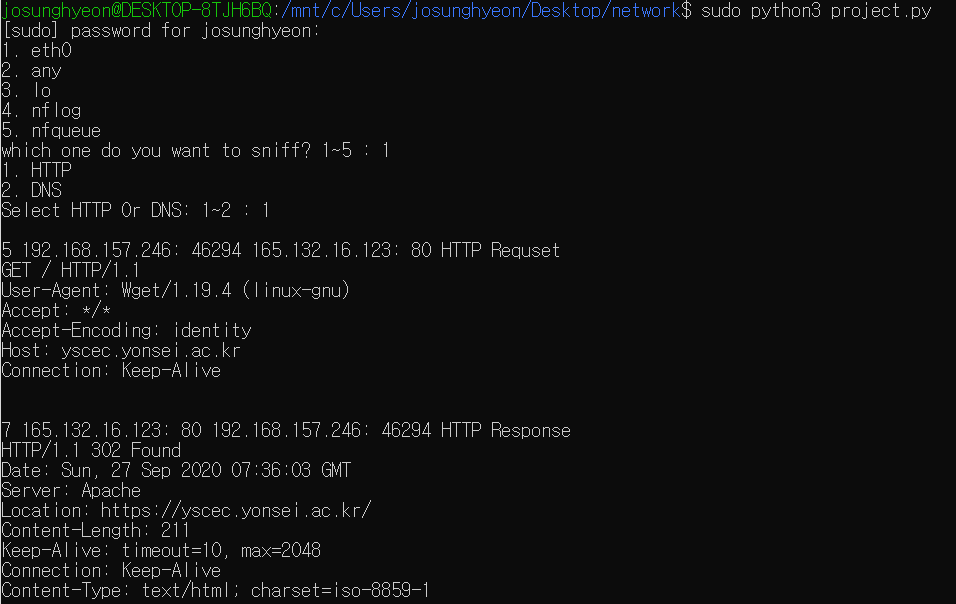
Dev(eth0), App(DNS) 선택 🡪 Packet 추출 🡪 IP, TCP, UPD 헤더의 정보를 각각 Parsing 🡪 DNS header의 시작 위치 설정(IP Protocol 정보 이용) 🡪 DNS Header 길이 12Bytes를 2Bytes 단위로 Parsing 🡪 DNS ID, DNS Flag, DNS Count 부분으로 나누어 Parsing 🡪 비트 단위 분석 🡪 Format에 맞추어 출력

[아래에 Screenshot 계속]

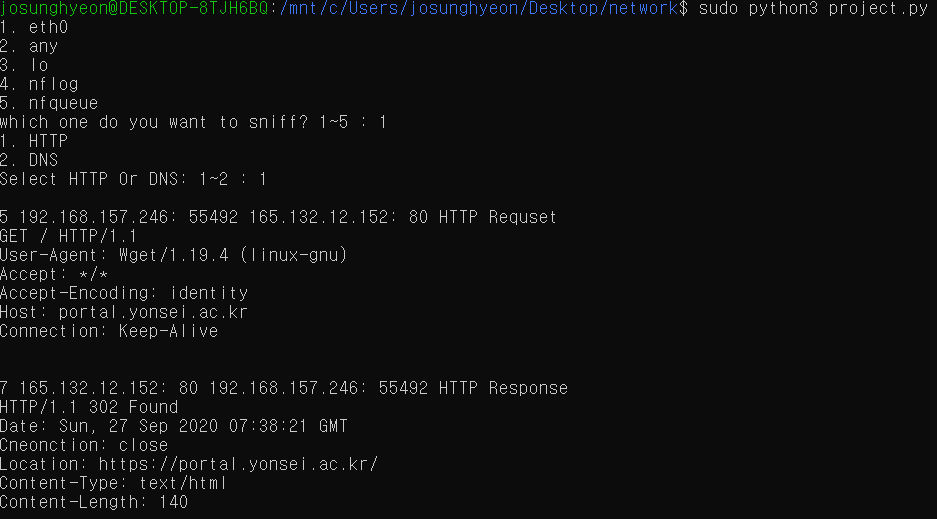
4) Screenshots of Result



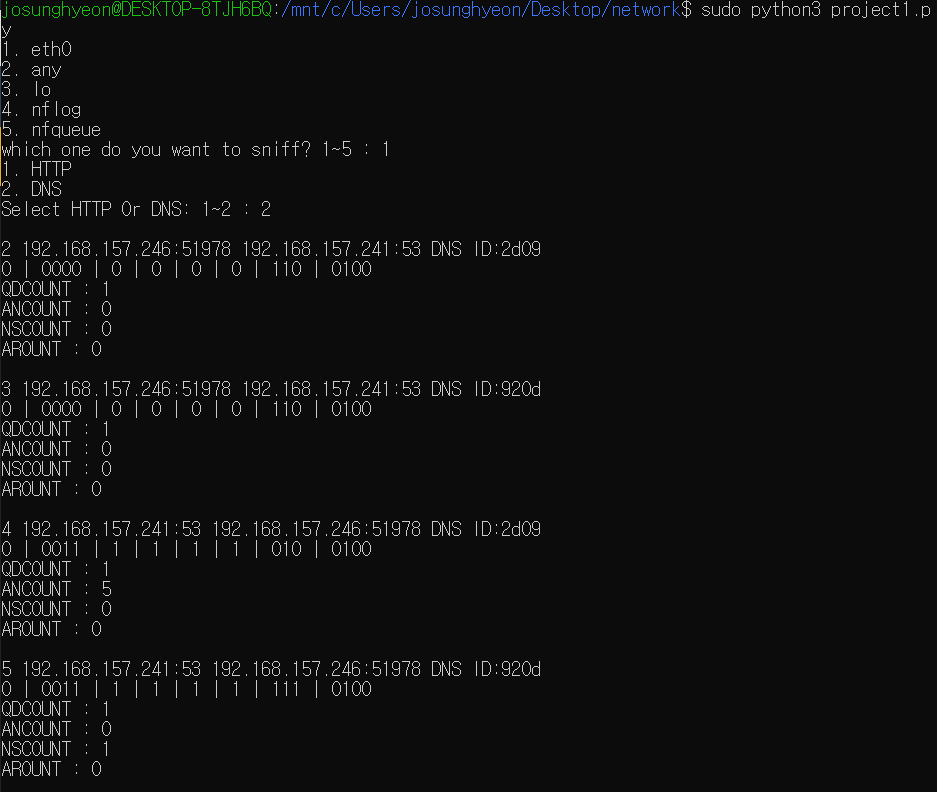
: <http://yonsei.ac.kr> HTTP Header Sniffing



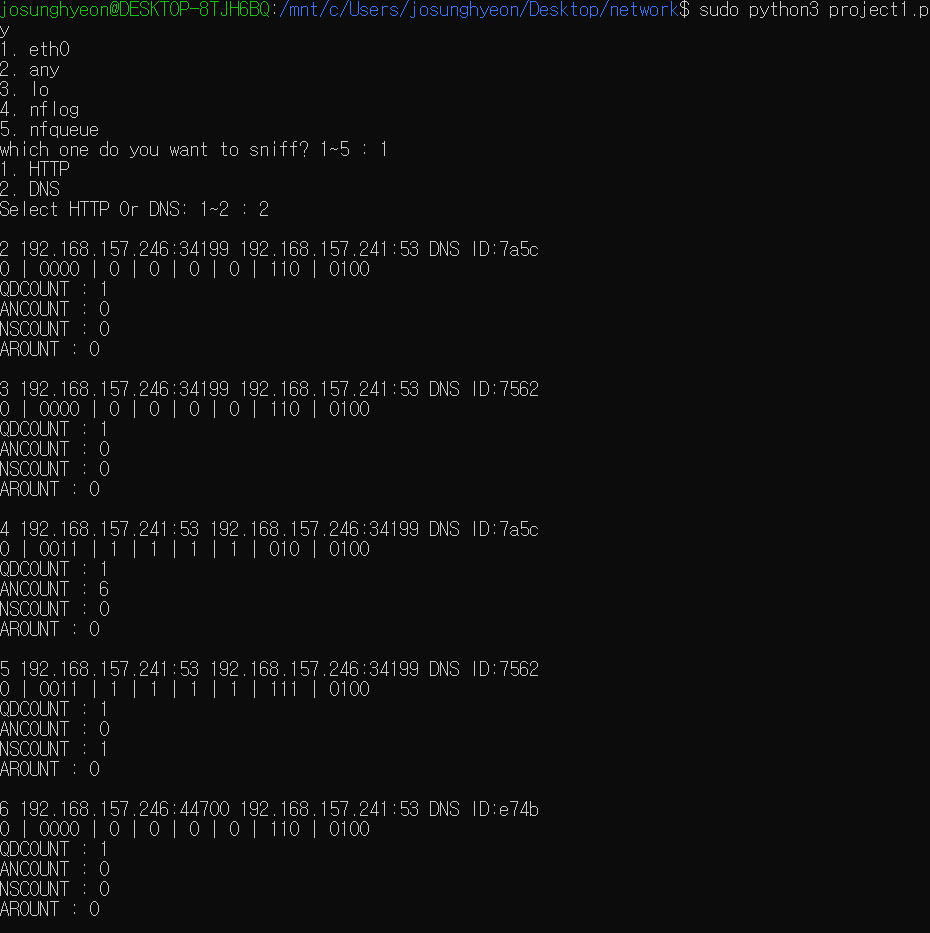
: <http://yscec.yonsei.ac.kr> HTTP Header Sniffing



: <http://portal.yonsei.ac.kr> HTTP Header Sniffing



: <http://yscec.yonsei.ac.kr> DNS Header Sniffing



: <http://naver.com> DNS Header Sniffing