컴퓨터 네트워크 과제1 – Wireshark를 이용한 TCP Traffic분석

학번 : 201520908

학과 : 소프트웨어

성명 : 유성민

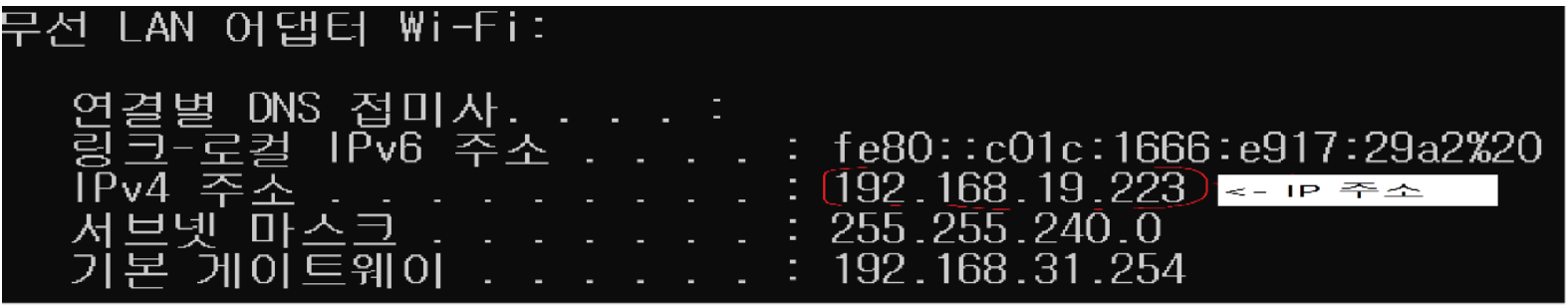
1) 관찰 방법

1차 패킷 캡쳐는 2019년 5월 5일 일요일 아주대학교 중앙도서관 4층 열람실에서 개인노트북을 이용하여 Ajou Univ WIFI 네트워크에 접속하였고 2차 패킷 캡쳐는 2019년 5월 6일 월요일 아주대학교 중앙도서관 2층 커뮤니티 라운지에서 사람이 많은 시간대에 가서 1차와 동일한 방법으로 동일한 네트워크에 접속하였고 관찰을 위하여 사용한 Network application은 internet explorer browse이다. TCP분석을 위해 평소에 즐겨보는 NAVER 포털사이트에서 제공하는 축구 영상 시청 서비스를 선택하였다. 선택 이유는 영상 시청이 가장 Packet교환이 활발 할 것 같다고 생각하였기 때문이다. 충분한 샘플을 얻기 위해 영상은 5분정도 시청하기로 정하였다. 주된 관찰 타깃은 "TCP connection setup과정에서 3-way handshaking이 발생하는가?", "setup후 data교환이 어떻게 이루어지는가?", 상호작용 후 connection close과정 그리고 특별한 경우인 Keep-alive, connection-reset, Retransmission과정이다. 본인이 해보고 싶은 사항으로는 혼잡제어 관해 더 자세한 이해를 위하여 사람이 많은 시간대와 적은 시간대에서 네트워크에 접속하여 두 가지 경우의 차이를 확인해 보았다. 이를 위하여 사용할 기능은 과제 시작 전 미리 Wireshark 프로그램을 실행하여 파악하였는데, 1. IO graph, 2. Endpoints, 3. Protocol Hierarchy, 4. Expert info, 5. TCP stream graphs기능 이상 5가지 기능을 주로 사용하였다.

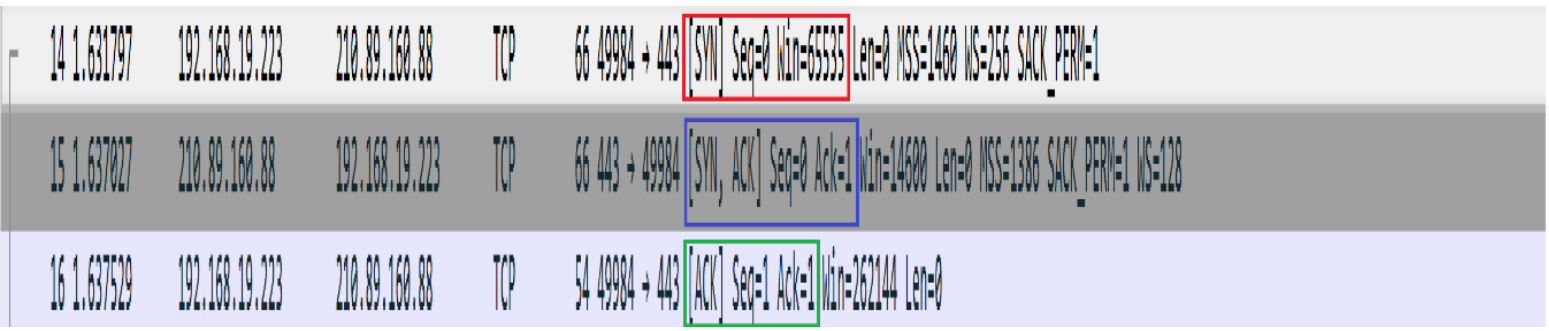
2) Traffic 분석

시작하기 전에 source와 destination의 IP파악을 위해서 본인 PC의 IP를 확인해 보았는데, CMD 프로그램을 실행하여 ipconfig명령을 사용하였다. 나의 접속 IP는 192.168.19.223으로 확인되었다. 보다 정확한 분석을 위해서

Wireshark를 실행시킨 상태에서 internet browser를 실행시키고 TCP의 Traffic관찰을 위해 필터로 TCP-ONLY를 선택하고 본격적인 Traffic 분석을 시작해보았다.



<그림1. IP Config명령으로 본인의 IP 파악하기>



<그림2. 브라우저를 실행시킨 직후 Wireshark에 처음으로 잡힌 패킷>

```
Flags: 0x002 (SYN)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...0 .... = Acknowledgment: Not set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
  > .... .... ..1. = Syn: Set
    .... .... ...0 = Fin: Not set
    [TCP Flags: ··········S·]
Window size value: 65535
```

```
Flags: 0x012 (SYN, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
  > .... .... ..1. = Syn: Set
    .... .... ...0 = Fin: Not set
    [TCP Flags: ·····A··S·]
Window size value: 14600
```

<그림3. 왼쪽은 본인pc가 서버에게 보낸 패킷, 오른쪽은 서버로부터 본인pc로 온 패킷>

95	1.792588	192.168.19.223	183.111.26.110	TCP	66 49986 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
96	1.792589	192.168.19.223	183.111.26.110	TCP	66 49985 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
98	1.794820	192.168.19.223	183.111.26.110	TCP	66 49987 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
99	1.795479	183.111.26.110	192.168.19.223	TCP	66 443 → 49985 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1386 SACK_PERM=1 WS=128
100	1.795480	183.111.26.110	192.168.19.223	TCP	66 443 → 49986 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1386 SACK_PERM=1 WS=128
101	1.795641	192.168.19.223	183.111.26.110	TCP	54 49985 → 443 [ACK] Seq=1 Ack=1 Win=262144 Len=0
102	1.795718	192.168.19.223	183.111.26.110	TCP	54 49986 → 443 [ACK] Seq=1 Ack=1 Win=262144 Len=0

<그림4. Syn FLAG 패킷 => 하나의 Application은 여러 개의 TCP연결을 설정 할 수 있다.>

첫 번째로 connection setup이다. 이렇게 판단한 이유는 수업시간에 배운 내용을 토대로 과정을 천천히 상기시켜 보면, 첫 번째로 본인 컴퓨터(Sender)가 TCP연결을 초기화한다. 본인이 보낸 패킷의 SYN FLAG가 1이었고 첫 순서번호로 0을 선택하였다. 두번째로는 나와 연결된 서버(receiver)는 SYN FIAG를 포함한 ACK으로 응답을 해야 하고 ACK=1이므로 1번 받기를 기대함을 의미한다. ACK FLAG가 1인 것을 볼 수 있는데 이는 ACK의 번호가 의미 있는 값임을 알려준다. 마지막으로 ACK대한 응답으로 본인 PC는 Seq=1을 서버에 보낸다. 이로써 본인 PC가 GHOST USER가 아님을 밝히고 이 과정은 본인 PC와 서버 210.89.160.88의 3-Way handshaking이었다. 초기 연결 외에도 많은 SYN Flag를 포함하는 segment가 있었다. 이를 통해 알 수 있던 점은 첫째로, TCP가 Persist-connection이기 때문에 연결은 브라우저 하나와 하면 되는 줄 알았는데, 착각이었다. 실제로는 연결 설정과 연결 종료가 수 번 있었다. 여러 개의 SYN Flag를 포함한 segment를 비교 대조한 결과 차이점은 port number였는데, 포트번호에 대한 개념이 부족 했음을 깨닫게 되었다. 1개의 port number가 부여되는 줄 알았으나(예를 들어 80 - HTTP), 프로그램 하나에도 기능 마다 필요한 데이터가 다를 수 있다. 그런 이유로 여러 개의 포트번호가 부여되는 것으로 생각 해보았다. 그렇게 생각하는 이유는 단순한 영상재생일 지라도 기능은 다양하기 때문이다. 이를테면, 화질 선택, 멈춤, 재생속도 변경, 건너 뛰기 등등 기능이 많기 때문이다. 두 번째로 알 수 있었던 것이 있었는데 Seq num이 모두 0으로 시작한다는 것이다. 이 것은 수업시간에 배운 대로라면 OS에서 지정하는 Random number인데(겹침의 문제로 인한 피해를 방지하기 위함), 첫 연결에 순서번호를 0으로 시작해서 보다 쉽게 사용자가 패킷 분석을 할 수 있도록 도와준다. 때문에 분석을 하면서 많이 도움되었다. 이 기능을 해지하려면 Preference->relative sequence number의 체크를 해제하면 된다. 실제 순서번호를 보니 매우 복잡하여 이 기능이 없었다면 어디가 시작인지 알기 어려웠을 것 같다. 그러나 정확한 분석을 위해서 기능을 해제하는 것이 좋다.

두 번째로 연결의 종료 과정이다.

종료를 인지하기 위해서는 FIN FLAG를 포함하는 segment를 찾으면 되는데, 이 것은 정상적인 상호작용을 하였다면 SYN Flag를 포함하는 segment와 짝을 이루는 줄 알았으나 export info 기능을 사용하니 fin flag를 포함하는 패킷이 더 많았다. 이를 토대로 추측해 볼 때 비정상적인 종료 때문인 것으로 생각된다.

```
606 119.329215   210.89.160.88    192.168.19.223   TCP   60 80 → 53425 [FIN, ACK] Seq=395 Ack=1109 Win=16896 Len=0
607 119.329355   192.168.19.223   210.89.160.88    TCP   54 53425 → 80 [ACK] Seq=1109 Ack=396 Win=261632 Len=0
608 119.337635   192.168.19.223   210.89.160.88    TCP   54 53425 → 80 [FIN, ACK] Seq=1109 Ack=396 Win=261632 Len=0
609 119.340365   210.89.160.88    192.168.19.223   TCP   60 80 → 53425 [ACK] Seq=396 Ack=1110 Win=16896 Len=0
```

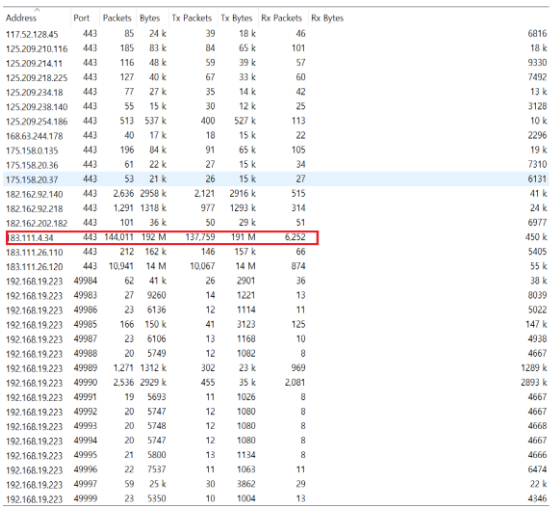
<그림5. 4-way handshaking FIN Flag segments>

```
Flags: 0x011 (FIN, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
    .... .... ..0. = Syn: Not set
    .... .... ...1 = Fin: Set
```

<그림 6. Fin flag가 포함된 패킷 헤더 일부>

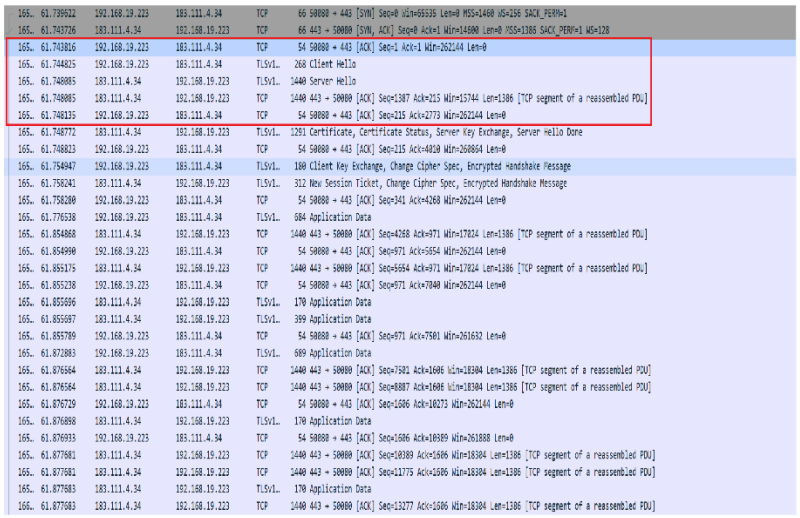
본격적으로 Fin flag를 분석해본다. 수업시간에 배운 내용을 토대로 위 패킷을 분석해보면, 처음에는 Server나 Client 둘 중 아무나 연결종료요청을 보낼 수 있다. 위 패킷의 경우 IP를 근거로 판단해 볼 때, 첫 번째로 서버에서 먼저 close 요청이 들어온다. 두 번째로 본인PC는 연결 종료 요청에 대한 ACK을 보낸다. 세 번째로 본인PC는 남은 일을 마치고 연결 종료 요청을 한다. 이 때, 이에 대한 ACK이 올 때까지 segment lifetime * 2만큼의 시간을 대기하여야 한다. 왜냐하면, 이를 하지 않는 경우 Server가 wait상태에 빠져 서비스를 제공하지 못하는 상태가 되어버린다. 이를 방지하기 위해 Client는 Server가 연결종료가 이루어 지게끔 책임질 필요가 있다. 위 과정을 끝으로 서버 210.89.160.88과의 연결을 마무리 짓는다. FIN Flag를 포함하는 segment 선택 과정에서 Wireshark의 endpoints 기능을 사용하였는데 이는, 무수한 패킷 중 source와 destination 선택이 가능하기 때문에 특정한 한 연결동안 벌어진 패킷 교환을 볼 수 있는 기능이다. 앞서 말했듯이 무수한 패킷을 보다 보면 어디가 끝이고 ACK이 제대로 왔는지 제대로 종료되어서 더 이상의 패킷 교환이 없었는지 파악하기 어렵다 때문에 이기능을 활용하였다.

세 번째로는 Data 교환에 관해서 분석해보고자 한다.



Address	Port	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	
117.52.128.45	443	85	24 k	39	18 k	46		6816
125.209.210.116	443	185	83 k	84	65 k	101		18 k
125.209.214.11	443	116	48 k	59	39 k	57		9330
125.209.218.225	443	127	40 k	67	33 k	60		7492
125.209.234.18	443	77	27 k	35	14 k	42		13 k
125.209.238.140	443	55	15 k	30	12 k	25		3128
125.209.254.186	443	513	537 k	400	527 k	113		10 k
168.63.244.178	443	40	17 k	18	15 k	22		2296
175.158.0.135	443	196	84 k	91	65 k	105		19 k
175.158.20.36	443	61	22 k	27	15 k	34		7310
175.158.20.37	443	53	21 k	26	15 k	27		6131
182.162.92.140	443	2,636	2958 k	2,121	2916 k	515		41 k
182.162.92.218	443	1,291	1318 k	977	1293 k	314		24 k
182.162.202.182	443	101	36 k	50	29 k	51		6977
183.111.4.34	443	144,011	192 M	137,759	191 M	6,252		450 k
183.111.26.110	443	212	162 k	146	157 k	66		5405
183.111.26.120	443	10,941	14 M	10,067	14 M	874		55 k
192.168.19.223	49984	62	41 k	26	2901	36		38 k
192.168.19.223	49983	27	9260	14	1221	13		8039
192.168.19.223	49986	23	6136	12	1114	11		5022
192.168.19.223	49985	166	150 k	41	3123	125		147 k
192.168.19.223	49987	23	6106	13	1168	10		4938
192.168.19.223	49988	20	5749	12	1082	8		4667
192.168.19.223	49989	1,271	1312 k	302	23 k	969		1289 k
192.168.19.223	49990	2,536	2929 k	455	35 k	2,081		2893 k
192.168.19.223	49991	19	5693	11	1026	8		4667
192.168.19.223	49992	20	5747	12	1080	8		4667
192.168.19.223	49993	20	5748	12	1080	8		4668
192.168.19.223	49994	20	5747	12	1080	8		4667
192.168.19.223	49995	21	5800	13	1134	8		4666
192.168.19.223	49996	22	7537	11	1063	11		6474
192.168.19.223	49997	59	25 k	30	3862	29		22 k
192.168.19.223	49999	23	5350	10	1004	13		4346

<그림6. endpoint>



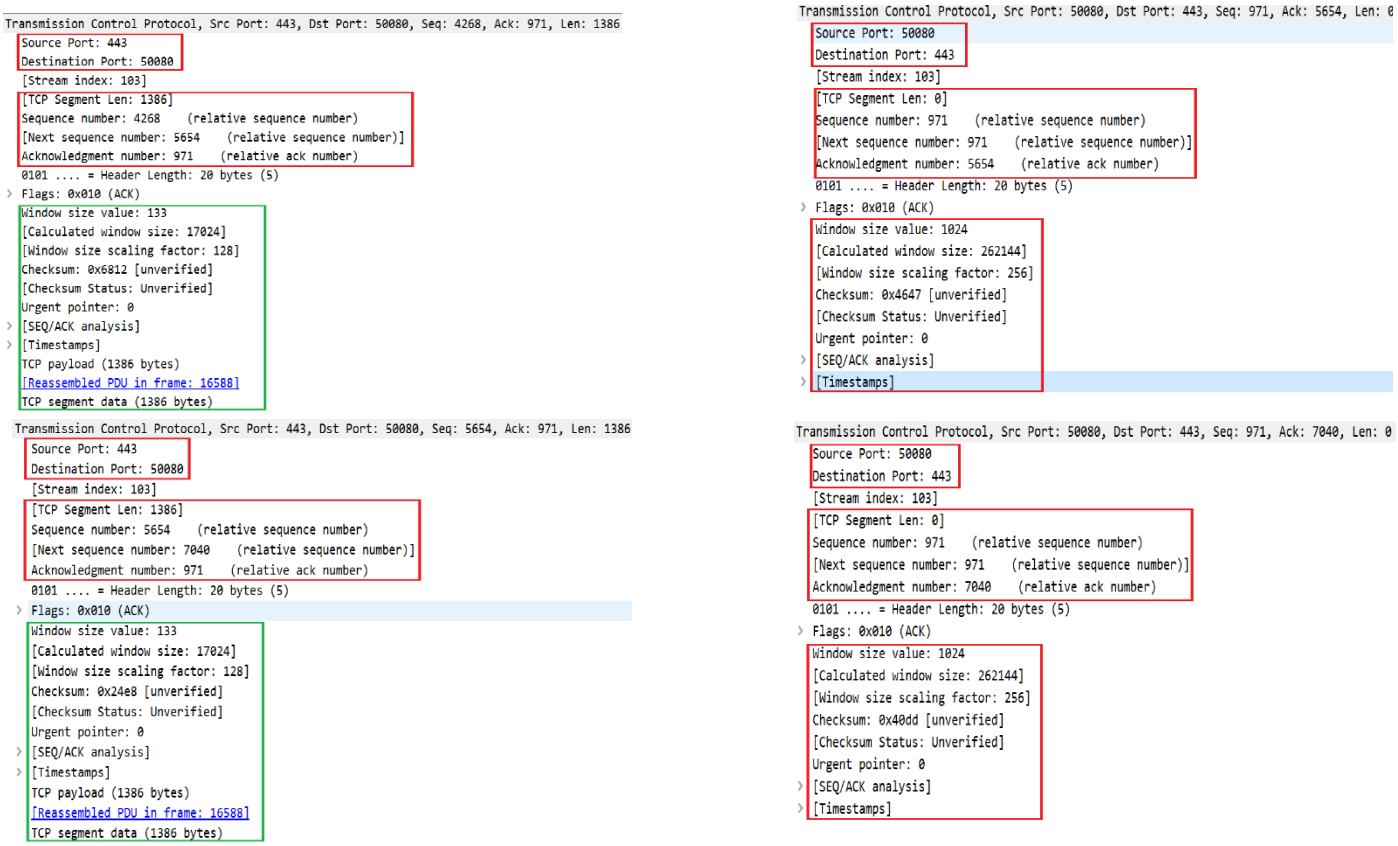
```
165.. 61.739622  192.168.19.223  183.111.4.34     TCP    66 50080 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
165.. 61.743726  183.111.4.34    192.168.19.223   TCP    66 443 → 50080 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1386 SACK_PERM=1 WS=128
165.. 61.743816  192.168.19.223  183.111.4.34     TCP    54 50080 → 443 [ACK] Seq=1 Ack=1 Win=262144 Len=0
165.. 61.744825  192.168.19.223  183.111.4.34     TLSv1..  268 Client Hello
165.. 61.748085  183.111.4.34    192.168.19.223   TLSv1..  1440 Server Hello
165.. 61.748085  183.111.4.34    192.168.19.223   TCP    1440 443 → 50080 [ACK] Seq=1387 Ack=215 Win=15744 Len=1386 [TCP segment of a reassembled PDU]
165.. 61.748135  192.168.19.223  183.111.4.34     TCP    54 50080 → 443 [ACK] Seq=215 Ack=2773 Win=262144 Len=0
165.. 61.748772  183.111.4.34    192.168.19.223   TLSv1..  1281 Certificate, Certificate Status, Server Key Exchange, Server Hello Done
165.. 61.748823  192.168.19.223  183.111.4.34     TCP    54 50080 → 443 [ACK] Seq=215 Ack=4010 Win=260864 Len=0
165.. 61.754947  192.168.19.223  183.111.4.34     TLSv1..  180 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
165.. 61.756241  183.111.4.34    192.168.19.223   TLSv1..  312 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
165.. 61.756280  192.168.19.223  183.111.4.34     TCP    54 50080 → 443 [ACK] Seq=341 Ack=4268 Win=262144 Len=0
165.. 61.776538  192.168.19.223  183.111.4.34     TLSv1..  684 Application Data
165.. 61.854868  183.111.4.34    192.168.19.223   TCP    1440 443 → 50080 [ACK] Seq=4268 Ack=971 Win=17024 Len=1386 [TCP segment of a reassembled PDU]
165.. 61.854990  192.168.19.223  183.111.4.34     TCP    54 50080 → 443 [ACK] Seq=971 Ack=5654 Win=262144 Len=0
165.. 61.855175  183.111.4.34    192.168.19.223   TCP    1440 443 → 50080 [ACK] Seq=5654 Ack=971 Win=17024 Len=1386 [TCP segment of a reassembled PDU]
165.. 61.855238  192.168.19.223  183.111.4.34     TCP    54 50080 → 443 [ACK] Seq=971 Ack=7040 Win=262144 Len=0
165.. 61.855696  183.111.4.34    192.168.19.223   TLSv1..  170 Application Data
165.. 61.855697  183.111.4.34    192.168.19.223   TLSv1..  399 Application Data
165.. 61.855789  192.168.19.223  183.111.4.34     TCP    54 50080 → 443 [ACK] Seq=971 Ack=7501 Win=261632 Len=0
165.. 61.872883  183.111.4.34    192.168.19.223   TLSv1..  689 Application Data
165.. 61.876564  183.111.4.34    192.168.19.223   TCP    1440 443 → 50080 [ACK] Seq=7501 Ack=1606 Win=18304 Len=1386 [TCP segment of a reassembled PDU]
165.. 61.876564  183.111.4.34    192.168.19.223   TCP    1440 443 → 50080 [ACK] Seq=8887 Ack=1606 Win=18304 Len=1386 [TCP segment of a reassembled PDU]
165.. 61.876729  192.168.19.223  183.111.4.34     TCP    54 50080 → 443 [ACK] Seq=1606 Ack=10273 Win=262144 Len=0
165.. 61.876898  183.111.4.34    192.168.19.223   TLSv1..  170 Application Data
165.. 61.876933  192.168.19.223  183.111.4.34     TCP    54 50080 → 443 [ACK] Seq=1606 Ack=10389 Win=261888 Len=0
165.. 61.877681  183.111.4.34    192.168.19.223   TCP    1440 443 → 50080 [ACK] Seq=10389 Ack=1606 Win=18304 Len=1386 [TCP segment of a reassembled PDU]
165.. 61.877681  183.111.4.34    192.168.19.223   TCP    1440 443 → 50080 [ACK] Seq=11775 Ack=1606 Win=18304 Len=1386 [TCP segment of a reassembled PDU]
165.. 61.877683  183.111.4.34    192.168.19.223   TLSv1..  170 Application Data
165.. 61.877683  183.111.4.34    192.168.19.223   TCP    1440 443 → 50080 [ACK] Seq=13277 Ack=1606 Win=18304 Len=1386 [TCP segment of a reassembled PDU]
```

<그림7. Data 교환>

```
.111.4.34      192.168.19.223   TCP   1440 443 → 50080 [ACK] Seq=4268 Ack=971 Win=17024 Len=1386 [TCP segment of a reassembled PDU]
.168.19.223    183.111.4.34     TCP     54 50080 → 443 [ACK] Seq=971 Ack=5654 Win=262144 Len=0
.111.4.34      192.168.19.223   TCP   1440 443 → 50080 [ACK] Seq=5654 Ack=971 Win=17024 Len=1386 [TCP segment of a reassembled PDU]
.168.19.223    183.111.4.34     TCP     54 50080 → 443 [ACK] Seq=971 Ack=7040 Win=262144 Len=0
```

<그림8. Data교환 패킷 중 일부>

이 부분에서 애를 먹었다. 왜냐하면 Wireshark의 사용이 아직 미숙하여 동영상을 시청하면서 패킷을 자세히 보지 않아 어떤 서버와 통신하는지 인지하지 못했다. 그래서 여기서도 endpoint의 기능을 사용했는데 table을 보면서 가장 많은 패킷 교환이 이루어진 서버를 찾아보았다. Table의 결과로 볼 때, 118.111.4.43 IP를 가진 Server의 443 번 port number를 가진 프로세스와 본인 pc의 50080번 port number를 가진 프로세스가 가장 패킷 교환이 많았다. 이 두 프로세스 간의 논리적통신과정 중 일부의 헤더를 분석해 보고자 한다.



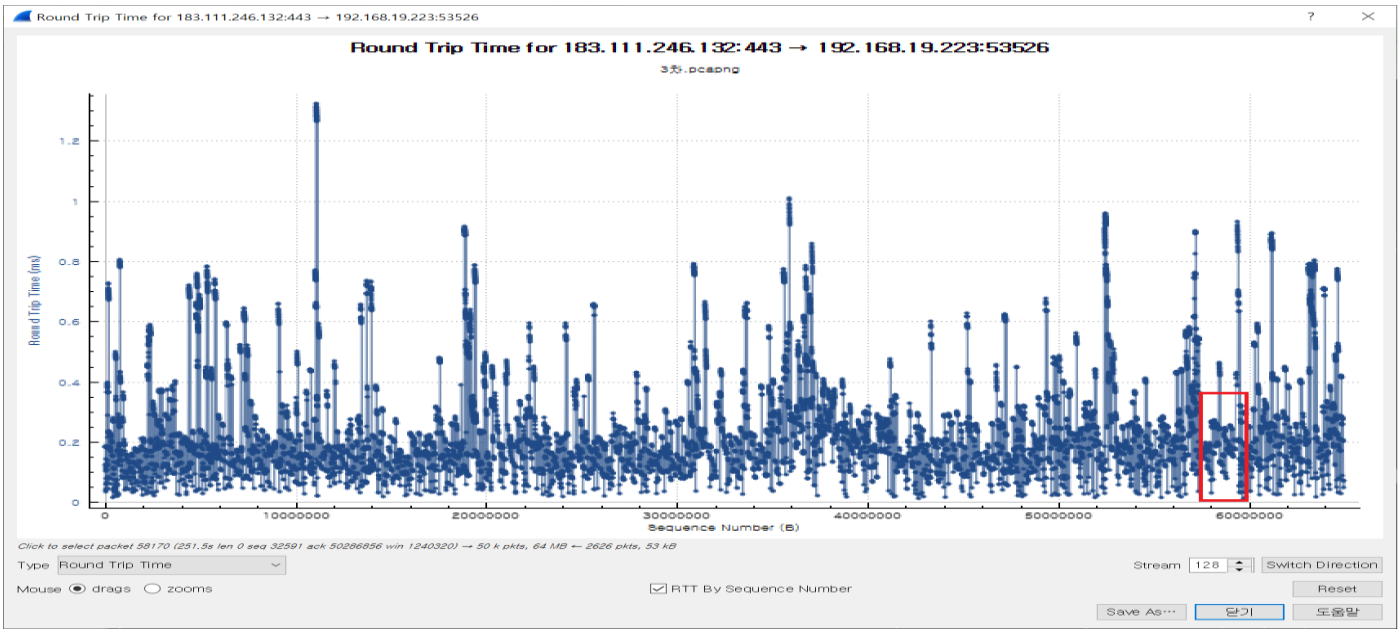
<그림9. 왼쪽은 서버가 본인pc로 보낸 segment 오른쪽은 본인pc가 서버로 보낸 segment>

보내는 메시지의 크기는 MSS값을 초과할 수 없는데, MSS는 그림 7에서 3-Way-Handshaking 하는 과정에 명시돼있다 (MSS=1460Byte) 위 그림의 Server가 보낸 segment를 보면서 알 수 있는 정보들을 취합해보면, source port number(443), destination port number(50080), seq num(4286), ack num(971이 뜻하는 바는 970byte까지 잘 받았음), next seq num인데 이 값은 다음에 시작할 순서번호를 의미한다 {현재 순서번호 + 데이터의 크기 = 4268 + 1386(MSS를 넘지 않음) = 5654} 그리고 Window size이다. Window size를 알려면 scaling factor와 size value를 곱해야 하는데 scaling factor는 통신이 효율적으로 이루어 지도록 네트워크 상태에 따라 수시로 증가하는 값이다 (기존 window size*2^n), size value는 TCP 수신 소켓의 수신 버퍼(Window size)를 16bit로 표현한 값이다. 이 것은 수업시간에 배웠던 혼잡제어나 흐름제어에서 이용되는 값이다. (receiver는 buffer의 크기를 명시한다.) 그리하여 window size는 고정이 아니라 수시로 바뀌는 값임을 알 게 되었다. 결과적으로, scaling factor값이 128(2^7)이고 window size가 133 이므로 133 * 128 = 17024라는 값이 나오게 되는 것이다. 요즘 기술의 혼잡제어는 이런 방식으로 Window size를 증가시킨다고 할 수 있다. 반대로 본인PC가 Server에 보낸 패킷을 살펴보면 본인PC는 Server에 데이터를 보내지 않는다. 받았다는 응답만 할 뿐이다. 순서번호와 ACK번호를 살펴보면 두 개가 다른 패킷인데 seq num이 971으로 동일하고 ACK번호는 각각 5654, 7040으로 다르다. 분명 seq num은 증가해야 하는데 보낸 데이터가 없으니 증가하지 않는 것이다. SYN Flag를 포함하는 패킷을 공부하다 보면 데이터가 없음에도 순서번호는 1씩 증가했다. 이런 이유로 순서번호는 무조건 증가해야 한다는 생각이 잘못되었음을 알게 됐다. 이런 이유로 본인이 수 많은 패킷들 중 위 4개를 선택한 이유이다. 다음은 교환 도중 발생하는 재전송에 관해 분석해 보겠다. 수업시간에 배운 재전송의 시나리오는 세 가지가 있었는데 retransmission, fast retransmission, spurious retransmission이다. 재전송된 패킷을 찾기에 앞서 얼마나 발생했는지 알기 위해 Wireshark의 Expert info 기능을 이용하였다. 재전송이 112(retransmission) + 1(spurious retransmission)+2(fast retransmission) 총 114번 발생하였

다. 생각보다 적게 발생한 것 같다. 혼잡한 시간대라서 재전송이 많이 발생할 줄 알았는데 꼭 그런 건 아니었다. 본격적으로 1번째 시나리오인 retransmission의 경우부터 살펴본다. 이것의 원인은 패킷 손실에 의한 timeout인데, sender가 segment를 보내고 윈도우에서 가장 왼쪽 패킷에 대한 타이머를 설정한다 이것을 보내고 이에 대한 ACK이 오는 시간이 타이머의 시간보다 오래 걸렸을 때, Sender는 동일 데이터를 1개만 재전송하게 된다. 다음은 retransmission된 segment이다. 이 세그먼트에 대한 RTO(Request Time Out)은 0.003168000 sec이었지만 이 시간보다 ACK이 도착하는 시간이 더 오래 걸렸음을 의미한다. 이런 이유로 상대순서번호가 57100930번째인 세그먼트는 재전송 되었다 타이머는 sample RTT(바로 전 패킷 RTT)와 Estimated RTT(처음부터 시작된 RTT의 평균)의 가중치로 계산한다. 그림12의 결과로 미루어 볼 때 sample RTT가 작아져서 Timer가 상대적으로 앞선 패킷들보다 작아져서 순서번호가 500000000의 후반대에서 재전송이 대량 발생한 이유를 알 수 있게 되었다.

```
Transmission Control Protocol, Src Port: 443, Dst Port: 53526, Seq: 57100930, Ack: 39761, Len: 1386
    Source Port: 443
    Destination Port: 53526
    [Stream index: 128]
    [TCP Segment Len: 1386]
    Sequence number: 57100930    (relative sequence number)
    [Next sequence number: 57102316    (relative sequence number)]
    Acknowledgment number: 39761    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window size value: 501
    [Calculated window size: 64128]
    [Window size scaling factor: 128]
    Checksum: 0xdba5 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ~ [SEQ/ACK analysis]
        [iRTT: 0.002989000 seconds]
        [Bytes in flight: 229610]
        [Bytes sent since last PSH flag: 2772]
      ~ [TCP Analysis Flags]
          > [Expert Info (Note/Sequence): This frame is a (suspected) retransmission]
            [The RTO for this segment was: 0.003168000 seconds]
            [RTO based on delta from frame: 64344]
  > [Timestamps]
    TCP payload (1386 bytes)
    Retransmitted TCP segment data (1386 bytes)
```

<그림 10. Retransmission의 패킷 헤더>



<그림 11. 빨간 박스는 순서번호 57100930패킷의 대략적 위치인데 저 시간에 RTT가 감소한 것을 알 수 있다.>

```
Transmission Control Protocol, Src Port: 443, Dst Port: 50063, Seq: 12274, Ack: 2471, Len: 11
    Source Port: 443
    Destination Port: 50063
    [Stream index: 86]
    [TCP Segment Len: 11]
    Sequence number: 12274    (relative sequence number)
    [Next sequence number: 12285    (relative sequence number)]
    Acknowledgment number: 2471    (relative ack number)
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window size value: 281
    [Calculated window size: 35968]
    [Window size scaling factor: 128]
    Checksum: 0xd727 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ~ [SEQ/ACK analysis]
        [iRTT: 0.010208000 seconds]
        [Bytes in flight: 11]
        [Bytes sent since last PSH flag: 11]
      ~ [TCP Analysis Flags]
          > [Expert Info (Note/Sequence): This frame is a (suspected) spurious retransmission]
          > [Expert Info (Note/Sequence): This frame is a (suspected) retransmission]
  > [Timestamps]
    TCP payload (11 bytes)
[Reassembly error, protocol TCP: New fragment overlaps old data (retransmission?)]
  > [Expert Info (Error/Malformed): New fragment overlaps old data (retransmission?)]
```

<그림 12. 동그라미 친 부분이 이 패킷의 특징인데 새로운 데이터가 old data를 overlap했다는 것을 알려준다.>

다음은 두 번째 시나리오 spurious transmission이다. 그림12는 spurious retransmission이 발생한 패킷의 헤더이다.

이 재전송의 원인은 retransmission과 동일하지만 차이가 있다. 수업시간에 배운 내용을 상기하면 조금 timeout이 된 경우인데, receiver가 data를 받았음에도 불구하고 재전송 되는 경우이다. Spurious retransmission의 빈도수는 굉장히 적었는데 두 번의 캡쳐 모두 1번씩 발생했었다. 이는 timer를 설정할 때 넉넉하게 설정해주지 않아서 발생한다. 세 번째 시나리오는 fast retransmission이다. 이 재전송이 발생하는 이유는 패킷 손실이다. 이를 해결하기 위하여 TCP는 받지 못한 Data에 대한 중복ACK을 보내도록 설계되어 있다 수업시간에 배운 내용을 상기하면, 중복ACK이 3번 이상 발생하면 Sender는 이에 대한 순서번호를 가진 data를 재전송하게 된다. 중복ACK이 발생했는지 확인하기 위해 Export info기능을 이용하여 순서번호 57073079번의 패킷들만 따로 필터링 해보았다.

643…	261.451490	192.168.19.223	183.111.246.132	TCP	66 [TCP Dup ACK 64277#10] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 L…
643…	261.452495	192.168.19.223	183.111.246.132	TCP	66 [TCP Dup ACK 64277#11] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 L…
643…	261.452625	192.168.19.223	183.111.246.132	TCP	66 [TCP Dup ACK 64277#12] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 L…
643…	261.452697	192.168.19.223	183.111.246.132	TCP	66 [TCP Dup ACK 64277#13] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 L…
643…	261.452760	192.168.19.223	183.111.246.132	TCP	66 [TCP Dup ACK 64277#14] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 L…
643…	261.452855	192.168.19.223	183.111.246.132	TCP	74 [TCP Dup ACK 64277#15] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 L…
643…	261.453472	192.168.19.223	183.111.246.132	TCP	74 [TCP Dup ACK 64277#16] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 L…
642…	261.450220	192.168.19.223	183.111.246.132	TCP	66 [TCP Dup ACK 64277#1] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 Le…
642…	261.450335	192.168.19.223	183.111.246.132	TCP	66 [TCP Dup ACK 64277#2] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 Le…
642…	261.450673	192.168.19.223	183.111.246.132	TCP	66 [TCP Dup ACK 64277#3] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 Le…
643…	261.451097	192.168.19.223	183.111.246.132	TCP	66 [TCP Dup ACK 64277#4] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 Le…
643…	261.451168	192.168.19.223	183.111.246.132	TCP	66 [TCP Dup ACK 64277#5] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 Le…
643…	261.451233	192.168.19.223	183.111.246.132	TCP	66 [TCP Dup ACK 64277#6] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 Le…
643…	261.451304	192.168.19.223	183.111.246.132	TCP	66 [TCP Dup ACK 64277#7] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 Le…
643…	261.451366	192.168.19.223	183.111.246.132	TCP	66 [TCP Dup ACK 64277#8] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 Le…
643…	261.451425	192.168.19.223	183.111.246.132	TCP	66 [TCP Dup ACK 64277#9] 53526 → 443 [ACK] Seq=39761 Ack=57073079 Win=1240320 Le…

<그림 13. 순서번호 57073079번에 대한 중복 ACK 16개>

```
Transmission Control Protocol, Src Port: 443, Dst Port: 53526, Seq: 57073079, Ack: 39761, Len: 1386
   Source Port: 443
   Destination Port: 53526
   [Stream index: 128]
   [TCP Segment Len: 1386]
   Sequence number: 57073079    (relative sequence number)
   [Next sequence number: 57074465    (relative sequence number)]
   Acknowledgment number: 39761    (relative ack number)
   0101 .... = Header Length: 20 bytes (5)
 > Flags: 0x010 (ACK)
   Window size value: 501
   [Calculated window size: 64128]
   [Window size scaling factor: 128]
   Checksum: 0x9409 [unverified]
   [Checksum Status: Unverified]
   Urgent pointer: 0
 ~ [SEQ/ACK analysis]
      [iRTT: 0.002989000 seconds]
      [Bytes in flight: 230996]
      [Bytes sent since last PSH flag: 1386]
   ~ [TCP Analysis Flags]
      > [Expert Info (Note/Sequence): This frame is a (suspected) fast retransmission]
      > [Expert Info (Note/Sequence): This frame is a (suspected) retransmission]
 ~ [Timestamps]
      [Time since first frame in this TCP stream: 96.590065000 seconds]
      [Time since previous frame in this TCP stream: 0.000001000 seconds]
   TCP payload (1386 bytes)
   TCP segment data (1386 bytes)
```

<그림 14. 순서번호 57073079에 대한 fast retransmission의 패킷 헤더 1386byte의 데이터를 포함한다.>

굉장히 많았다. 본인은 세 번 발생하면 곧바로 재전송을 하여 이정도로 많이 ACK을 보내는 줄은 몰랐는데, 추측해보건데 ACK의 패킷을 조사한 결과 데이터를 포함하지 않았다. 다시 말해, 재전송 되는 데이터보다 길이가 짧다. 그런 이유로 Sender는 3개 받으면 중복 ACK인 것을 인지하고 재전송 하게 되는데 Data가 재전송해서 in-flight하는 시간보다 ACK이 In-flight하는 시간이 짧기 때문에 발생한다고 생각한다.

지금부터는 special case인 RST flag를 가진 패킷과 keep-alive 패킷 헤더에 관한 분석을 시작한다 KEEP-ALIVE 패킷은 사용자 또는 OS가 지정하는 옵션이다 컨셉은 HOST A와 B가 서로 TCP 연결을 한다고 했을 때, 연결 설정 이후 서로 ESTABLISH 상태인 경우 지정된 시간 내에 서로 패킷 교환이 없을 경우가 데이터가 없는 패킷(PROBE)을 보내면서 정상으로 응답이 오면 연결유지, 그렇지 않은 경우는 상대방에게 장애가 생겼다고 판단하고 연결을 종료 시킨다. 위의 경우에 KEEP-ALIVE옵션을 사용하면 좋은데, A-B가 연결된 경우 B시스템이 장애로 인해서 RESTART될 경우 A는 KEEP-ALIVE설정에 의해 PROBE를 보내면 B는 RST(RESET) FLAG로 응답하는데 이 때 B의 연

결이 끊겼다는 것을 감지할 수 있다. 결론적으로 TCP에서는 상대방의 장애를 알 수 없기 때문에 KEEP-ALIVE설정으로 장애를 감지하는 것이다. 이 두 패킷은 유독 카카오톡 어플리케이션에서 상대적으로 많이 발생해서 이 패킷들을 이용해 보려 한다. 대화창을 갑자기 닫으면 위 패킷들이 생성되었다. 카카오톡 어플리케이션은 유독 사용자들이 갑자기 끄고 다시 시작하는 경우가 많기 때문에 이를 개발한 프로그래머가 KEEP-ALIVE옵션을 넣어 놓은 것으로 추측할 수 있다. 어플리케이션의 특성상 자주 종료되고 금방 재 시작되는 프로그램, 서버가 불안정하여 장애가 자주 발생하는 프로그램, 시간 민감성 어플리케이션에 사용되면 좋을 것이다.

<그림 15. KEEP ALIVE PACKETS 이후 연결 종료>

526…	209.292410	117.18.232.200	192.168.19.223	TCP	60 [TCP Keep-Alive] 443 → 53521 [ACK] Seq=9295 Ack=463 Win=147456 Len=0
526…	209.292461	192.168.19.223	117.18.232.200	TCP	54 [TCP Keep-Alive ACK] 53521 → 443 [ACK] Seq=463 Ack=9296 Win=260864 Len=0
526…	209.632882	117.18.232.200	192.168.19.223	TLSv1…	118 Application Data
526…	209.632883	117.18.232.200	192.168.19.223	TLSv1…	85 Encrypted Alert
526…	209.632994	192.168.19.223	117.18.232.200	TCP	54 53521 → 443 [ACK] Seq=463 Ack=9391 Win=260864 Len=0
526…	209.633624	117.18.232.200	192.168.19.223	TCP	60 443 → 53521 [FIN, ACK] Seq=9391 Ack=463 Win=147456 Len=0
526…	209.633684	192.168.19.223	117.18.232.200	TCP	54 53521 → 443 [ACK] Seq=463 Ack=9392 Win=260864 Len=0
526…	209.633801	192.168.19.223	117.18.232.200	TCP	54 53521 → 443 [FIN, ACK] Seq=463 Ack=9392 Win=260864 Len=0
526…	209.637838	117.18.232.200	192.168.19.223	TCP	60 443 → 53521 [ACK] Seq=9392 Ack=464 Win=147456 Len=0
526…	209.924909	117.18.232.200	192.168.19.223	TCP	60 [TCP Keep-Alive] 443 → 53522 [ACK] Seq=36970 Ack=724 Win=148480 Len=0
526…	209.925006	192.168.19.223	117.18.232.200	TCP	54 [TCP Keep-Alive ACK] 53522 → 443 [ACK] Seq=724 Ack=36971 Win=262144 Len=0
527…	228.853932	192.168.19.223	117.18.232.200	TCP	54 53522 → 443 [FIN, ACK] Seq=724 Ack=36971 Win=262144 Len=0
527…	228.857901	117.18.232.200	192.168.19.223	TCP	60 443 → 53522 [FIN, ACK] Seq=36971 Ack=725 Win=148480 Len=0

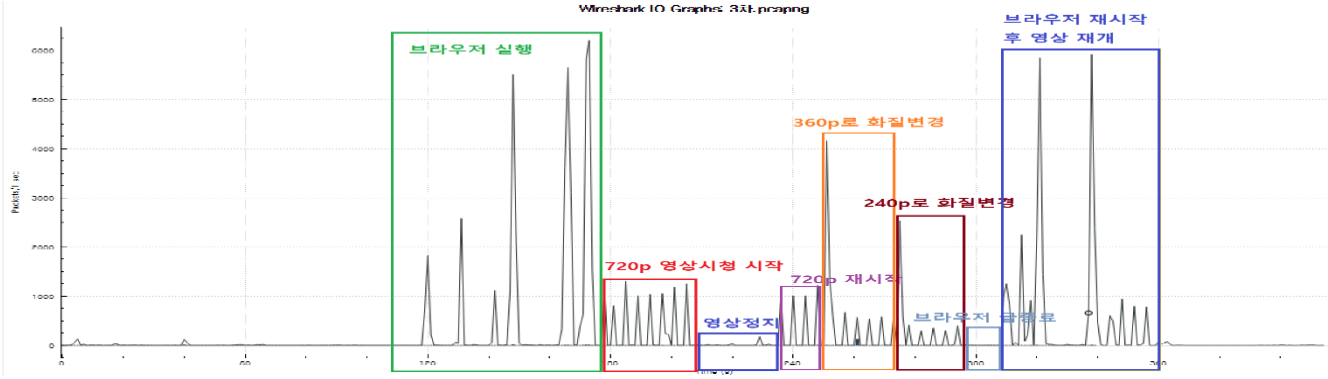
"RST FLAG는 어떤 경우에 헤더에 포함되는가?" RESET 어떤 이유로 연결이 갑자기 종료됐을 때, 다시 연결하기 위해 이 헤더가 사용된다. 이론대로 실험을 해보았는데, 영상시청 도중 갑자기 종료를 시켰을 때, 패킷 중 가장 마지막으로 발생하였다. 즉, 데이터가 교환되고 있는 와중에(영상이 끝나지 않고 시청 도중에) 갑자기 사용자가 연결을 끊어 버렸으니 비정상적으로 종료된 것이다.

711…	294.476965	192.168.19.223	183.111.246.132	TCP	54 53526 → 443 [ACK] Seq=53379 Ack=64921288 Win=1239040 Len=0
711…	296.542260	192.168.19.223	183.111.246.132	TCP	54 53526 → 443 [RST, ACK] Seq=53379 Ack=64921288 Win=0 Len=0

```
Flags: 0x014 (RST, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
>   .... .... .1.. = Reset: Set
    .... .... ..0. = Syn: Not set
    .... .... ...0 = Fin: Not set
```

<그림 16. RST패킷과 헤더 세부>

이제부터는 최종적으로 Traffic의 종합적 분석을 해보려고 한다. 트래픽의 변화를 관찰하기 위해서 Wireshark의 IO Graph 기능을 활용하였다. 보다 다양한 상황을 얻기 위해서 영상정지, 영상 앞 뒤로 건너뛰기, 영상 화질 속도 변경 그리고 에이전트 강제종료를 해보았다. 이런 상황 설정 때문에 위에서 대량의 재전송을 얻을 수 있었다.



<그림 17. IO 그래프 구간 별로 다양한 상황 설정>

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	52998	100.0	67837286	4121 k	0	0	0
Ethernet	100.0	52998	1.1	741972	45 k	0	0	0
Internet Protocol Version 4	100.0	52998	1.6	1059960	64 k	0	0	0
Transmission Control Protocol	100.0	52998	97.3	66035341	4011 k	43355	57416275	3488 k
Transport Layer Security	18.4	9735	95.6	64866381	3940 k	9642	64490070	3918 k
Data	0.0	1	0.0	1386	84	1	1386	84

<그림 18. Protocol hierarchy 기능으로 본 프로토콜 사용 백분율>

Wireshark · Expert Information · 2차 packet 수집 19.5.6.pcapng

Severity	Summary	Group	Protocol	Count
Warning	TCP Zero Window segment	Sequence	TCP	2
Warning	DNS response retransmission. Original response in ...	Protocol	DNS	1
Warning	DNS query retransmission. Original request in fram...	Protocol	DNS	1
Warning	Ignored Unknown Record	Protocol	TLS	13
Warning	This frame is a (suspected) out-of-order segment	Sequence	TCP	5
Warning	Previous segment(s) not captured (common at capt...	Sequence	TCP	7
Warning	Connection reset (RST)	Sequence	TCP	150
Note	This frame is a (suspected) spurious retransmission	Sequence	TCP	1
Note	ACK to a TCP keep-alive segment	Sequence	TCP	3
Note	TCP keep-alive segment	Sequence	TCP	3
Note	This frame is a (suspected) fast retransmission	Sequence	TCP	2
Note	This session reuses previously negotiated keys (Ses...	Sequence	TLS	9
Note	Duplicate ACK (#1)	Sequence	TCP	28
Note	This frame is a (suspected) retransmission	Sequence	TCP	112
Note	"Time To Live" != 255 for a packet sent to the Loca...	Sequence	IPv4	2
Chat	TCP window update	Sequence	TCP	52
Chat	Connection finish (FIN)	Sequence	TCP	298
Chat	GET /jk?c=28&p=r8G7rT_yJM3EVn4ep_hZd8oTschJJ...	Sequence	HTTP	82
Chat	Connection establish acknowledge (SYN+ACK): serv...	Sequence	TCP	233
Chat	Connection establish request (SYN): server port 443	Sequence	TCP	239

<그림19. 과제를 하면서 매우 유용하게 사용하였던 expert info 기능>

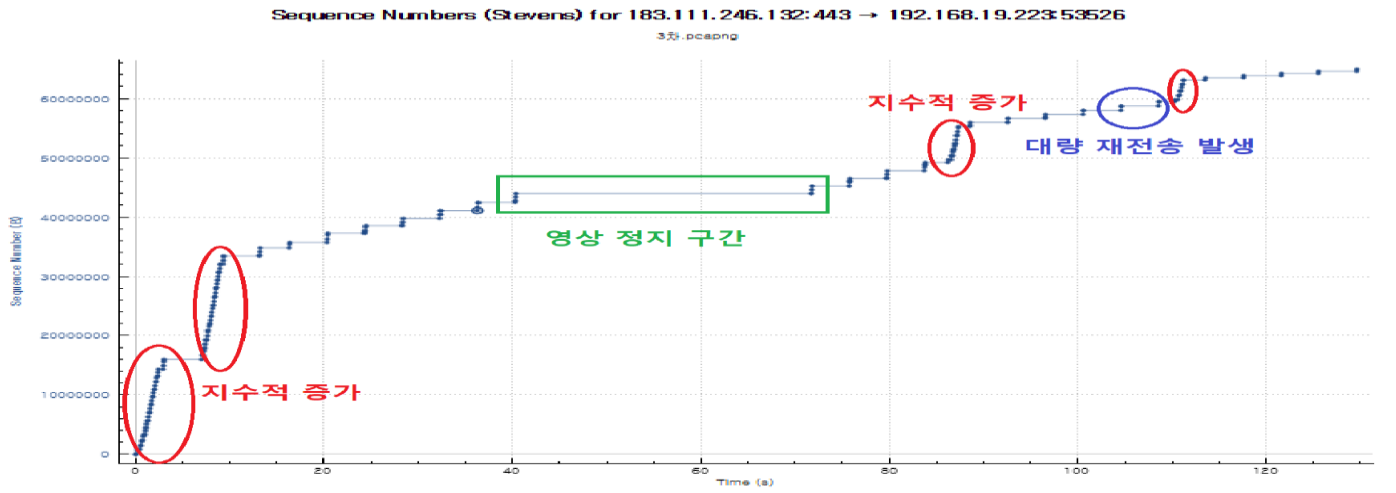
본인은 Streaming service를 이용했음에도 불구하고 UDP는 쓰이지 않았다. NAVER에서 제공하는 영상은 DASH 프로토콜을 기반으로 TCP 연결 했음을 알 수 있다. 캡쳐 하지는 않았지만 유튜브 시청시에는 UDP가 쓰인다, 왜냐하면 구글은 QUIC 프로토콜을 사용하여 UDP기반에서 영상서비스를 제공하기 때문이다. 브라우저를 실행시키면 패킷이 급상승하는데 객체를 굉장히 많이 받아와서 페이지를 그리기 때문이다. 페이지가 그려지면 받아오는 패킷이 적어진다. 다음은 영상 정지이다 받아오는 패킷이 굉장히 적어 지기 때문에 패킷 교환이 없는 것처럼 보인다. 다음은 화질 변경인데, 영상은 여러가지 버전을 만들어 놓는다. 왜냐하면 사용자마다 사용하는 링크에 속도가 차이가 있기 때문이다. 차례로 720p, 360p 그리고 240p인데 점점 작아지는 것을 볼 수 있다. 이는 받아오는 픽셀 수의 차이가 있어서 받아오는 데이터가 적이지기 때문이다. 그리고 그래프에는 거의 드러나지 않았지만 영상속도에 차이를 두면 간격이 조밀 해지는데 너무 짧게 실행하여 그래프에는 거의 드러나지 않았다. 본인은 더 빨리 재생해야 하므로 그만큼 필요한 패킷을 미리 버퍼링하고 있어야 하는 줄 알고 그래프가 요동칠 줄 알았으나 아니었다. 이 것을 보면서 TCP가 시간을 보장해주지는 않지만 Application이 시간민감성에 잘 대처하도록 설계 되어 있다는 것이 떠올랐는데 배운 내용으로 추측해보면, 간격이 좁아졌다는 의미는 받아오는 시간이 줄어들었음을 의미하기 때문에 PSH flag를 이용하여 다른 패킷들과 차별을 두어 먼저 처리하면 될 것 같다. 실제로 적지만 PSH 플레그가 들어간 패킷들이 종종 있었다. 다음은 영상시청 간에 발생한 PSH FLAG가 포함된 패킷이다.

```
Flags: 0x018 (PSH, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 1... = Push: Set
    .... .... .0.. = Reset: Not set
    .... .... ..0. = Syn: Not set
    .... .... ...0 = Fin: Not set
    [TCP Flags: ·····AP···]
```

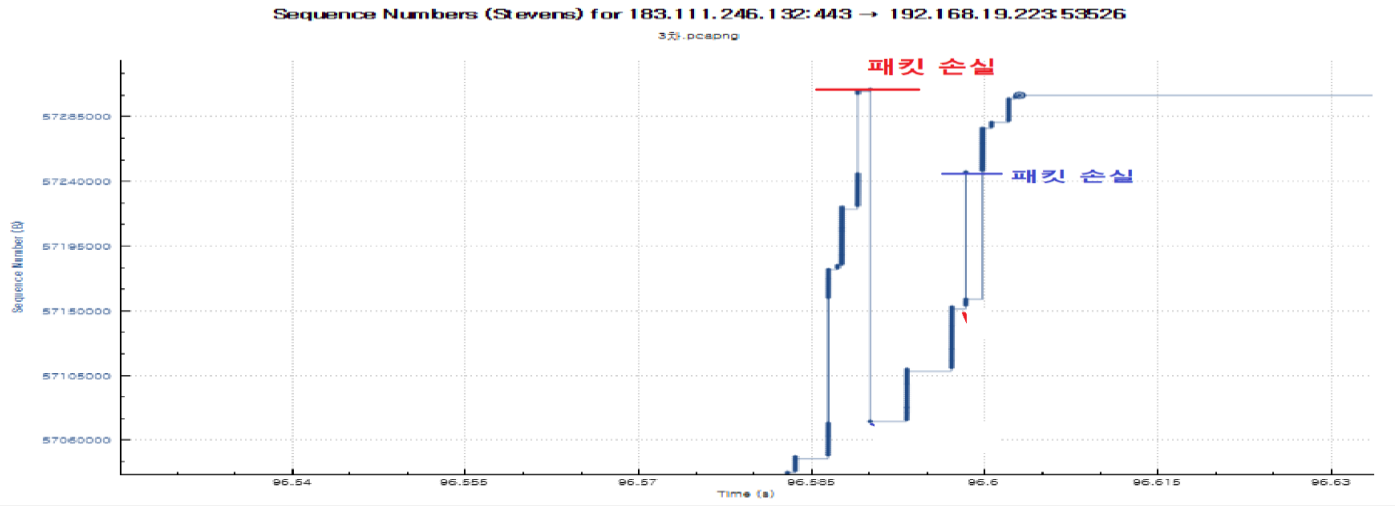
<그림 20. PSH FLAG Packet>

다음으로는 에이전트 급종료이다. 저 부분에서 RST패킷이 발생하였다. 데이터 교환 중 갑자기 클라이언트가(본인

PC) 연결을 끊어 비정상적으로 종료됐기 때문이다. 그리고 노트북을 들고 노트북을 사용하는 학우들이 많은 곳으로 이동하였는데, 갑자기 대량의 재전송이 발생하였다. 원인을 꼭 그렇다고 단정지을 수는 없지만 아마 학우들이 이미 보낸 패킷보다 밀려서 큐에서 본인이 보낸 패킷이 버려졌을 수도 있다. Drop strategy에서 Drop tail 방식을 채택한 것 같다. 다음으로 본인이 하고 싶은 것은 TCP 혼잡제어에 관한 분석이다. 이를 위해 TCP Stream graph기능을 이용해 보려 한다. 분석을 위해 사용할 개념은 slow start와 congestion avoidance 2가지 phase이다. 그래프를 전체적으로 분석해보고 다음은 재전송 대량발생은 순서번호 57000000 ~ 57200000 사이였다. 그 정보를 토대로 ZOOM해서 어떤 현상이 발생하는지 보았다. 수업 시간에 배운 대로 기술이 발전할수록 AIMD방식에서 어떻게 변하는지 어떻게 WINDOW SIZE를 키우는 크기를 더 빠르게, 감소폭을 더 작게 만드는지 관찰하였다.



<그림 21 TCP Stream graph>



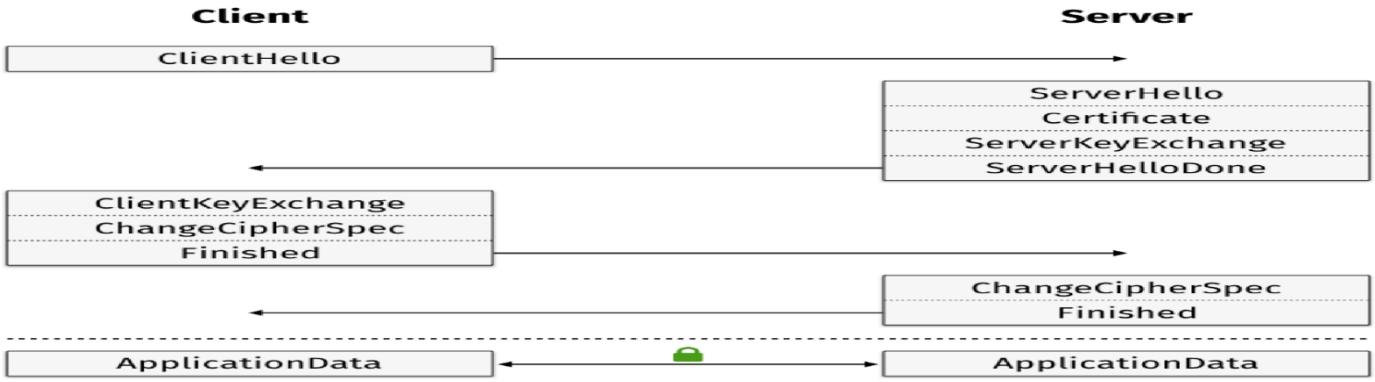
<그림 22. 손실된 시점 TCP Stream graph>

전체 그래프로만 관찰하면 어디서 손실이 발생했는지 관찰하기 힘들었다. 이것은 감소가 배수적으로 발생하지 않는다는 증거이다. 배수적으로 감소했다면, 확실히 눈에 확 띄었을 것이다. 실제로 ZOOM해서 확인해본 결과도 마찬가지였다. 순서번호가 갑자기 뚝 떨어짐을 알 수 있다. 이는 패킷이 손실되었고 다시 보냈는데 패킷 손실 좌우 로 봤을 때 증가폭이 오른쪽 손실이 더 작음을 알 수 있다 이는 손실 이후 cwnd를 천천히 올린다는 이론과 맞아떨어진다. 그리고 대량 발생구간에서 fast retransmission과 retransmission이 발생하였는데 두 경우 순서번호가 본래로 복구되는 시간이 많이 차이 났다. 이것은 수업시간에 배운 critical문제이다. Duplicate ack은 패킷 낱개지만, 그에 비해 retransmission은 패킷 여러 개가 손실되어 아예 ack이 오지 못하거나 ack이 손실되기 때문인데, 이것은 네트워크가 혼잡함을 의미하기 때문에 과거에는 retransmission이 발생하면 cwnd를 1로 설정하고, duplicate ack인 경우 cwnd를 1/2로 감소시킨 것이었다. 현재는 그렇게 극단적으로 감소시키지는 않는다. 마지막으로, 혼잡제어관찰 간에 느낀 점은 세부적으로 증가폭과 감소폭 부분은 바뀌었지만 slow start, congestion avoidance의 concept는 변하지 않은 것 같다.

3) 새로 알게 된 지식

Traffic분석을 하면서 몰랐던 것들이 많이 나왔는데 그 중에서 TLSv1.2, 4-way handshaking이다. TLSv1.2는 본인이 분명 TCP ONLY로 필터링을 했음에도 불구하고 계속 나와서 인터넷 검색을 해보다가 알게 됐다. TLS(Transport layer security)는 도청, 간섭 그리고 위조를 방지위해 설계되었고, 암호화를 통해 최종단의 인증, 통신기밀성을 유지시킨다. V1.2는 버전을 뜻한다. TCP와 공통점은 handshaking후에 시작되는 프로토콜이고, TCP와 차이점은 Application data가 암호화된다는 점이다.

```
16098 164.868853   192.168.19.223   183.111.246.132  TLSv1.2    268 Client Hello
16099 164.868871   192.168.19.223   183.111.246.132  TLSv1.2    268 Client Hello
16100 164.873101   183.111.246.132  192.168.19.223   TLSv1.2   1440 Server Hello
16101 164.873295   192.168.19.223   183.111.246.132  TCP         54 53527 → 443 [ACK] Seq=215 Ack=1387 Win=262144 Len=0
16102 164.873936   183.111.246.132  192.168.19.223   TCP       1440 443 → 53527 [ACK] Seq=1387 Ack=215 Win=15744 Len=1386 [TCP segment of a reassembled PDU]
16103 164.873937   183.111.246.132  192.168.19.223   TLSv1.2   1291 Certificate, Certificate Status, Server Key Exchange, Server Hello Done
16104 164.874093   192.168.19.223   183.111.246.132  TCP         54 53527 → 443 [ACK] Seq=215 Ack=4010 Win=262144 Len=0
16105 164.874531   183.111.246.132  192.168.19.223   TLSv1.2   1440 Server Hello
16106 164.874531   183.111.246.132  192.168.19.223   TCP       1440 443 → 53526 [ACK] Seq=1387 Ack=215 Win=15744 Len=1386 [TCP segment of a reassembled PDU]
16107 164.874657   192.168.19.223   183.111.246.132  TCP         54 53526 → 443 [ACK] Seq=215 Ack=2773 Win=262144 Len=0
16108 164.875428   183.111.246.132  192.168.19.223   TLSv1.2   1291 Certificate, Certificate Status, Server Key Exchange, Server Hello Done
16109 164.875533   192.168.19.223   183.111.246.132  TCP         54 53526 → 443 [ACK] Seq=215 Ack=4010 Win=260864 Len=0
16110 164.881968   192.168.19.223   183.111.246.132  TLSv1.2    180 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
16111 164.882503   192.168.19.223   183.111.246.132  TLSv1.2    766 Application Data
16116 164.885873   183.111.246.132  192.168.19.223   TLSv1.2    312 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
16117 164.885988   192.168.19.223   183.111.246.132  TCP         54 53526 → 443 [ACK] Seq=1053 Ack=4268 Win=262144 Len=0
16118 164.885989   192.168.19.223   183.111.246.132  TLSv1.2    180 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
16124 164.888492   183.111.246.132  192.168.19.223   TLSv1.2    831 Application Data
16125 164.888533   192.168.19.223   183.111.246.132  TCP         54 53526 → 443 [ACK] Seq=1053 Ack=5045 Win=261120 Len=0
16126 164.890133   183.111.246.132  192.168.19.223   TLSv1.2    312 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
16127 164.890277   192.168.19.223   183.111.246.132  TCP         54 53527 → 443 [ACK] Seq=341 Ack=4268 Win=261632 Len=0
16132 164.901616   192.168.19.223   183.111.246.132  TLSv1.2    766 Application Data
16133 164.906751   183.111.246.132  192.168.19.223   TCP       1440 443 → 53526 [ACK] Seq=5045 Ack=1765 Win=18560 Len=1386 [TCP segment of a reassembled PDU]
16134 164.906754   183.111.246.132  192.168.19.223   TCP       1440 443 → 53526 [ACK] Seq=6431 Ack=1765 Win=18560 Len=1386 [TCP segment of a reassembled PDU]
16135 164.906755   183.111.246.132  192.168.19.223   TLSv1.2    170 Application Data
```



<그림 23. TLS의 Handshaking 위는 본인 pc에서 발생한 것 아래는 인터넷에서 찾아본 handshaking과정>

4) 관찰 후 소감

이론이 실상에서 어떻게 쓰이는지를 보면서 과제를 하면서 잘못 이해하고 있는 부분도 많았고, 새로 알게 된 부분도 많았다. 예상을 벗어나는 부분도 많았고 여러모로 얻어가는 게 많은 과제였다. 제일 흥미로웠던 부분을 하나 꼽아보자면 혼잡제어 부분이다. 분석하는데 제일 어려웠고 애를 많이 먹었다. 재전송 방식마다 그래프가 달라지고 알고 있던 개념과는 달리 지수적 증가가 2번 연속으로 나오는 부분이 가장 흥미로웠다. 또 하나 흥미로웠던 부분은 FLAG가 배운 것 보다 더 많았다는 점이다. 수업 시간에 배운 FLAG는 U A P R S F 6 가지였는데, 과제를 하면서 아래 그림과 같이 4개의 FLAG가 더 있다는 것을 알게 되었다.

```
Flags: 0x010 (ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
```

<그림 24. 새로 알게 된 FLAG>

마지막으로 제일 애를 먹었던 점은 Wireshark를 처음 사용해 봐서 시행착오가 많았던 점이다.