# Wireshark를 이용한 transport layer traffic 분석

제출자 : 수학과 201621138 허창현

## 1. 개요

본 과제는 wireshark를 이용하여 transport layer traffic을 분석하는 과제이다. 따라서, 2에는 wireshark의 어떤 기능을 이용하여 원하는 traffic들을 분석하였는지 기술하였으며, 3-1에서는 UDP를 사용하는 사례(NBNS)를 찾은 뒤, 왜 UDP를 사용하는지에 대해서 분석하고 추측해 보았다.

3-2, 3-3에서는 사용되는 TCP segment의 헤더가 비슷하기에 기본적인 TCP 헤더들에 대해서 가볍게 정리한뒤 각 항목에 따라 중점적으로 봐야할 TCP헤더들을 해당항목에 상세히 기술하였다. 3-2-1, 3-2-2의 connection set up, connection close에서는 사용되는 flag들을 위주로, 3-2-3의 application data를 포함한 TCP segment에서는 segment len, sequence number, next sequence number를 위주로, 3-3에서는 window size를 위주로 기술하였다.

끝으로, 4에서는 과제를 수행하는 과정에서 알게된 새로운 사실들에 대해서 서술하였다.

## 2. 관찰 방법

5월 10일, 5월 20일, 5월 24일 각각 집 앞 카페에서 30분정도 wireshark를 키고, 들어오는 packet들을 육안으로 먼저 관찰했다. 예를 들어 RSTsegment나 retransmission segment들이 들어오는지 관찰했다.

5월 24일을 끝으로 필요한 segment들이 모두 들어왔다고 생각하고, 심층분석을 시작하였다. 우선, 3-1의 UDP에 대해서는 wireshark의 기능(statistics->conversation-> conversation type에서 UDP만을 체크)을 이용하여 UDP들만 보이게 한 뒤, 그중 하나를 골라 [follow stream]을 누름으로써, 관찰했다.

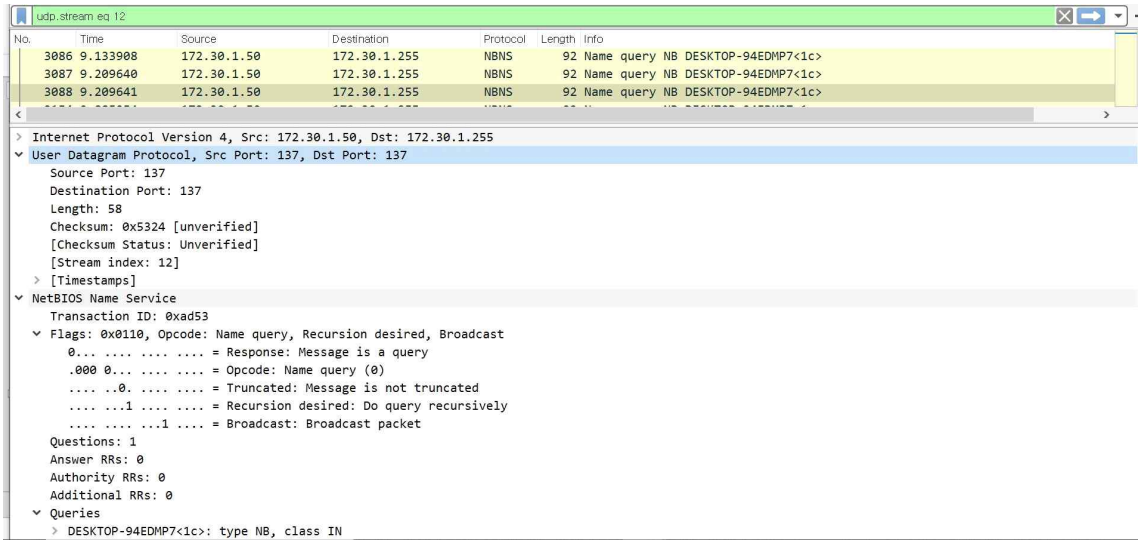
3-2-1, 3-2-2에 대해서도 wire shark의 기능(statistics->conversation-> conversation type에서 TCP만을 체크)을 이용하여서 segment들을 캡쳐할 수 있었다.

3-2-3에서는 application data를 포함한 segment를 찾아야함으로, wire shark의 기능(statistics->conversation-> conversation type에서 TCP만을 체크)을 사용하여 TCP들만 보이게 한뒤, application data를 포함한 segment를 찾아야함으로(아마 application data를 실어야함으로, 데이터 이동이 클것이라는 추측하에), bytes수가 큰 쪽을 골라 [follow stream]을 하여 원하는 segment를 찾아냈다.

3-3, 3-4 또한 해당 기능을 이용하여 TCP들만 보이게 한 뒤, 아마도 정보의 교환이 많이 이어날 때 이와 같은 오류들이 발생할 확률이 올라갈 것이라는 추측하에 bytes수가 큰 쪽 위주로 [follow stream]을 하여 원하는 segment들을 찾아냈다. 또한, retransmssion을 관찰할 EO는 보다 자세히 관찰하기 위해서 statistics의 TCP stream에서 time sequence(stevens)를 클릭했을 때 나오는 그래프를 이용하여 보다 자세히 관찰해 볼 수 있었다.

3. Traffic 분석

1) UDP 분석




< 그림 1, UDP를 사용하는 사례(NBNS)>

<그림 1>의 UDP header부분에 대해서 먼저 자세히 살펴보겠다. 우선 source port와 destiantion port가 137이다. 즉, 발신포트와 수신포트 모두 137번 포트를 사용한다는 것을 알 수 있다. 다음으로 나오는 header는 length이다. 이 header는 UDP header를 포함한 packet전체의 길이를 byte단위로 표시해주는 header이다. 여기서는 58로 나와있음으로, <그림 1>에 나온 packet 전체의 길이는 58byte임을 알 수 있다. 3번째로 나오는 header는 Checksum은 header와 data에 변형오류를 검출하기 위해서 사용되는 header이다.

NBNS는 NetBIOS Name service의 약자로 네트워크에 연결된 다른 컴퓨터들 상호간의 이름이 충돌되지 않도록 도와주는 역할을 한다. 일반적으로 UDP를 전송 protocol로 사용하며, NBNS traffic의 TCP port는 137번인 경우가 대부분이다.(<그림 1>에서도 port 번호가 137인 것을 위에서 확인했었다.)

NBNS가 UDP를 사용하는 이유를 알기 위해 먼저 NetBIOS의 두가지 통신모드(세션, 데이터그램)에 대해서 알아보겠다.

세션모드는 두 개의 컴퓨터들이 대화를 위해 연결을 위해 맺을 수 있도록 해주는 통신모드이다. 대화를 위한 연결이기에 데이터의 확실한 전달 여부보다는 최신 데이터가 도착하는 것이 더 우선시되기에 UDP를 사용하는 것 같다.

데이터그램 모드는 근거리통신망에 상의 모든 컴퓨터들에게 메시지를 뿌리는 기능을 지니고 있다. 모든 컴퓨터들에게 메시지를 뿌리는 기능을 하기에 모든 메시지에 대해서 일일이 전달 여부를 확인하는 TCP보다는 UDP가 더 효율적이기에 UDP를 사용하는 것 같다.

2) TCP 분석

- TCP segment header에 대한 정리

현재 3-2(Traffic분석중 TCP분석)에서 주로 쓰이는 헤더들은 다음과 같다.

- source port : 송신측의 응용프로세스를 구분하는 port번호
- destination port : 수신측의 응용프로세를 구분하는 port번호
- sequence number : 송신된 데이터의 순서번호
  - next sequence number : sequence number + 보낸 데이터의 길이
- Ack number : 수신된 데이터의 byte수 + ACK(다음번에 기대되는 순차번호)
- Header length : 헤더 크기
- SYN : 연결 요청시 사용
- ACK : 확인 응답할 때 사용
- URG : 긴급 데이터를 표시할 때 사용
- PSH : 버퍼링된 데이터를 가능한한 빨리 상위 계층 응용프로그램에 전달할 때 사용
- FIN : 상대방에게 연결 종료 의사를 표시할 때 사용
- RST : 연결 끊기
- window : TCP 수신버퍼를 바이트 크기로 표시, 수신자가 사용가능한 버퍼공간을 표시
- checksum : TCP segment에 포함되는 프로토콜 헤더와 데이터 모두에 대한 변형 오류를 검출하기 위해 사용
- urgent pointer : 긴급 데이터가 들어있는 위치를 표시

- connection setup 단계에서 교환하는 TCP segment header 분석

| | | | | | |
|---|---|---|---|---|---|
| 15 | 2.178331 | 172.30.1.50 | 211.115.106.205 | TCP | 66 63076 → 80 [SYN] Seq=4274217927 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM= |
| 16 | 2.183173 | 211.115.106.205 | 172.30.1.50 | TCP | 66 80 → 63076 [SYN, ACK] Seq=1740043201 Ack=4274217928 Win=29200 Len=0 MSS=14 |
| 17 | 2.183305 | 172.30.1.50 | 211.115.106.205 | TCP | 54 63076 → 80 [ACK] Seq=4274217928 Ack=1740043202 Win=131328 Len=0 |

<그림 2-1, connection setup 사례, 3-way handshake>

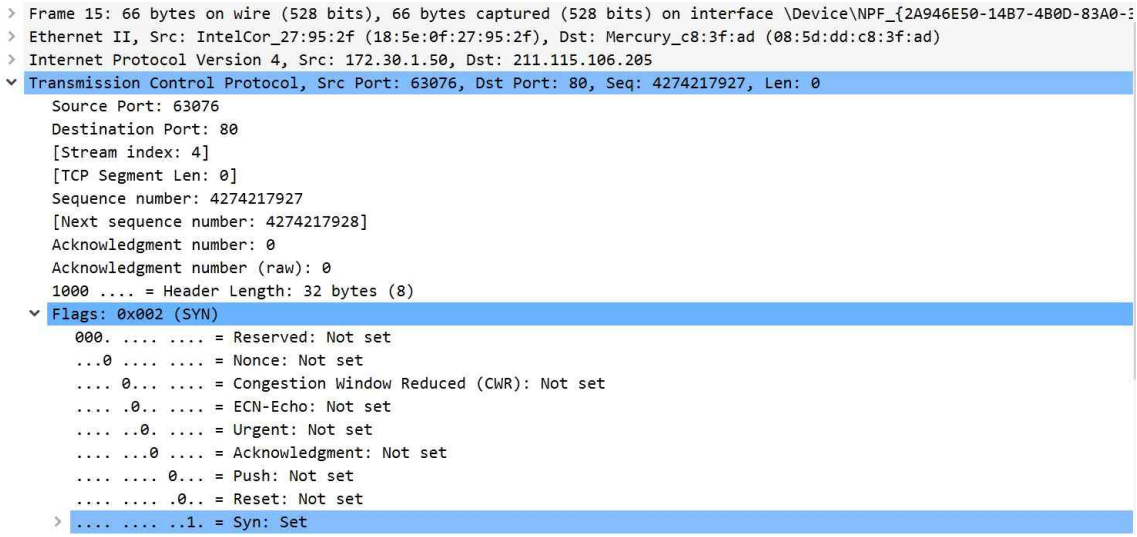
Connection setup은 3-way handwhake과정을 통해서 이뤄진다. 3-way handshake은 다음과 같은 과정을 통해서 이뤄진다.

먼저 client가 server에 연결요청 메시지(SYN)을 전송한다. 이 과정에서의 client의 연결 상태는 closed이고, server의 연결상태는 listen에서 SYN_RCV로 변경된다. <그림 2-1>의 첫 번째 줄을 보면 나의 IP주소(172.30.1.50)에서 server의 IP주소(211.115.106.205)를 destination으로 SYN segment가 보내졌음을 알 수 있다.

두 번째 과정은 server에서 client로 SYN에 대한 확인 메시지(ACK)과 client도 연결상태를 열어달라는 의미의 메시지(SYN)을 합쳐(SYN+ACK)을 보내게 된다. 이때 client의 연결상태는 closed에서 established로 바뀌고. server의 연결 상태는 SYN_RCV이다.

세 번째 과정은 client가 server로 연결 요청 수락을 의미하는(ACK)메시지를 보낸다. 이때 client의 연결상태는 established이고, server의 연결상태는 SYN_RCV에서 established로 변경된다.

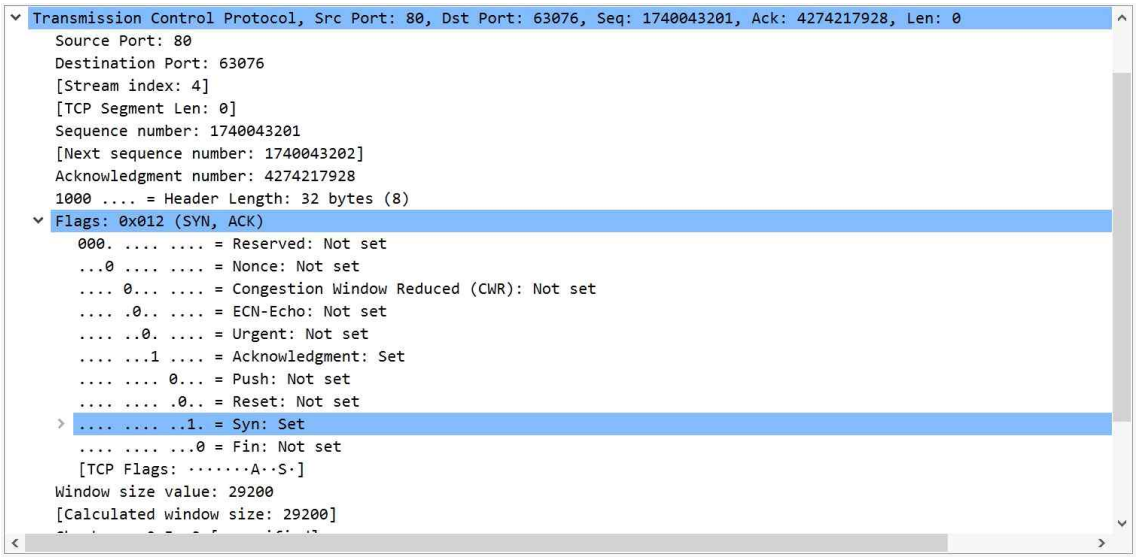
이렇게 3개의 과정을 거치면 최종적으로 client와 server모두 connection을 established한 상태가 되어, connection을 이뤘음을 알 수 있다.




<그림 2-1-1, connection setup syn segment>

위 그림은 3-handshake의 첫 번째 과정인 client -> server(SYN)이다. 우선 client의 port는 source port에 적힌 63076이며, server의 port는 destination port에 적힌 80이다. 또한, SYN message의 목적에 맞게 flag의 Syn 부분이 1로 되어있음을 확인할 수 있다.

여기서 가장 주시할 점은 sequence number이다. 현재 sequence number는 4274217927이며 아래에 적힌 next sequence number는 4274217928은 뒤에 따라온 ACK message에 ACK number로 사용되어질 것이다.

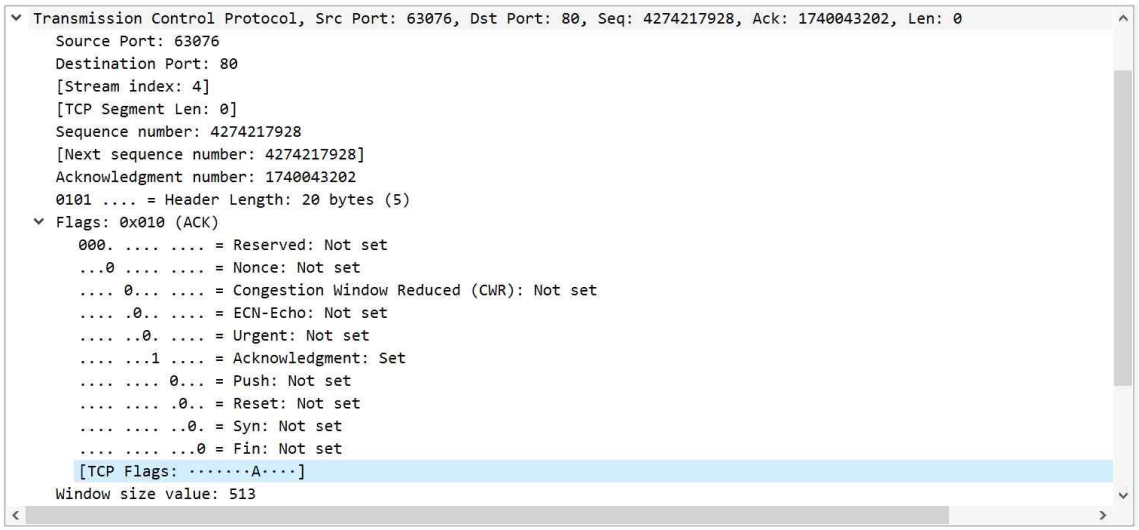



<그림 2-1-2, connection setup syn, ack segment>

위 그림은 3-handshake의 두 번째 과정인 server -> client(SYN+ACK)이다. server의 port, source port는 80이며, client의 port destination port는 63076이다. 앞서 <그림 2-1-1>에서 말했던 client와 server의 port번호들이 일치함을 확인할 수 있다. 또한, SYN+ACK메시지임으로 flag의 SYN부분과 ACK부분이 1로 되어있다.

여기서 ACK number는 앞선 <그림 2-1-1>에서 예측한 대로 4274217928임을 확인할 수 있었다. SYN에 대한 sequence number는 1740043201이며, next sequence number는

1740043202이다. 이를 토대로 예측해보면 후에 등장할 ACK 메시지에 대한 ACK number은 17400432O2가 될 것이다.

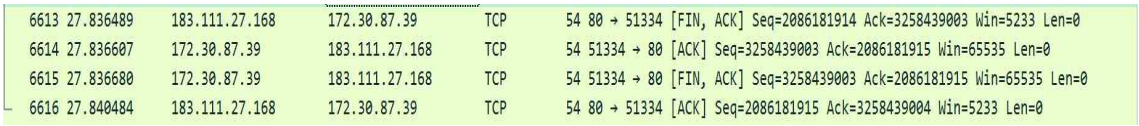



<그림 2-1-3, connection setup ack segment>

위 그림은 3-handshake의 세 번째 과정인 client -> server(ACK)이다. 우선 client의 port는 source port에 적힌 63076이며, server의 port는 destination port에 적힌 80이다. 또한, ACK message임으로 flag의 ACK 부분이 1로 되어있음을 확인할 수 있다.

여기서 sequence number는 <그림 2-1-1>의 next sequence number이자, <그림 2-1-2>의 ACK number인 42742179Z8이며, ACK number는 앞서 예측한 대로 1740043202이다.

- connection 종료 단계에서 교환하는 TCP segment header 분석



| | | | | | |
|---|---|---|---|---|---|
| 6613 | 27.836489 | 183.111.27.168 | 172.30.87.39 | TCP | 54 80 → 51334 [FIN, ACK] Seq=2086181914 Ack=3258439003 Win=5233 Len=0 |
| 6614 | 27.836607 | 172.30.87.39 | 183.111.27.168 | TCP | 54 51334 → 80 [ACK] Seq=3258439003 Ack=2086181915 Win=65535 Len=0 |
| 6615 | 27.836680 | 172.30.87.39 | 183.111.27.168 | TCP | 54 51334 → 80 [FIN, ACK] Seq=3258439003 Ack=2086181915 Win=65535 Len=0 |
| 6616 | 27.840484 | 183.111.27.168 | 172.30.87.39 | TCP | 54 80 → 51334 [ACK] Seq=2086181915 Ack=3258439004 Win=5233 Len=0 |

<그림 2-2, connection close, 4-way handshake>

connection close는 4-way handshake을 통해서 이루어진다. 4-way handshake의 과정은 다음과 같다. <그림 2-2>를 보게 되면, server에서 먼저 연결을 끊기를 요청하였기에, 이를 기반으로 4-way handshake에 대해서 설명을 시작하겠다.

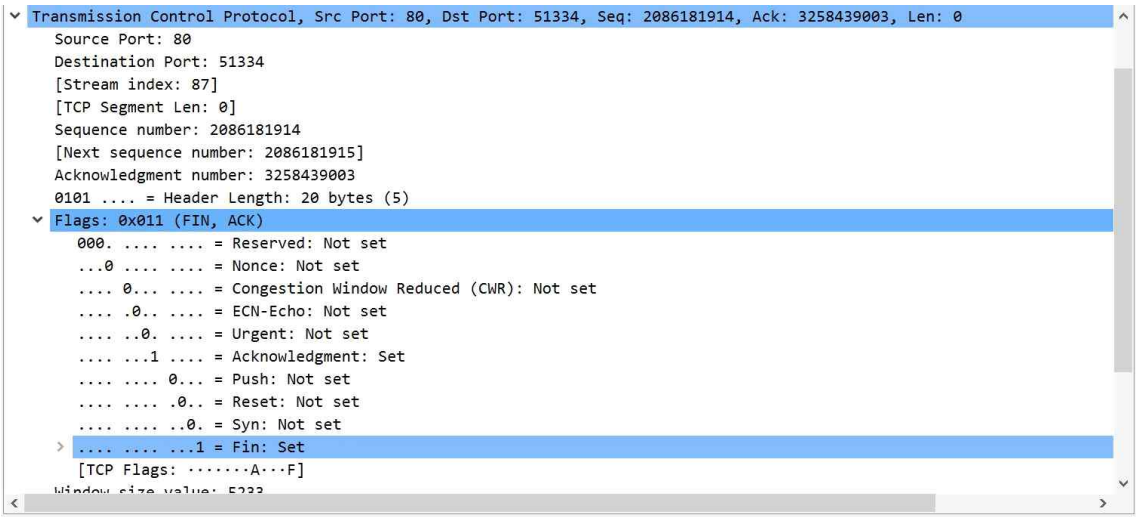
우선, server에서 client로 FIN, ACK메시지를 전송한다. 이때 server의 연결 상태는 established에서 FIN_WAIT1이 되며, client의 연결상태는 메시지를 받음과 동시에 established에서 CLOSED_WAIT상태가 된다. <그림 2-2>의 첫 번째 줄을 보면 server의 IP주소(183,111.27.168)에서 나의 IP주소(172.30.87.39)를 destination으로 FIN, ACK 메시지가 보내졌음을 확인할 수 있다.

두 번째 과정은 client에서 server로 ACK메시지를 보내는 과정이다. 이 메시지가 도착한 후 server는 FIN_WAIT2 상태가 된다.

세 번째 과정이 시작되게 된다. client는 server로 연결 종료에 합의한다는 의미의 FIN,ACK 메시지를 보낸다. 이 메시지를 받으면, server는 TIME_WAIT상태에 놓이게 된다.

마지막 과정은 server에서 client로 ACK메시지를 보내는 과정이다. 이 메시지가 보내지게 되면, server의 연결상태는 TIME_WAIT에서 closed가 된다. 이 메시지를 client가 수신하면 client의 연결상태 또한 closed가 된다.

양측의 연결 상태가 모두 closed가 되었으므로, 연결이 종료되었음을 알 수 있다.



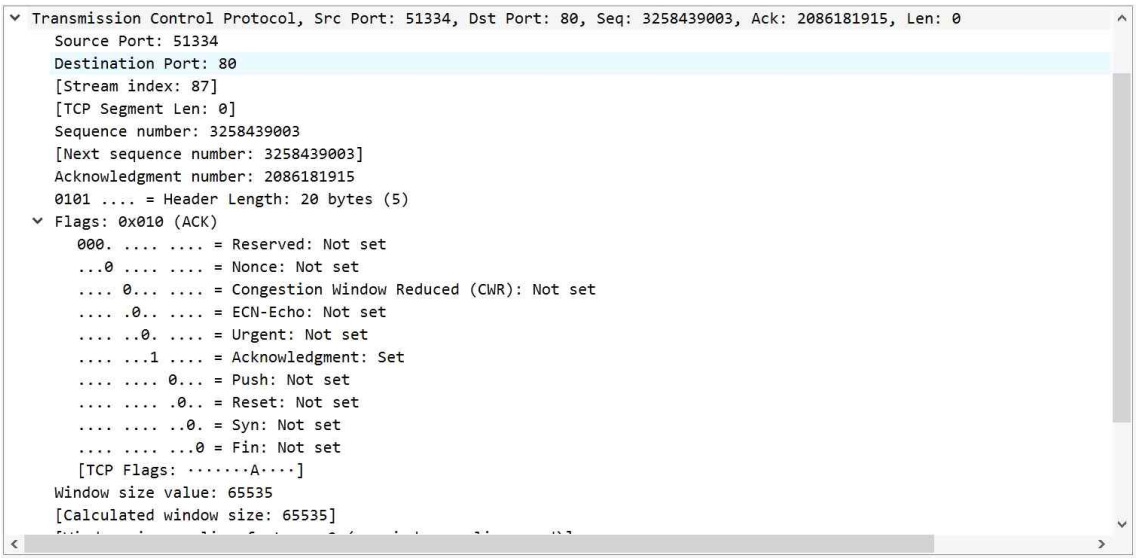
```
Transmission Control Protocol, Src Port: 80, Dst Port: 51334, Seq: 2086181914, Ack: 3258439003, Len: 0
    Source Port: 80
    Destination Port: 51334
    [Stream index: 87]
    [TCP Segment Len: 0]
    Sequence number: 2086181914
    [Next sequence number: 2086181915]
    Acknowledgment number: 3258439003
    0101 .... = Header Length: 20 bytes (5)
  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
      [TCP Flags: ·····A···F]
    Window size value: 5233
```

<그림 2-2-1, connection close fin, ack segment>

위 그림은 4-way handshake의 첫 번째 과정인 server -> client(FIN,ACK)이다. server의 port번호는 source port로 80이며, 나의 port번호는 destination의 port 번호인 51334이다. 또한, FIN, ACK 메시지이기에, flag의 FIN, ACK부분이 1로 되어있음을 확인할 수 있다.

여기서 sequence number는 2086181914이며, next sequence number는 2086181915이며, ack number는 3258439003, 따라서 다음에 올 메시지의 sequence number는

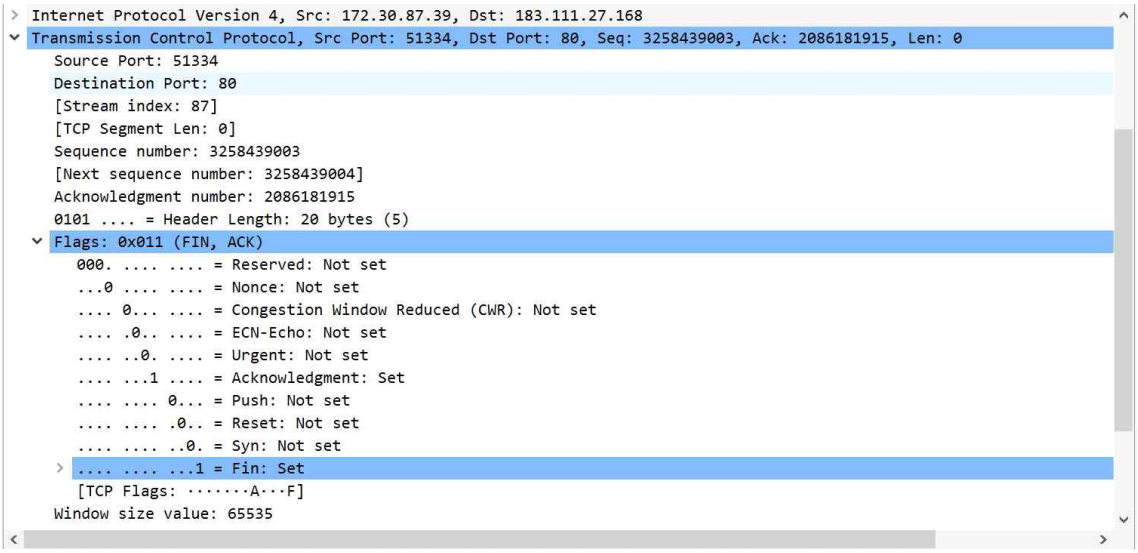
3258439003이며, ack number는 2086181915임을 유추할 수 있다.




<그림 2-2-2, connection close ack segment>

위 그림은 4-way handshake의 두 번째 과정인 client -> server(ACK)이다. 나의 port번호는 source port로 51334이며, server의 port번호는 destination의 port 번호인 80이다. 또한, ACK 메시지이기에, flag의 ACK부분이 1로 되어있음을 확인할 수 있다.

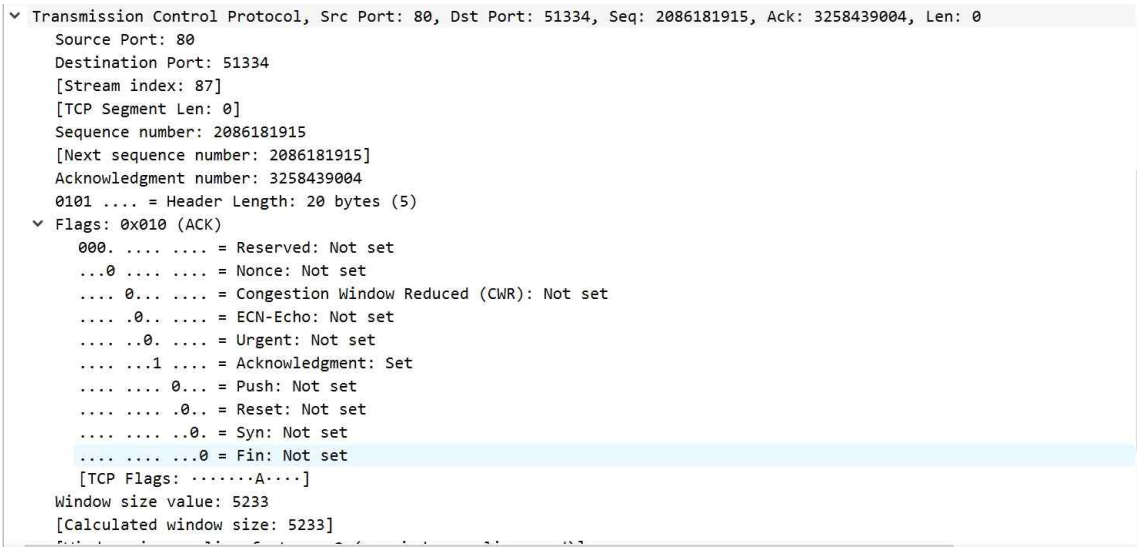
여기서 sequence number는 앞서 예측했듯이 3258439003이며, ack number는 2086181915임을 확인할 수 있다. 이 메시지가 전달된 후 서버는 FIN_WAIT2 상태가 된다.




<그림 2-2-3, connection close fin, ack segment>

위 그림은 4-way handshake의 세 번째 과정인 client -> server(FIN,ACK)이다. 나의 port번호는 source port로 51334이며, server의 port번호는 destination의 port 번호인 80이다. 또한, FIN, ACK 메시지이기에, flag의 FIN, ACK부분이 1로 되어있음을 확인할 수 있다.

여기서 sequence number는 3258439003이며, next sequence number는 3258439004이며, ack number는 2086181915, 따라서 다음에 올 메시지의 sequence number는 2086181915이며, ack number는 3258439003임을 유추할 수 있다.

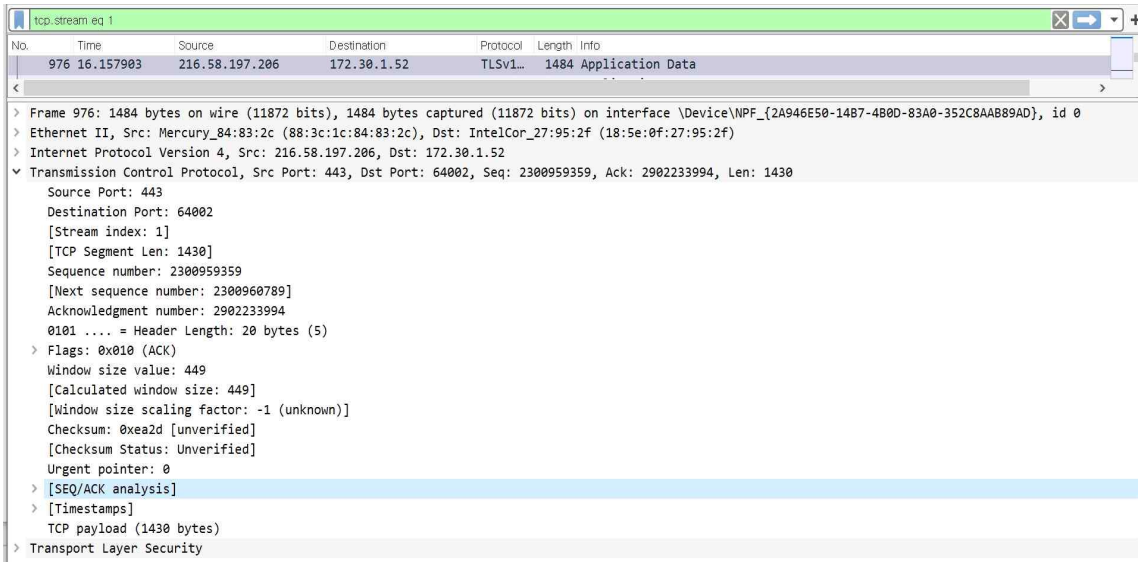



<그림 2-2-4, connection close ack segment>

위 그림은 4-way handshake의 네 번째 과정인 server -> client(ACK)이다. server의 port번호는 source port로 80이며, 나의 port번호는 destination의 port 번호인 51334이다. 또한, ACK 메시지이기에, flag의 ACK부분이 1로 되어있음을 확인할 수 있다.

여기서 sequence number는 앞서 예측했듯이 2086181915이며, ack number는 3258439004이다. 이 메시지를 끝으로 server와 client(나)는 모두 connection close상태가 된다.

- application data를 포함한 TCP segment의 header 분석



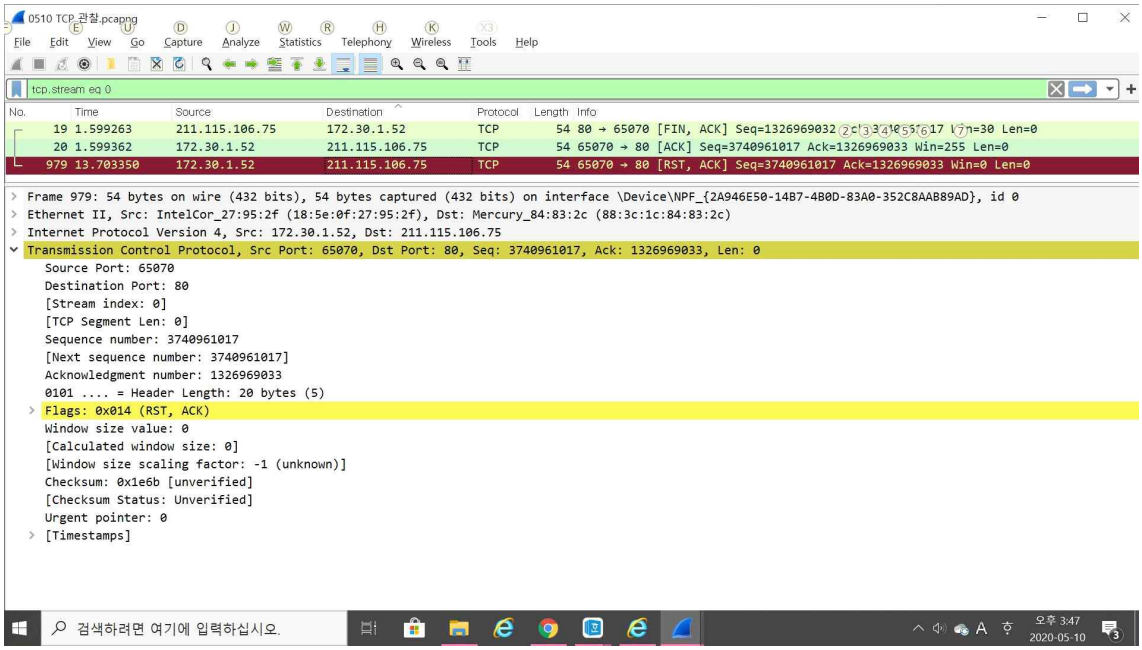

<그림 2-3, application data를 포함한 TCP segment>

위 그림의 첫 번째와 두 번째로 등장하는 헤더 source port와 destination port는 각각 발신포트와 수신포트를 의미한다. 즉, 여기서는 발신 포트가 443, 수신 포트가 64002이다. 더 나아가서 그 위에 적힌 IP주소로부터 유추해보았/을 때, 발신자는 server이며 수신자는 client(나)일 것이다.

그다음에 등장하는 헤더인 TCP segment len과 sequence number, next sequence number를 묶어서 보겠다. 이 셋은 각각 segment의 길이, 순서번호, 그 다음 순서번호를 뜻하며, 위 그림에서 그 값은 각각 1430, 2300959359, 2300960789(2300959359+1430)이다. 수업시간에 배웠던대로 그 다음 순서번호는 현재 사용중인 순서번호에 segment길이를 더한 값이라는 것을 확인할 수 있다.

3) TCP 분석 심화

3-1) connection reset이 발생하는 경우의 TCP segment header 분석




< 그림 3-1, connection reset>

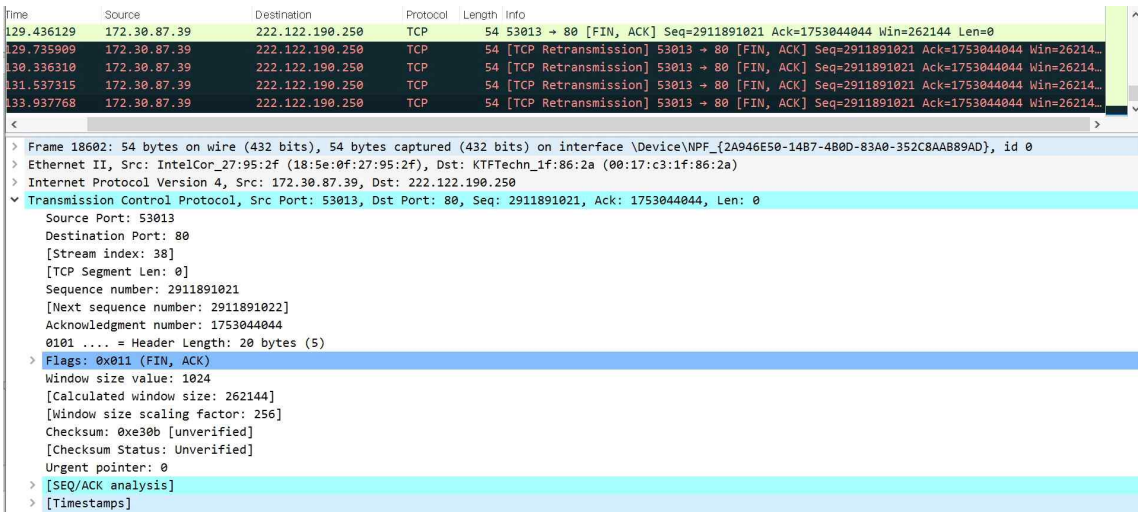
Connection reset은 header에 RST flag를 세팅하여 전송하는 TCP segment이다. <그림 3-1>의 노란줄이 쳐진 부분을 보면 RST flag가 세팅되어있는 것을 확인할 수 있다. 이러한 segment를 수신한다는 것은 디바이스가 연결을 초기화하라는 것을 의미한다.

다른 segment들과 동일하게 RST segment또한 sequence number와 같은 필드를 통하여 데이터 유효성을 확인한다. <그림 3-1>에서의 segment number는 3740961017인 것을 알 수 있다. 이러한 방법을 통해서 이미 연결이 종료된 곳에서 의심스러운 segment가 오는 것을 방지할 수 있다. RST segment가 유효하다고 판단이 되면, 현재 디바이스의 연결 상태에 의거하여 segment를 처리한다. 여러 상황에서 RST segment는 디바이스의 연결을 종료시키고 closed상태가 되게 한다. 이에 따라, 디바이스는 TCP를 사용하고 있는 상위 계층의 프로세스들에게 연결이 종료되었다고 알려준다.

<그림 3-1>에서의 경우는 RST segment위의 segment들이 FIN, ACK과 ACK segment인 것으로 미루어 보았을 때, 가장 일반적으로 RST segment가 보내지는 상황인 half-open connection인 것 같다. half-open connection은 한쪽 디바이스에서 상대방에게 통보없이 연결을 종료시켰을 때 발생한다. 즉, 한쪽은 connection을 established한 상태지만, 다른 device는 closed상태인 것을 의미한다.

<그림 3-1>에서 눈여겨봐야하는 헤더는 flags헤더와 window size 헤더이다. RST segment 답게 flags의 RST부분이 체크되어있는 것을 확인할 수 있으며, 저 부분을 열어보았을 때, RST부분이 1로 되어있을 것이다. 또한 연결이 초기화되었기에 window size헤더부분을 보게 되면 window size 또한 0으로 초기화 된 것을 확인할 수 있다.

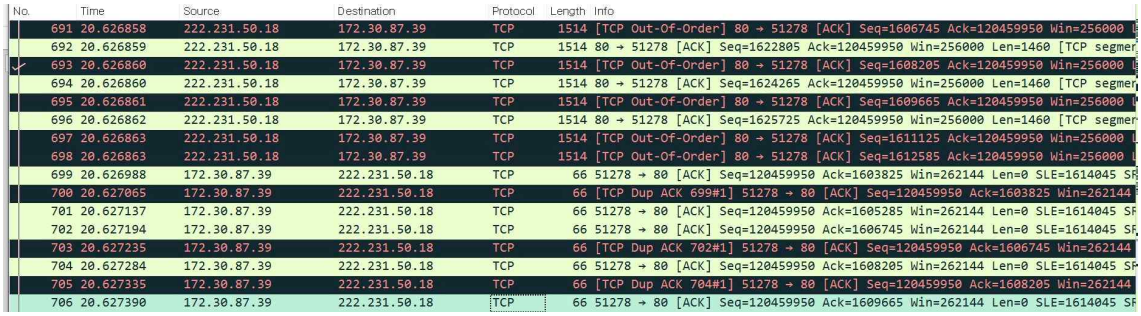
3-2) retransmission이 발생하는 경우의 TCP segment header 분석




<그림 3-2-1, TCP retransmission>

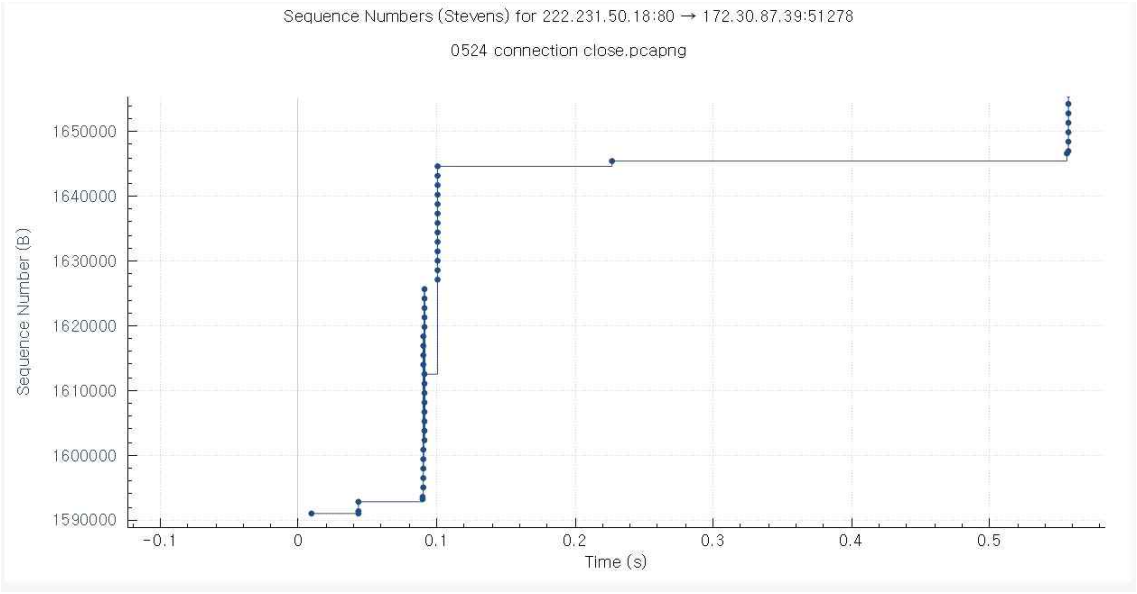
TCP retransmission은 일반적으로 송신자의 timer가 expire할 때 발생한다. 송신자는 packet을 보낼 때, 별도의 timer를 setting해두는데, 해당 timer는 정해진 시간(RTT)내에 수신자로부터 ACK이 오지 않는 경우에 expire하게 된다. 일반적인 retransmission이 발생하는 경우는 다음 3가지로 나뉜다. 첫째 packet lost, packet이 수신자에게 가는 도중 유실되는 경우이다. packet이 도착하지 않았기에 수신자측에서 ACK을 보낼 일은 당연히 없고 이에 따라, timer가 expire하여 retransmission이 발생하게 된다. 둘째 ACK lost, 수신자가 packet을 잘 전달받아서 ACK을 보냈지만, ACK이 송신자에게 가는 도중 유실되는 경우이다. 즉. ACK이 송신자에게 전달되지 못했기에, timer는 expire하게 되고 retransmission이 일어난다. 셋째 Early timeout, 송신자의 packet이 수신자에 잘 전달되었고, 이에따라, 수신자도 ACK을 잘보냈지만, 네트워크 지연이 발생해서 송신자의 time가 expire한뒤 ACK이 도착한 경우이다. 이 경우도 timer가 expire하였기에, retransmission이 발생한다.

<그림 3-2>의 경우를 보면 TCP segment header의 window size value가 1024이다. 하지만 바로위의 잘전달된 packet의 segment header내의 window size는 124이다. 따라서, <그림 3-2>의 retransmission은 급격히 늘어난 window size로 인한 packet lost로 추측해 볼 수 있다. 송신자측에서 한번에 너무 많은 양의 packet을 보내서 loss가 발생했고, 이로인한 retransmission이라고 생각된다.



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 691 | 20.626858 | 222.231.50.18 | 172.30.87.39 | TCP | 1514 | [TCP Out-Of-Order] 80 → 51278 [ACK] Seq=1606745 Ack=120459950 Win=256000 L |
| 692 | 20.626859 | 222.231.50.18 | 172.30.87.39 | TCP | 1514 | 80 → 51278 [ACK] Seq=1622805 Ack=120459950 Win=256000 Len=1460 [TCP segmer |
| 693 | 20.626860 | 222.231.50.18 | 172.30.87.39 | TCP | 1514 | [TCP Out-Of-Order] 80 → 51278 [ACK] Seq=1608205 Ack=120459950 Win=256000 L |
| 694 | 20.626860 | 222.231.50.18 | 172.30.87.39 | TCP | 1514 | 80 → 51278 [ACK] Seq=1624265 Ack=120459950 Win=256000 Len=1460 [TCP segmer |
| 695 | 20.626861 | 222.231.50.18 | 172.30.87.39 | TCP | 1514 | [TCP Out-Of-Order] 80 → 51278 [ACK] Seq=1609665 Ack=120459950 Win=256000 L |
| 696 | 20.626862 | 222.231.50.18 | 172.30.87.39 | TCP | 1514 | 80 → 51278 [ACK] Seq=1625725 Ack=120459950 Win=256000 Len=1460 [TCP segmer |
| 697 | 20.626863 | 222.231.50.18 | 172.30.87.39 | TCP | 1514 | [TCP Out-Of-Order] 80 → 51278 [ACK] Seq=1611125 Ack=120459950 Win=256000 L |
| 698 | 20.626863 | 222.231.50.18 | 172.30.87.39 | TCP | 1514 | [TCP Out-Of-Order] 80 → 51278 [ACK] Seq=1612585 Ack=120459950 Win=256000 L |
| 699 | 20.626988 | 172.30.87.39 | 222.231.50.18 | TCP | 66 | 51278 → 80 [ACK] Seq=120459950 Ack=1603825 Win=262144 Len=0 SLE=1614045 SR |
| 700 | 20.627065 | 172.30.87.39 | 222.231.50.18 | TCP | 66 | [TCP Dup ACK 699#1] 51278 → 80 [ACK] Seq=120459950 Ack=1603825 Win=262144 |
| 701 | 20.627137 | 172.30.87.39 | 222.231.50.18 | TCP | 66 | 51278 → 80 [ACK] Seq=120459950 Ack=1605285 Win=262144 Len=0 SLE=1614045 SR |
| 702 | 20.627194 | 172.30.87.39 | 222.231.50.18 | TCP | 66 | 51278 → 80 [ACK] Seq=120459950 Ack=1606745 Win=262144 Len=0 SLE=1614045 SR |
| 703 | 20.627235 | 172.30.87.39 | 222.231.50.18 | TCP | 66 | [TCP Dup ACK 702#1] 51278 → 80 [ACK] Seq=120459950 Ack=1606745 Win=262144 |
| 704 | 20.627284 | 172.30.87.39 | 222.231.50.18 | TCP | 66 | 51278 → 80 [ACK] Seq=120459950 Ack=1608205 Win=262144 Len=0 SLE=1614045 SR |
| 705 | 20.627335 | 172.30.87.39 | 222.231.50.18 | TCP | 66 | [TCP Dup ACK 704#1] 51278 → 80 [ACK] Seq=120459950 Ack=1608205 Win=262144 |
| 706 | 20.627390 | 172.30.87.39 | 222.231.50.18 | TCP | 66 | 51278 → 80 [ACK] Seq=120459950 Ack=1609665 Win=262144 Len=0 SLE=1614045 SR |

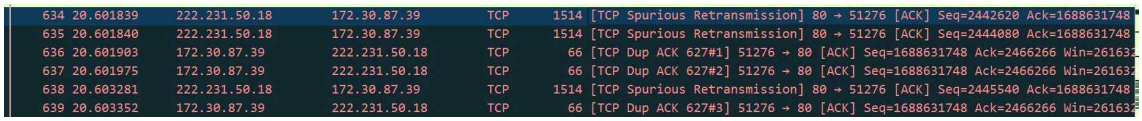
< 그림 3-2-2, duplicated ACK으로 인한 TCP Retransmission>




< 그림 3-2-3, duplicated ACK으로 인한 TCP Retransmission graph>

<그림 3-2-2>는 retransmission이 발생하는 또 다른 예인 duplicated_ack으로 인한 retransmission 사례이다. TCP에서 receiver는 packet이 loss되어서 늦게 오는 것인지, 혹은 packet이 순서가 뒤바뀌어서 늦게 오는 것인지 알 방법이 없다. 그래서 만약, 기다리고 있는 sequence number보다 나중의 sequence number를 가진 packet이 들어오면 receiver는 packet loss로 판단하고 DUP_ACK을 TCP sender에게 보내게 된다. 만약 네트워크에서 순서가 뒤바뀐 경우일수 도 있기에, 나중 packet이 3개 연속해서 들어오기 전에 순서가 뒤바뀐 packet이 올것이라 가정하여 sender가 DUP_ACK을 3번 받을 경우 packet을 retransmit하도록 설정되어 있다. 실제 packet loss라면 DUP_ACK은 retransmission이 일어날 때까지 계속 보내지게 된다.

<그림 3-2-3>는 statistics의 TCP stream에서 Time sequence(stevens)를 들어갔을 때, 관찰할 수 있는 그래프이다. 해당 그래프의 x축 0.1부근을 보면 그래프가 솟아있는데 뒤의 그래프가 앞의 중간부분을 잡아가서 다시 연결하는 것을 볼 수 있다. 이 지점을 클릭해보면 <그림 3-2-2>가 나온다. 즉 해당 부분에서 DUP_ACK들이 보내졌다는 것을 알 수 있다. 중간에 연결된 저 지점의 packet이 오지 않아서 DUP_ACK이 보내졌던 것으로 추측할 수 있다.



| | | | | | |
|---|---|---|---|---|---|
| 634 | 20.601839 | 222.231.50.18 | 172.30.87.39 | TCP | 1514 [TCP Spurious Retransmission] 80 → 51276 [ACK] Seq=2442620 Ack=1688631748 |
| 635 | 20.601840 | 222.231.50.18 | 172.30.87.39 | TCP | 1514 [TCP Spurious Retransmission] 80 → 51276 [ACK] Seq=2444080 Ack=1688631748 |
| 636 | 20.601903 | 172.30.87.39 | 222.231.50.18 | TCP | 66 [TCP Dup ACK 627#1] 51276 → 80 [ACK] Seq=1688631748 Ack=2466266 Win=261631 |
| 637 | 20.601975 | 172.30.87.39 | 222.231.50.18 | TCP | 66 [TCP Dup ACK 627#2] 51276 → 80 [ACK] Seq=1688631748 Ack=2466266 Win=261631 |
| 638 | 20.603281 | 222.231.50.18 | 172.30.87.39 | TCP | 1514 [TCP Spurious Retransmission] 80 → 51276 [ACK] Seq=2445540 Ack=1688631748 |
| 639 | 20.603352 | 172.30.87.39 | 222.231.50.18 | TCP | 66 [TCP Dup ACK 627#3] 51276 → 80 [ACK] Seq=1688631748 Ack=2466266 Win=261631 |

< 그림 3-2-4, TCP spurious retransmission>

spurious retransmission 직역하면, 가짜 재전송이라는 뜻이다. wireshark에서 해당표기는 다음과 같은 경우에 일어난다. sender가 packet을 보냈고, receiver 또한, packet을 잘 받았다. 그렇기에 ACK을 sender에게 다시 보냈다. 그러나 ACK이 loss되어, sender에게 잘 전달되지 못했다. 하지만, sender 입장에서는 자신의 packet이 loss된건지, receiver가 보낸 ACK이 손실된건지 알 방법이 없기에, 다시 packet을 재전송하게 된다. 이러한 재전송을 spurious retransmission이라고 한다. 즉, receiver 입장에서는 필요없는 retransmission인 것이다.

4.새로 알게 된 지식

TCP segment header들에 대해서 보다 심층적으로 공부할 수 있었으며, 수업시간에서 이론으로만 배웠던 내용들이 실제로도 작용되고 있다는 사실이 놀라웠다. UDP를 사용하는 사례에 대해 더욱 자세히 접급하여, UDP를 사용하게 된 이유를 알아보는 과정에서 UDP를 사용하는 다양한 다른 사례들에 대해서도 알아볼 수 있었다. 예를 들어 NBNS, DNS, SSD 등 등에 대해서 보다 더 잘 이해하게 되었다.

또한 TCP connection setup이 일어날 때 사용되는 3-way handshake 또는, TCP connection close가 일어날 때 사용되는 4-way handshake, application data를 포함한 TCP segment들은 비교적 쉽게 관찰할 수 있었으며, 이때 주고받는 segment의 sequence number를 예측해보며, TCP에 대한 이해도를 보다 높일 수 있었다.

TCP connection reset과 TCP retransmission 현상은 화면에 창을 여러개 띄워두거나, 사람이 많은 카페에 갔을 때, 빈번하게 일어났다. 이번 과제를 하면서 특히, retransmission에 대해서 더욱 자세히 이해할 수 있게 되었다. 수업시간에는 배우지 않았던 spurious retransmission이라는 현상을 발견하여 이에 대해, 공부해볼 수도 있었고, DUP_ACK에 대해서 그래프로 보며 공부하니 더욱 잘 이해되었다.