Wireshark 를 이용한 Transport Layer Traffic 분석

수학과 201621136 이 재협

1. 개요

Wireshark 를 이용하여 Transport layer protocol 중 TCP 와 UDP 의 Traffic 을 분석하였다. 약 3 주의 기간 동안, 스터디 카페에서 진행하였다. 아프리카 TV 사이트에 접속하여 관찰되는 UDP header 를 자세하게 분석하였고 해당 Protocol 이 UDP 를 사용하는 이유를 알아보았다. 또한, 마이크로소프트 뉴스 사이트에 접속하여 TCP Connection setup, Connection 종료, 그리고 Application 데이터를 교환하는 과정에서의 TCP segment header 를 분석하였다.

TCP 를 사용하면서 발생하는 Retransmission 과 Reset 이 발생하는 경우에 대해서도 관찰하였다. Retransmission 이 발생했을 때, Sequence number 와 ACK number 를 보며 어떤 데이터가 재전송되었는지 확인할 수 있었다. Reset 이 발생한 경우를 보며, Reset 이 발생하게 된 이유와 flag 의 RST bit 가 설정되는 것을 확인할 수 있었다. 뿐만 아니라, Wireshark 를 사용하며 발견된 새로운 Application layer protocol 중 하나인 ARP 에 대해서 조사하였다. 이를 통해, ARP 의 용도와 헤더 필드들을 사례를 통해 알게 되었다.

2. 관찰 방법

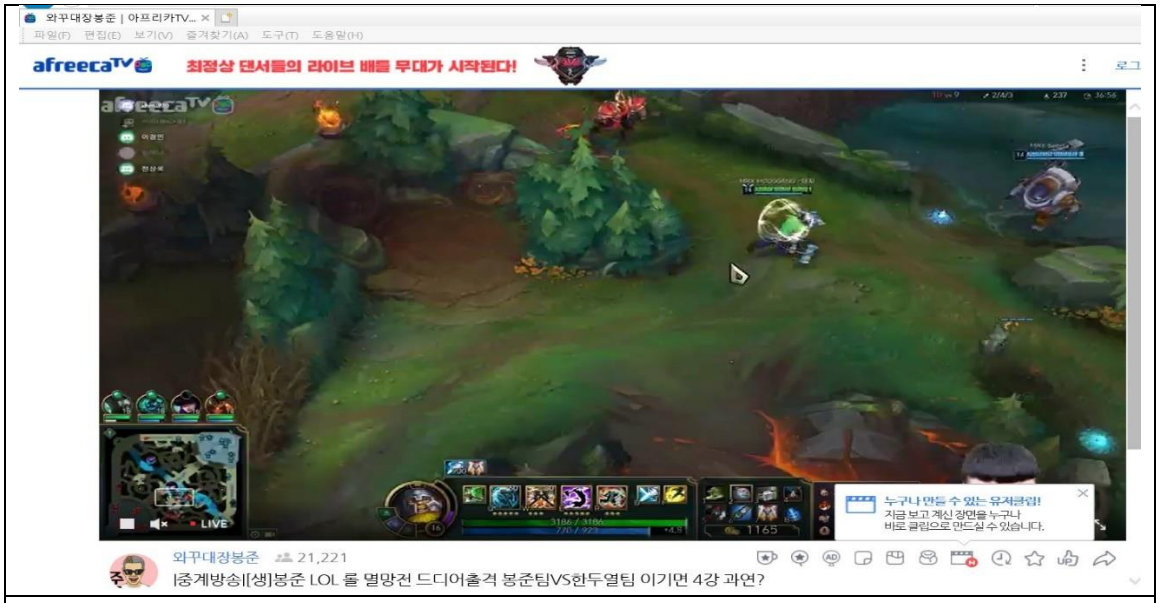
5 월 10 일부터 2~3 일에 한 번씩 관찰을 진행하였다. 아프리카 TV(www.afreecatv.com), 마이크로소프트 뉴스(www.msn.com) 등 다양한 사이트를 관찰하였다. 제출한 파일은 5 월 19 일 이후에 관찰된 자료이다. 장소는 내 방에 있는 iptime 공유기의 WiFi 혹은 내가 다니는 스터디 카페의 Wifi 를 이용하였다. 다른 곳에서 Wireshark 를 사용하게 될 경우 발생할 수 있는 오류를 방지하기 위해서 되도록이면 스터디 카페에서 관찰을 진행하였다.

3. Traffic 분석

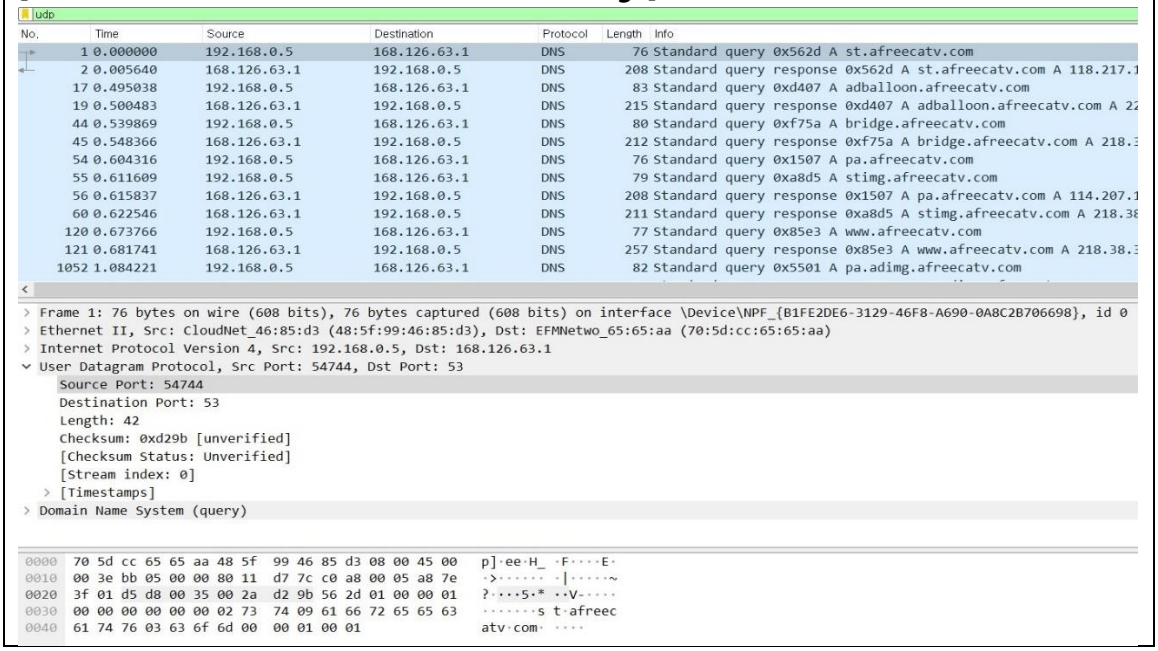
1)UDP 분석

평소 내가 즐겨하던 게임인 피파 온라인 4 와 음악 스트리밍 사이트인 멜론 등 UDP 를 사용할 것으로 추측되는 서비스들을 많이 실험해 보았다. 그러던 중, 인터넷 방송 스트리밍 사이트인 아프리카 TV(www.afreecatv.com)에서 아래 사진과 같은 결과를 얻을 수 있었다.

<아프리카 TV 실제 스트리밍 화면>



[사진 1: 스트리밍 서비스 이용 중 나타난 Message]



Filter 기능을 통해 아프리카 TV 에서 사용하는 DNS 라는 Protocol 은 UDP 를 사용하는 것을 알 수 있었다. **[사진 1]**을 토대로 UDP 에 사용되는 헤더들에 대해 알아보자.

① **Source Port**: 출발지의 포트 번호를 표시한다. 어플리케이션에 따라 포트 번호가 정해져 있는 것도 있지만, 대부분의 경우 처음 Segment 를 전송하는 측에서 임의의 번호를 사용한다.

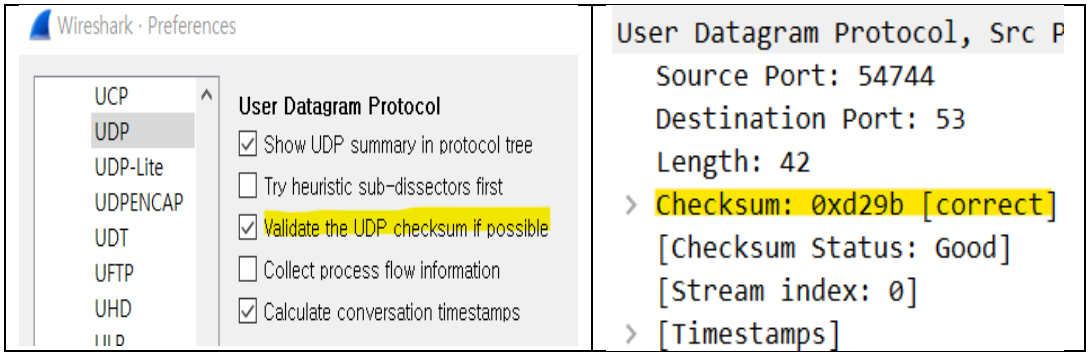
어떤 메시지를 주고받을 때, 상대방의 IP 주소와 포트 번호를 알아야 한다. IP 를 통해 Receiver 의 컴퓨터에 도착된 메시지는 포트 번호를 이용하여 알맞은 어플리케이션에게 전달되기 때문이다. 포트 번호는 0 번~65535 사이의 숫자이며, 어플리케이션에게 할당된

일종의 고유 식별 번호이다. 포트 번호는 IANA(인터넷 할당 번호 관리 기관)에서 관리한다. **[사진 1]**에서 Source Port 는 54744 이므로 동적 할당된 포트 번호임을 알 수 있다.

② **Destination Port:** 목적지의 포트 번호를 표시한다. 53 번은 DNS 에 해당하는 포트 번호이며, **[사진 1]**의 상단을 보면 사용된 Protocol 은 DNS 라는 것도 확인할 수 있다.

③ **Length:** 헤더와 데이터를 포함한 패킷의 바이트 단위이다. 헤더의 크기가 8 이므로, Length 의 값은 최소 8 이상이다.

④ **Checksum:** 패킷이 잘 전송되었는지 확인할 때 Checksum 을 사용한다. Checksum=0 은 Checksum 을 비활성화한 것이며, Receiver 는 Checksum 을 계산하지 않는다. UDP Checksum 의 경우, UDP 헤더의 값과 IP header 의 일부 필드를 사용한다. IPv4 를 사용한다면 Source 와 Destination IP 주소, Length 값 등을 이용하여 IPv4 Pseudo 헤더를 만든다. 이를 토대로 정해진 규칙을 통해 Checksum 을 계산한다.
Wireshark 의 Edit – Preference 기능을 이용하여 UDP, TCP 등의 다양한 protocol 의 Checksum 이 맞는지 확인해볼 수 있다. 아래의 사진은 Checksum 확인 기능을 활성화한 모습이다.



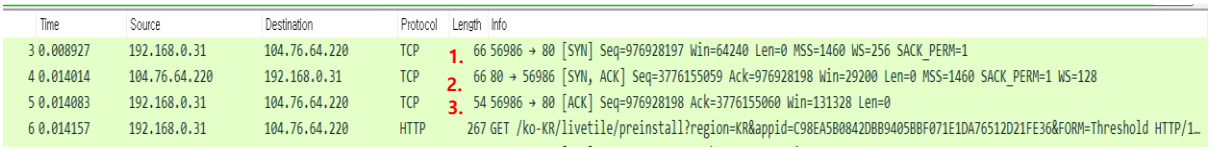
UDP 를 사용하는 이유에 대해 알아보자. 먼저, 아프리카 TV 는 BJ 의 모습 혹은 컴퓨터 화면 등을 시청자들에게 '실시간으로' 제공하는 서비스이다. 만약, 아프리카 TV 가 TCP 를 사용한다고 가정해 보자. TCP 는 영상을 전송하던 도중 손실이 발생하면 해당 데이터가 제대로 도착할 때까지 대기하는 경우가 생길 수 있다. 이 때, 영상의 품질에 크게 영향을 끼치지 않을 정도의 작은 손실이라도 영상이 멈추는 상황이 발생할 수 있다. 그러므로 아프리카 TV 는 데이터의 정확성보다는 빠른 전송 속도에 비중을 둔다. 즉, 약간의 데이터 손실을 감수하면서, Latency 를 줄이는 방법이 서비스에 더 적합하다는 것이다. 이로 인해, 아프리카 TV 는 UDP 를 사용한다.

2) TCP 분석

TCP 분석에는 msn 사이트의 날씨 사이트(http://tile-service.weather.microsoft.com)를 이용하였다. Statistics-Conversations-Follow Stream 기능을 통해 해당 사이트와 주고받은 패킷을 쉽게 필터링할 수 있었다. 또한, Protocol preference – Relative sequence number 옵션을 해제하였다. **i**, **ii** 에서는 각 단계에 중요한 역할을 하는 몇몇 헤더들과 Option 헤더에 대해서만 언급하고 다른 헤더 필드의 상세한 분석은 iii 에서 다룰 예정이다.

i. Connection setup 단계에서 교환하는 TCP segment header 분석

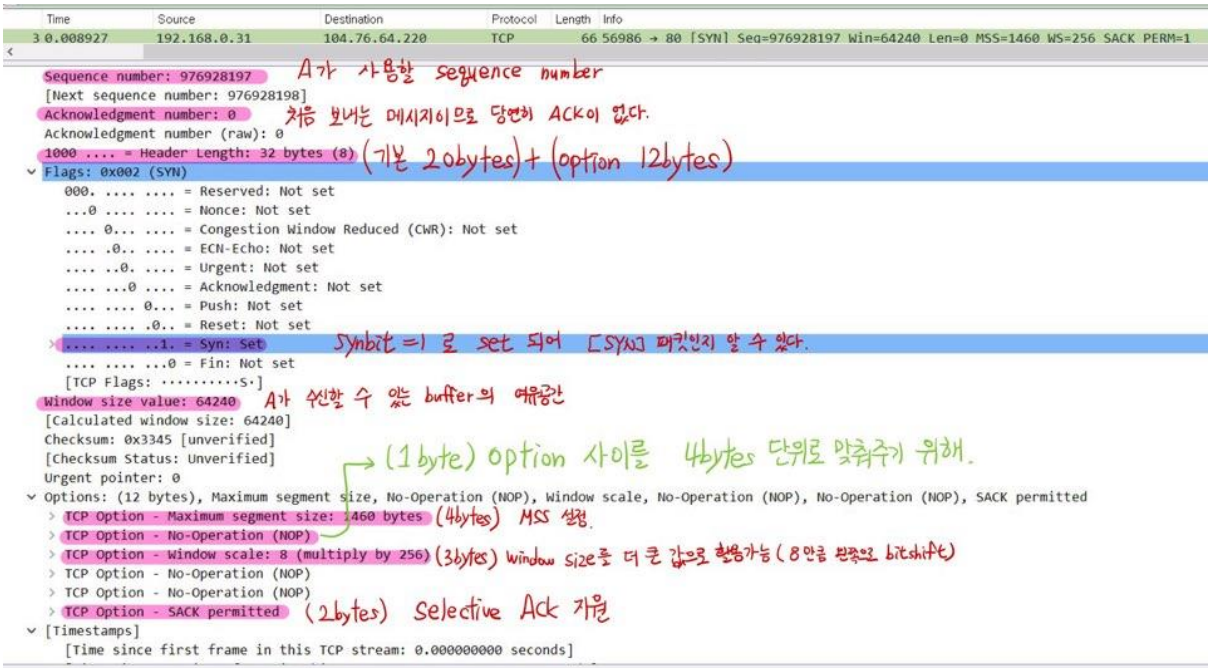
[사진 2: Connection setup 단계]



	Time	Source	Destination	Protocol	Length	Info
3	0.008927	192.168.0.31	104.76.64.220	TCP	1. 66	56986 → 80 [SYN] Seq=976928197 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
4	0.014014	104.76.64.220	192.168.0.31	TCP	2. 66	80 → 56986 [SYN, ACK] Seq=3776155059 Ack=976928198 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
5	0.014083	192.168.0.31	104.76.64.220	TCP	3. 54	56986 → 80 [ACK] Seq=976928198 Ack=3776155060 Win=131328 Len=0
6	0.014157	192.168.0.31	104.76.64.220	HTTP	267	GET /ko-KR/livetile/preinstall?region=KR&appid=C98EA5B0842DBB9405BBF071E1DA76512D21FE36&FORM=Threshold HTTP/1…

가장 위에 있는 3 개의 패킷이 Setup 단계에서 발생한 것이다.
192.168.0.75 는 내 컴퓨터(이하 **A**)의 IP 이고, 104.76.64.220 는 내가 접속한 날씨 사이트의 서버(이하 **B**) IP 이다.
먼저, 3-way-handshaking 과정은 **[사진 2]**의 1~3 의 단계로 이루어지며, 이를 각각 **SYN**, **SYN/ACK**, **ACK** 이라고 한다. 이 단계를 통해 TCP Connection 이 성사되며, Setup 단계에서 Sequence number, Window size, MSS, SACK 등을 Negotiation 한다.

[사진 3: SYN 단계]



- **Sequence number**: 데이터를 보낼 때 마다 각 데이터에 부여하는 고유한 번호이며, 이미 종료된 연결에서 사용하던 값과의 중복을 피하기 위해 임의로 부여한다. A 가 처음으로

사용하는 Sequence number 를 A 의 ISN(Initial Sequence Number)이라고 한다. 데이터를 한 번 전송할 때마다 이 번호가 1 씩 증가한다. A 의 Sequence number 를 기준으로 B 가 ACK 를 전송하고, 이를 통해 A 의 데이터들이 B 에게 잘 도착했는지 확인한다. 위의 사진의 경우, A 는 ISN 을 976928197 으로 설정하여 B 에게 패킷을 보냈다.

- **Flag: SYN** 단계에서는 Flag 중 Syn bit 가 1 로 설정된다. Syn flag 를 통해서, [SYN]단계에서 보낸 패킷이라는 것을 B 가 알 수 있다.

- **Window size value**: 수신 buffer 의 여유 용량을 말하며 Receiver window 혹은 rwnd 라고 표현하기도 한다. A 가 이후에 메시지를 받을 때 ACK 없이 한 번에 데이터를 받을 수 있는 크기를 말한다. B 는 이 rwnd 를 통해 A 에게 한꺼번에 보낼 데이터의 양을 결정한다. 이 헤더의 크기는 2bytes 이므로 표현할 수 있는 Window size 의 최댓값은 65,535 이지만, Option 의 Window scale 에 따라 더 늘어날 수 있다.

-Options

Maximum segment size: IP 헤더, TCP 헤더를 제외하고 TCP 가 담을 수 있는 payload 데이터의 최대 크기이다. MTU(Maximum Transmission Unit)는 IP datagram 의 최대 크기이며 Ethernet 환경의 경우 MTU 의 기본값은 1500 이다. 따라서, MSS = MTU – (IP 헤더의 크기) – (TCP 헤더의 크기) 이다. **[사진 3]**의 경우 Ethernet 을 사용했고, IP 헤더와 TCP 헤더의 크기는 20Byte 이상이므로 MSS 의 최댓값은 1460bytes 가 된다.

Window scale: 2bytes 로 표현되는 Window size value 의 값을 더 크게 만들기 위해 사용된다. 0~14 의 값이 들어가며, 해당 값만큼 왼쪽으로 bit shift 를 하여 65,535 보다 훨씬 큰 값의 window size 를 사용할 수 있다. A 와 B 모두 Window scale 옵션을 활성화하는 경우에만 확장된 크기의 Window size 를 사용할 수 있다.

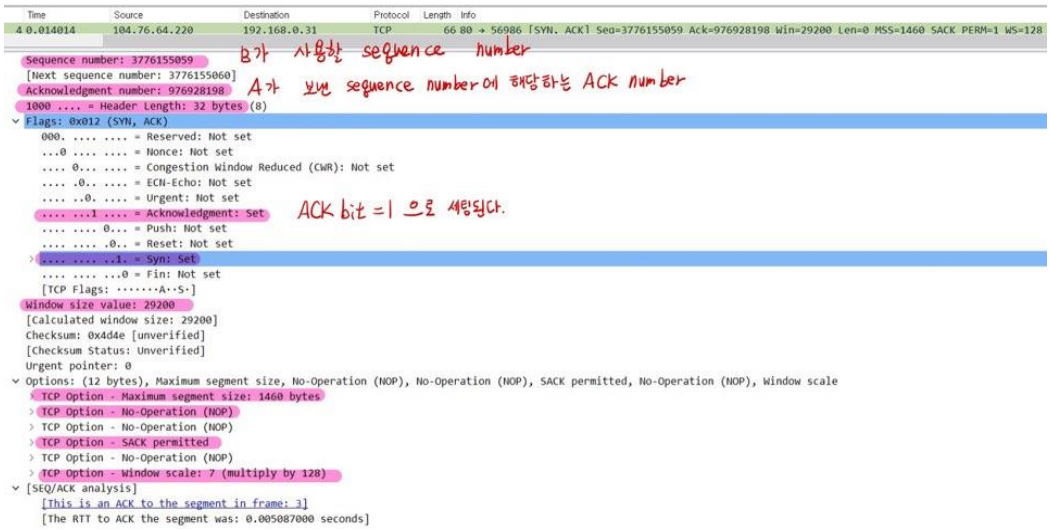
SACK permitted: SACK 이란 Selective Acknowledgement 의 약자이며, 손실된 데이터만 다시 보내도록 해주는 전송 방식이다. 서버가 Sequence number #1~#4 의 segment 를 보냈지만 #2 segment 가 loss 된 경우를 생각해보자.

일반적인 TCP 전송	클라이언트는 #1 ACK 만 보낸다. 이를 받은 서버는 #2~#4 segment 를 모두 재전송한다.
SACK	클라이언트는 SLE 와 SRE 를 이용하여 #1 ACK 뿐만 아니라 #3~#4 segment 도 잘 받았다고 알려준다. 이를 받은 서버는 #2 만 재전송한다.

이와 같은 방법으로 SACK 은 불필요하게 재전송되는 segment 를 줄일 수 있다. 두 호스트 모두가 SACK permitted 가 지원되는 경우에 SACK 방식으로 데이터가 전송된다.

No-Operation: Option 들이 4bytes 의 단위로 구분될 수 있도록 만들기 위해 사용된다. 크기는 1byte 이며 아래에 사용되는 Window scale 옵션과 SACK permitted 옵션이 각각 3bytes, 1byte 이므로 No-Operation 옵션이 사용되었다.

[사진 4: SYN/ACK]

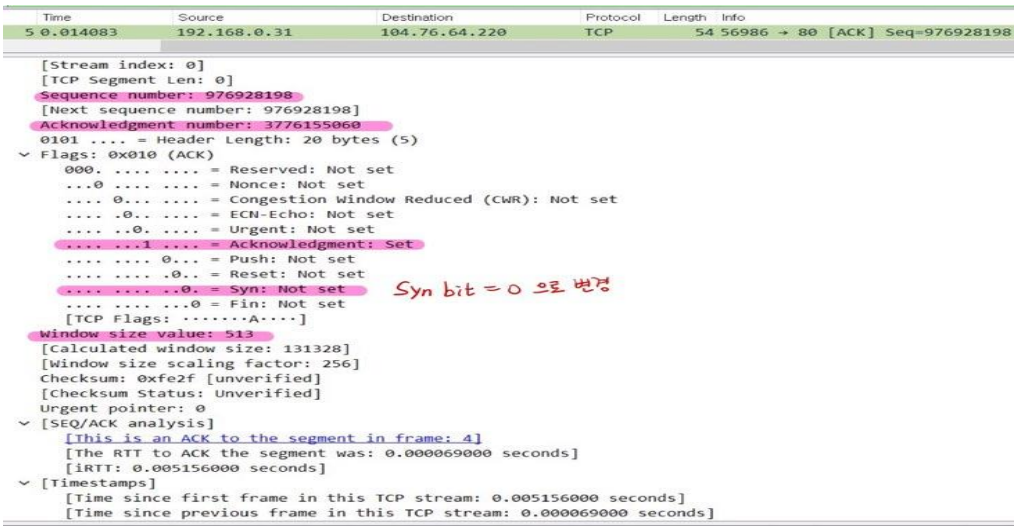


- Sequence number: 이번에는 B 가 자신이 보내는 패킷에 Sequence number 를 부여했다. B 는 ISN 을 3776155059 으로 설정하여 A 에게 보냈고 이를 바탕으로 A 는 ACK 을 보낼 것이다.

- Acknowledgment number: Acknowledgment number(이하 ACK)란, 상대방이 보낸 패킷을 잘 받았다는 표시이며, 상대방의 패킷에 부여된 Sequence number 를 기준으로 작성된다. 976928198 ACK 는 A 가 보냈던 976928197 까지의 데이터를 잘 받았다는 뜻이다. **- Flag:** Acknowledgment 와 Syn bit 가 1 이 되었음을 확인할 수 있다.

- Window size value 와 Window scale 이 다른 것 외에는 SYN segment 와 유사하였다.

[사진 5: ACK]



- **Sequence number:** 직전에 A 는 B 로부터 976928198 ACK 을 받았다. A 는 976928197 Seq 패킷이 잘 도착했다는 것을 알게 되었다. 따라서, A 는 다음 숫자인 976928198 을 Sequence number 로 설정하여 패킷을 보낸다.

- **Acknowledgment number:** 직전에 A 는 B 로부터 #3776155059 패킷을 받았다. 따라서, Acknowledgment number 를 #3776155060 으로 설정하여 B 의 패킷을 잘 받았음을 알려준다.

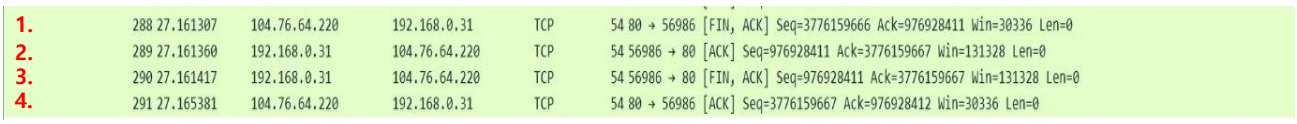
- **Flag:** Ack bit 가 1 인 상태로 유지된다. Setup 단계 이후에도 ACK 을 사용하는 동안에는, 해당 flag 는 변하지 않았다.

- **Window size value:** 513 으로 되어 있지만 바로 아래 Calculated window size 는 131,328 으로 되어있다. 그 이유는, **[SYN]**의 Option 중 Window scale 이 8 이므로 513*2^(8) = 131,328 으로 계산된 것이다.

ii. Connection 종료 단계에서 교환하는 TCP segment header 분석

서로 전송할 데이터가 없고 Connection 을 계속 유지할 필요가 없는 경우 Connection 종료가 이루어 진다. Connection 종료 단계는 아래와 같이 1~4 의 단계로 이루어진다.

[사진 6: Connection 종료 단계]



1. Source Port: 80 Destination Port: 56986 [Stream index: 0] [TCP Segment Len: 0] Sequence number: 3776159666 [Next sequence number: 3776159667] Acknowledgment number: 976928411 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	방향: B→A 내용: [FIN, ACK]을 통해 연결을 종료할 것을 알린다. Flag 의 Fin bit 가 1 인 것을 통해 A 는 TCP 연결이 종료될 것을 알게 된다.
2. Source Port: 56986 Destination Port: 80 [Stream index: 0] [TCP Segment Len: 0] Sequence number: 976928411 [Next sequence number: 976928411] Acknowledgment number: 3776159667 0101 .... = Header Length: 20 bytes (5) Flags: 0x010 (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 ....0 = Fin: Not set [TCP Flags: ·····A····]	방향: A→B 내용: B 의 [FIN, ACK] 패킷을 잘 받았다는 뜻으로 [ACK]을 보낸다. A 는 TCP 연결을 종료할 준비를 하기 시작한다. A 가 전송할 데이터가 남아 있다면, [ACK]을 보낸 이후에도 계속해서 데이터를 보낸다.
3. Source Port: 56986 Destination Port: 80 [Stream index: 0] [TCP Segment Len: 0] Sequence number: 976928411 [Next sequence number: 976928412] Acknowledgment number: 3776159667 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]	방향: A→B 내용: A 가 보내야 할 데이터들을 다 보냈고 TCP 연결을 종료할 준비를 모두 마쳤으면, A 가 B 에게 [FIN, ACK]을 보낸다. Fin bit 가 1 이 되었음을 확인할 수 있다.
4. Source Port: 80 Destination Port: 56986 [Stream index: 0] [TCP Segment Len: 0] Sequence number: 3776159667 [Next sequence number: 3776159667] Acknowledgment number: 976928412 0101 .... = Header Length: 20 bytes (5) Flags: 0x010 (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 ....0 = Fin: Not set [TCP Flags: ·····A····]	방향: B→A 내용: A 의 [FIN, ACK]을 잘 받았다는 [ACK]을 보낸다. A 는 B 의 [ACK]을 받은 후, TCP 연결을 종료한다. B 는 자신이 [ACK]을 보내고 MSL(Maximum Segment Lifetime) X 2 만큼의 시간이 지난 후에 TCP 연결을 종료한다.

B 가 2MSL 의 시간동안 대기하는 이유는 무엇일까? A 는 B 의 [ACK]을 수신해야만 완전한 종료가 가능하다. 이 때, B 가 보낸 [ACK]이 손실된다면 A 는 TCP 연결을 종료하지 못하고 기다려야 한다. 따라서, A 는 B 의 [ACK]이 돌아올 때까지 [FIN, ACK]을 재전송한다. 다시 말해, B 의 [ACK]이 손실되어 A 가 [FIN, ACK]을 재전송하는 경우를 대비하여 B 는 2MSL 동안 대기한 후 연결을 종료하는 것이다.

iii. Application 데이터를 포함한 TCP segment 의 header 분석

[사진 7: Application 데이터를 포함한 TCP segment]

3	0.008927	192.168.0.31	104.76.64.220	TCP	66 56986 → 80 [SYN] Seq=976928197 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
4	0.014014	104.76.64.220	192.168.0.31	TCP	66 80 → 56986 [SYN, ACK] Seq=3776155059 Ack=976928198 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
5	0.014083	192.168.0.31	104.76.64.220	TCP	54 56986 → 80 [ACK] Seq=976928198 Ack=3776155060 Win=131328 Len=0
6	0.014157	192.168.0.31	104.76.64.220	HTTP	267 GET /ko-KR/livetile/preinstall?region=KR&appid=C98EA5B0842DBB9405BBF071E1DA76512D21FE36&FORM=Threshold HTTP/1…
7	0.019051	104.76.64.220	192.168.0.31	TCP	54 80 → 56986 [ACK] Seq=3776155060 Ack=976928411 Win=30336 Len=0
8	0.023048	104.76.64.220	192.168.0.31	TCP	1514 80 → 56986 [ACK] Seq=3776155060 Ack=976928411 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
9	0.023050	104.76.64.220	192.168.0.31	TCP	1514 80 → 56986 [ACK] Seq=3776156520 Ack=976928411 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
10	0.023106	192.168.0.31	104.76.64.220	TCP	54 56986 → 80 [ACK] Seq=976928411 Ack=3776157980 Win=131328 Len=0
11	0.023158	104.76.64.220	192.168.0.31	TCP	1514 80 → 56986 [ACK] Seq=3776157980 Ack=976928411 Win=30336 Len=1460 [TCP segment of a reassembled PDU]
12	0.023159	104.76.64.220	192.168.0.31	HTTP/X…	280 HTTP/1.1 200 OK
13	0.023181	192.168.0.31	104.76.64.220	TCP	54 56986 → 80 [ACK] Seq=976928411 Ack=3776159666 Win=131328 Len=0

6 번 Segment 부터, Application 데이터를 포함한 TCP segment 임을 확인할 수 있었다. 먼저, 6 번 패킷을 이용하여 헤더 아직 언급되지 않은 필드들에 대해 알아보자.

[사진 8: TCP segment 의 헤더 필드]

tcp.stream eq 0

Time	Source	Destination	Protocol	Length	Info
6 0.014157	192.168.0.31	104.76.64.220	HTTP	267	GET /ko-KR/livetile/preins

```
Transmission Control Protocol, Src Port: 56986, Dst Port: 80, Seq: 976928198, Ack: 3776155060, Len: 213
    Source Port: 56986
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 213]
    Sequence number: 976928198
    [Next sequence number: 976928411]
    Acknowledgment number: 3776155060
    0101 .... = Header Length: 20 bytes (5)
  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···]
    Window size value: 513
    [Calculated window size: 131328]
    [Window size scaling factor: 256]
    Checksum: 0x0800 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  [SEQ/ACK analysis]
      [iRTT: 0.005156000 seconds]
      [Bytes in flight: 213]
      [Bytes sent since last PSH flag: 213]
  [Timestamps]
    TCP payload (213 bytes)
```

① **Header length:** TCP 헤더의 전체 길이를 byte 단위로 표현해준다. Option 이 없는 경우 기본적으로 20bytes 이다. **[사진 3]**을 보면, 기존의 20bytes 에서 Option 에서의 12bytes 가 더해져서 32bytes 임을 볼 수 있다.

② **Flags ⇒** 이진수로 000000011000 이고 0x018 로 표현될 수 있다.

⑴ Reserved: 미래에 사용하기 위해 남겨둔 예비 필드이며 0 으로 채워진다.

⑵ NS(Nonce): Sender 의 패킷이 악의적인 의도로 은폐되는 것을 방지하기 위해 존재하며, 아직은 실험 중에 있는 flag 이다

⑶ ECN-Echo (ECE): SYN flag 가 1 인 경우, 즉 **[SYN]**과정 중에는 ECN(Explicit Congestion Notification)이 가능함을 의미한다. SYN flag 가 0 인 경우에는, IP 헤더 셋에 Congestion Experienced flag 가 설정된 패킷이 정상적인 전송 중에 수신되었다는 것을 의미한다.

⑷ Congestion Window Reduced (CWR): 송신 측 호스트에 의해 설정되는 것으로, 호스트가 ECE flag 가 포함된 TCP segment 를 수신했으며, Congestion Control 에 응답했음을 알려준다.

⑸ Urgent (URG): 전송하는 데이터 중에서 긴급히 전달하는 내용이 있을 때 활성화된다. 이 때, 다른 데이터에 비해서 높은 우선 순위를 가진다.

⑹ Acknowledgment (ACK): 상대방이 보낸 패킷을 잘 받아서 Acknowledgment Number 로 알려줄 때 사용한다.

⑺ Push (PSH): 받은 데이터를 Application Layer 로 곧바로 전송하도록 하는 Flag 이다. 상호작용이 중요한 Protocol 의 경우 빠른 응답이 필요하므로, Buffer 가 채워질 때까지 기다리지 않고 데이터를 전달한다. 위의 사진의 경우, PSH flag 가 활성화되어 있다. 따라서, 전송되는 데이터는 바로 Application Layer 로 옮겨지므로 Buffer 의 여유 데이터는 줄어들지 않는다. 이로 인해 Window size 가 변하지 않는 것을 알 수 있다.

⑻ Reset (RST): RST flag 가 활성화되면 세션을 끊는 것을 의미한다. 즉, 비정상적인 연결 종료가 일어났을 때 사용한다.

⑼ SYN: Synchronization 의 뜻을 가지고 있으며, TCP Setup 단계에서 가장 먼저 사용된다.

⑽ FIN: Finish 의 뜻을 가지고 있다. 더 이상 전송할 데이터가 없어 세션 연결을 정상적으로 종료할 때 사용된다.

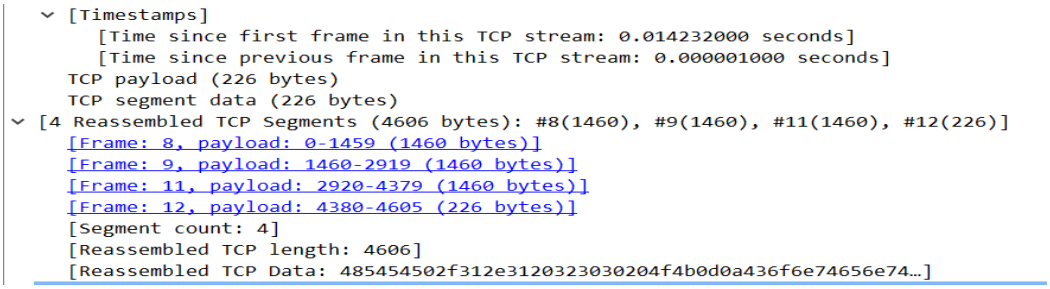
③ **Urgent pointer:** URG flag 가 설정된 경우, Sequence number 로부터의 Offset 을 나타낸다. 이 Offset 이 긴급한 데이터의 마지막 바이트를 가리킨다.

④ **Checksum:** UDP 에서의 Checksum 과 동일하다.

이번에는, 패킷을 교환하며 나타나는 Sequence number 와 Ack number 에 대해 알아보자. #6 에서 A 는 B 에게 데이터를 요청하였다. 이 때, Seq=976928198, Ack=3776155060 이며 Len=213 이다. #6 의 ACK=3776155060 이므로 #7 의 Seq = 3776155060 이 된다. #7 의 Ack 은 (#6 의 Seq + #6 의 Len) = 976928411 이다. 헤더를 제외한 다른 데이터는 담기지 않았으므로 Len=0 이다. #8 은 Seq=3776155060 부터 전송하기 시작했으며, #7 에서 Len=0 이므로 Seq 은 증가하지 않았다. 새로 도착한 A 의 데이터가 없으므로, Ack 은 유지된다. #8 의 Len 은 TCP 가 담을 수 있는 가장 많은 양의 데이터인 1460 이 된다. 이는 데이터가 4 번에 걸쳐서 전송되기 때문이며 자세한 설명은 [사진 8]을 통해 알 수 있다. #9 에서도 #8 과 같은 방식으로 Seq, Ack, Len 을 설정하였다. #10 에서 A 가 #9 까지의 데이터를 빠짐없이 잘 받았다고 ACK 을 보냈다. #10 의 ACK 은 3776156520 + 1460 = 3776157980 이 된다. #11~#12 에서도 쪼개진 데이터가 계속해서 A 에게 전달되고 있다. #13 은 A 가 B 에게 Ack 을 보내는 과정이다. #13 Ack = (#12 Seq + #12 Len) = 3776159666 이다. Ack 을 보내는 것이 목적이고 다른 데이터는 없으므로, Len=0 이다.

[TCP segment of a reassembled PDU]의 뜻에 대해 알아보자.
Application 이 많은 양의 데이터를 전송하려 해도, 하나의 TCP segment 가 담을 수 있는 데이터의 최댓값(MSS)은 한계가 있다. 따라서, 여러 개의 TCP segment 에 Application 데이터를 나누어 전송한다. 상대방은 나누어 전송된 TCP segment 들을 적절히 Reassemble 하여 원래의 데이터로 복원한다. 위의 사진의 경우, B 가 보내려고 하는 데이터는 4,606bytes 이다. MSS=1,460bytes 이므로, #8(1,460 bytes)+ #9(1,460 bytes)+ #11(1,460 bytes)+ #12(226 bytes)으로 나누어 보내게 된 것이다.

[사진 8: Reassembled TCP Segments]

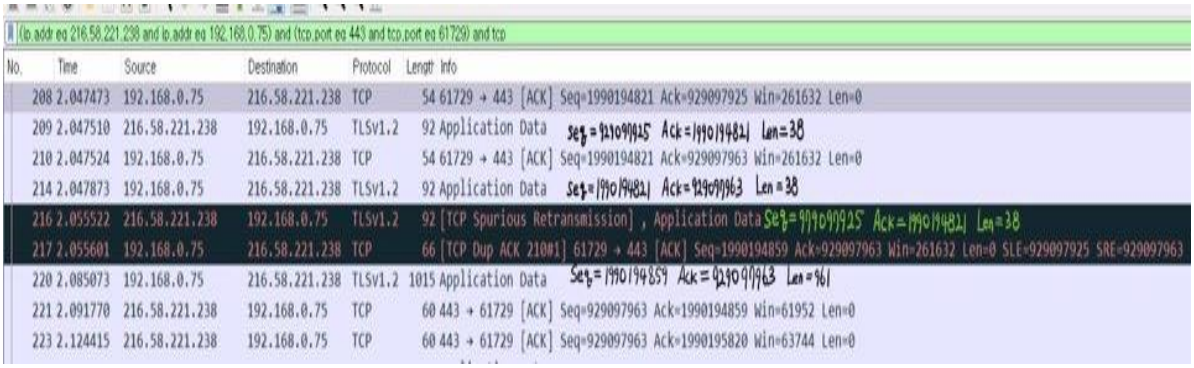


[사진 8]을 통해, 총 4,606bytes 의 데이터가 #8, #9, #11, #12 로 4 개의 Segment 에 걸쳐서 전송되었다는 것을 알 수 있다. 하나의 TCP segment 가 담을 수 있는 데이터의 양은 1460bytes 이므로 #8~#11 은 len=1460, #12 의 len=226 이 된다.

3) TCP 분석 심화

i. Retransmission

[사진 10: Retransmission]



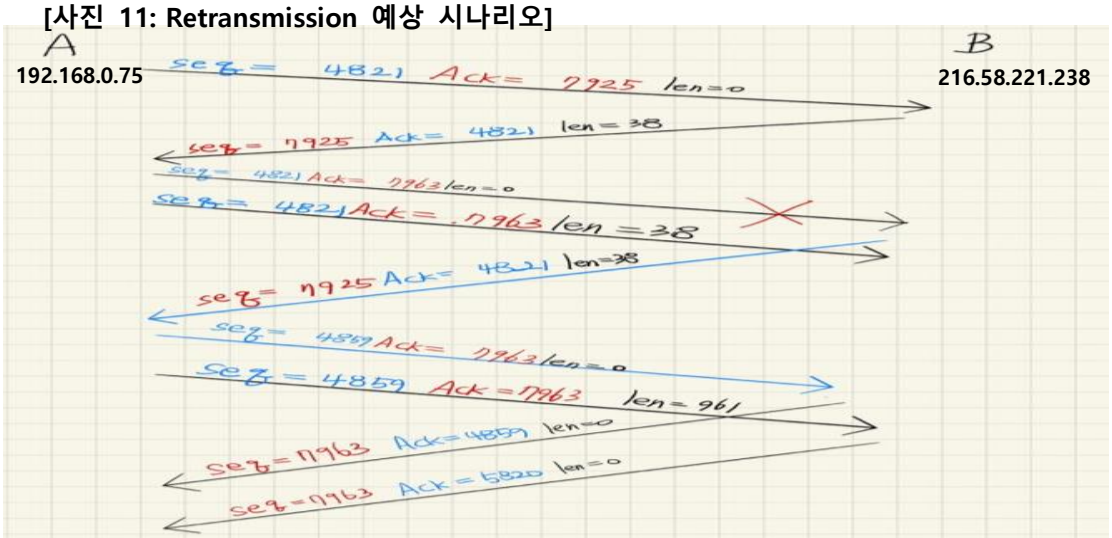
No.	Time	Source	Destination	Protocol	Length	Info
208	2.047473	192.168.0.75	216.58.221.238	TCP	54	61729 → 443 [ACK] Seq=1990194821 Ack=929097925 Win=261632 Len=0
209	2.047510	216.58.221.238	192.168.0.75	TLSv1.2	92	Application Data
210	2.047524	192.168.0.75	216.58.221.238	TCP	54	61729 → 443 [ACK] Seq=1990194821 Ack=929097963 Win=261632 Len=0
214	2.047873	192.168.0.75	216.58.221.238	TLSv1.2	92	Application Data
216	2.055522	216.58.221.238	192.168.0.75	TLSv1.2	92	[TCP Spurious Retransmission] , Application Data
217	2.055601	192.168.0.75	216.58.221.238	TCP	66	[TCP Dup ACK 210#1] 61729 → 443 [ACK] Seq=1990194859 Ack=929097963 Win=261632 Len=0 SLE=929097925 SRE=929097963
220	2.085073	192.168.0.75	216.58.221.238	TLSv1.2	1015	Application Data
221	2.091770	216.58.221.238	192.168.0.75	TCP	60	443 → 61729 [ACK] Seq=929097963 Ack=1990194859 Win=61952 Len=0
223	2.124415	216.58.221.238	192.168.0.75	TCP	60	443 → 61729 [ACK] Seq=929097963 Ack=1990195820 Win=63744 Len=0

TCP 를 통해 주고받는 패킷들을 분석하던 중 위의 사진과 같은 경우를 볼 수 있었다. #210 을 통해 A 는 #209 를 잘 받았다는 ACK 을 보냈으나 B 에게 잘 전달되지 못하였다. 따라서, B 는 #216 을 통해 #209 와 같은 패킷을 전송하였다. 한편, A 의 입장에서는 #209 에 대한 ACK 을 보냈으나, #209 와 동일한 데이터가 담긴 #216 을 받게 되어 [TCP Spurious retransmission]이 나타났다. B 가 #210 을 받지 못한 것으로 생각한 A 는 #212 를 통해 한 번 더 ACK 을 보낸다. 이 때, A 는 B 에게 전달해줄 데이터를 함께 실어서 보낸다. 따라서, Sequence number 는 1990194821 이 아닌 1990194859 가 되는 것이다.

사실, Connection 을 하면서 상대방이 어떤 상황인지 자신은 알지 못한다. 즉, A 는 B 가 어떤 상황에 놓여있는지 알지 못하고, Timeout 이나 B 에게서 전달되는 Duplicated ACK 등을 통해 예상을 할 뿐이다. 따라서, Spurious retransmission 이 발생했을 때, B 가 #210 을 받지 못한 것으로 예상했을 뿐이다.

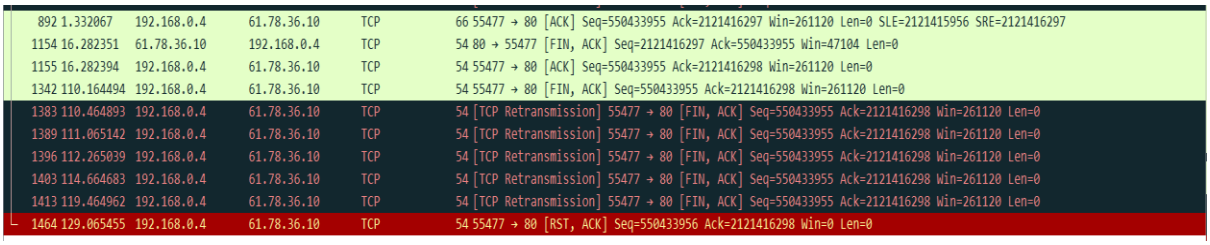
[사진 10]를 통해 예측한 A 와 B 의 통신 과정은 아래와 같다. Sequence number 와 Ack number 는 간단하게 표현하기 위해 가장 뒤의 4 자리만 표시하였다.

[사진 11: Retransmission 예상 시나리오]



ii. Reset

[사진 12: Reset]



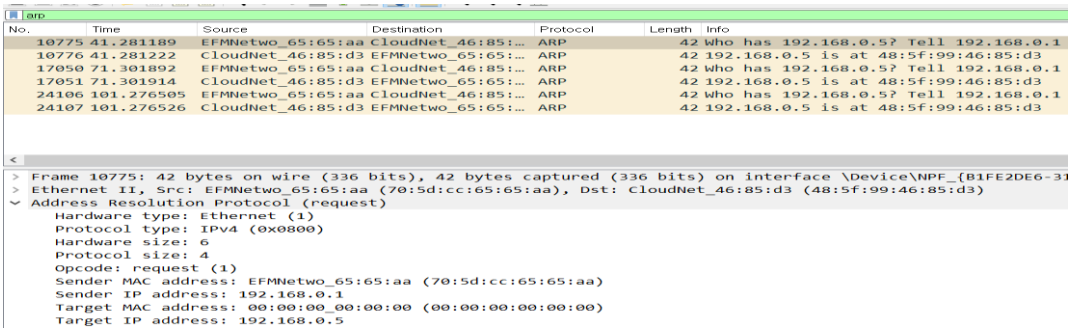
892	1.332067	192.168.0.4	61.78.36.10	TCP	66 55477 → 80 [ACK] Seq=550433955 Ack=2121416297 Win=261120 Len=0 SLE=2121415956 SRE=2121416297
1154	16.282351	61.78.36.10	192.168.0.4	TCP	54 80 → 55477 [FIN, ACK] Seq=2121416297 Ack=550433955 Win=47104 Len=0
1155	16.282394	192.168.0.4	61.78.36.10	TCP	54 55477 → 80 [ACK] Seq=550433955 Ack=2121416298 Win=261120 Len=0
1342	110.164494	192.168.0.4	61.78.36.10	TCP	54 55477 → 80 [FIN, ACK] Seq=550433955 Ack=2121416298 Win=261120 Len=0
1383	110.464893	192.168.0.4	61.78.36.10	TCP	54 [TCP Retransmission] 55477 → 80 [FIN, ACK] Seq=550433955 Ack=2121416298 Win=261120 Len=0
1389	111.065142	192.168.0.4	61.78.36.10	TCP	54 [TCP Retransmission] 55477 → 80 [FIN, ACK] Seq=550433955 Ack=2121416298 Win=261120 Len=0
1396	112.265039	192.168.0.4	61.78.36.10	TCP	54 [TCP Retransmission] 55477 → 80 [FIN, ACK] Seq=550433955 Ack=2121416298 Win=261120 Len=0
1403	114.664683	192.168.0.4	61.78.36.10	TCP	54 [TCP Retransmission] 55477 → 80 [FIN, ACK] Seq=550433955 Ack=2121416298 Win=261120 Len=0
1413	119.464962	192.168.0.4	61.78.36.10	TCP	54 [TCP Retransmission] 55477 → 80 [FIN, ACK] Seq=550433955 Ack=2121416298 Win=261120 Len=0
1464	129.065455	192.168.0.4	61.78.36.10	TCP	54 55477 → 80 [RST, ACK] Seq=550433956 Ack=2121416298 Win=0 Len=0

TCP connection 을 종료하던 중, [RST, ACK]이 나타났다. 이는, 존재하지 않는 Destination Port 로 TCP Request 가 오거나, 임의로 Connection 을 중지하는 등의 이유로 발생한다. 정상적인 Connection 종료가 이루어지려면, 61.78.36.10 의 IP 를 가진 호스트(이하 A)가 192.168.0.4 의 IP 를 가진 호스트(이하 B)에게 [ACK]을 보내주어야 한다. **[사진 11]**의 경우, A 는 Retransmission 을 반복하며 B 의 [ACK]을 기다리지만 [ACK]은 오지 않는다. 결국 Reset 을 하기 위해 flag 의 RST bit 를 1 로 설정하며 Connection 을 비정상적으로 종료한다. 이는, B 가 [FIN, ACK]을 A 에게 보내준 후, 먼저 Connection 을 종료하여 위와 같은 결과가 나왔을 가능성이 높다.

4. 새로 알게 된 지식

1) ARP (Address Resolution Protocol)

<Request>

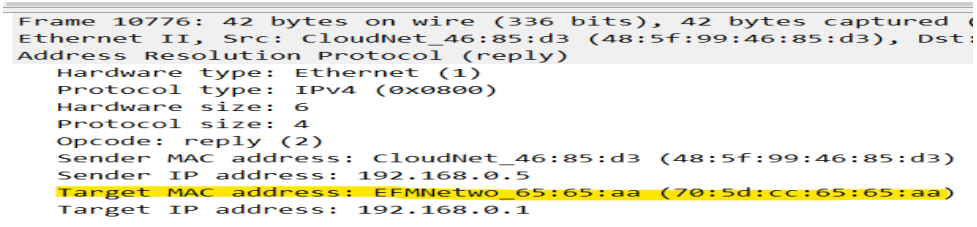


arp

No.	Time	Source	Destination	Protocol	Length	Info
10775	41.281189	EFMNetwo_65:65:aa	CloudNet_46:85:…	ARP	42	Who has 192.168.0.5? Tell 192.168.0.1
10776	41.281222	CloudNet_46:85:d3	EFMNetwo_65:65:…	ARP	42	192.168.0.5 is at 48:5f:99:46:85:d3
17050	71.301892	EFMNetwo_65:65:aa	CloudNet_46:85:…	ARP	42	Who has 192.168.0.5? Tell 192.168.0.1
17051	71.301914	CloudNet_46:85:d3	EFMNetwo_65:65:…	ARP	42	192.168.0.5 is at 48:5f:99:46:85:d3
24106	101.276505	EFMNetwo_65:65:aa	CloudNet_46:85:…	ARP	42	Who has 192.168.0.5? Tell 192.168.0.1
24107	101.276526	CloudNet_46:85:d3	EFMNetwo_65:65:…	ARP	42	192.168.0.5 is at 48:5f:99:46:85:d3

```
> Frame 10775: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{B1FE2DE6-31
> Ethernet II, Src: EFMNetwo_65:65:aa (70:5d:cc:65:65:aa), Dst: CloudNet_46:85:d3 (48:5f:99:46:85:d3)
v Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: EFMNetwo_65:65:aa (70:5d:cc:65:65:aa)
    Sender IP address: 192.168.0.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.0.5
```

<Reply>



```
Frame 10776: 42 bytes on wire (336 bits), 42 bytes captured (
Ethernet II, Src: CloudNet_46:85:d3 (48:5f:99:46:85:d3), Dst:
Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: CloudNet_46:85:d3 (48:5f:99:46:85:d3)
    Sender IP address: 192.168.0.5
    Target MAC address: EFMNetwo_65:65:aa (70:5d:cc:65:65:aa)
    Target IP address: 192.168.0.1
```

- Wireshark 를 사용하던 도중, ARP 라는 Protocol 이 사용되는 것을 보게 되었다. ARP 는 네트워크 상에서 IP 주소를 기반으로 MAC 주소를 알아낼 때 사용한다. MAC address 가

EFMNETtwo_65:65:aa 이고, IP 주소가 192.168.0.1 인 호스트 A 가 있다. A 는 192.168.0.1 의 IP 주소를 가진 호스트 B 의 MAC 주소를 알기 위해 ARP request 를 보냈다. 이를 받은 B 는 A 에게 자신의 MAC 주소는 CloudNet_46:85:d3 이라고 알려주었다.

- ARP 헤더 필드에 대해 알아보자.
①Hardware Type: 사용 중인 네트워크 유형을 나타낸다. Ethernet 의 경우 0x0001 이다.
②Protocol Type: IPv4, IPv6 등의 Protocol 중 어떤 것을 사용하는지 알려준다. IPv4 의 경우 0x0800 으로 설정된다.
③Hardware size: 논리주소의 길이를 정의하며, Ethernet 환경의 경우 6 byte 이다.
④Protocol Length: IPv4, IPv6 등의 Protocol 주소의 길이를 알려주며, IPv4 의 경우 4byte 이다.
⑤Operation: ARP request 는 1, ARP Reply 는 2 로 설정된다.
⑥Sender MAC address: 발신자의 MAC 주소이다.
⑦Sender Protocol Address: 발신자의 IP 주소이다.
⑧Target Hardware Address: 목적지의 MAC 주소이다. ARP request 의 경우, 목적지의 MAC 주소를 알 수 없기 때문에 0 으로 설정된다.
⑨Target Protocol Address: 목적지의 IP 주소이다.

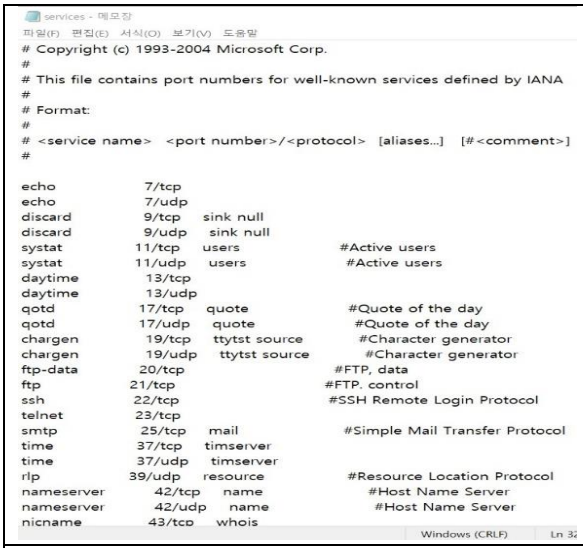
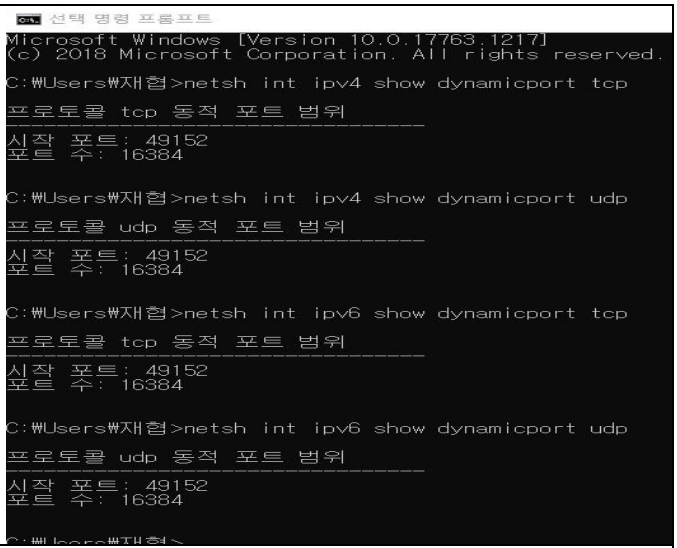
- ARP 정보는 누가 보냈는지 검증할 수 있는 수단이 없다. 따라서, 같은 네트워크 상에 있는 사용자가 변조된 정보를 보낼 경우 그 정보를 받은 사용자는 잘못된 MAC 주소로 패킷을 보내게 된다.
- ARP 와는 반대로 MAC 주소를 이용하여 IP 주소를 알아내는 RARP(Reverse Address Resolution Protocol)도 있다. 일반적으로 IP 주소는 하드 디스크 내의 구성 파일에 의해 설정된다. 그러나, 예전에는 하드 디스크가 달리지 않은 단말이 있어 자신의 IP 가 알지 못한 채로 부팅되는 경우가 있었다. 이 때, 자신의 MAC 주소를 이용해서 서버에게 IP 주소를 물어볼 때 RARP 가 사용되었다. 최근에는 대부분의 단말들이 하드 디스크를 사용하므로 RARP 의 사용 빈도는 낮은 편이다.

2) 포트 번호

0 번~1023 번: 잘 알려진 포트(well-known port)이며 널리 사용되는 유명한 어플리케이션 프로토콜을 위해 사용된다. 25 번을 사용하는 SMTP, 53 번을 이용하는 DNS, 80 번을 사용하는 HTTP 등이 그 예시이다.
1024~49151 번: 등록된 포트(registered port)이며 기관이나 사업자들을 위해서 IANA 에서 관리하는 포트이다. 1080 번을 사용하는 SCOKS 프록시(서버-클라이언트 간의 TCP 나 UDP 통신을 프록시 서버를 거쳐 진행하도록 해주는 프로토콜), 3306 번을 사용하는 MySQL

등이 있다.

49152 번~65535 번: 동적 포트(dynamic port)는 일반 사용자들이 자유롭게 사용할 수 있으며 포트 번호의 중복이나 불일치가 일어나지 않도록 번호를 임의로 부여한다.

<table>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>C:₩Windows₩System32₩drivers₩etc₩ 경로의 service 파일을 열어보면 이미 배정된 포트들과 그 역할을 확인할 수 있다.</td>
<td>동적 포트의 시작 번호와 범위는 위의 명령어로 확인할 수 있다.
시작 포트는 49152 이고 총 16384 개의 포트가 사용되고 있다는 것을 확인할 수 있다.</td>
</tr>
</table>

5. 참고 문헌

Chris Sanders.『와이어샤크를 활용한 실전 패킷 분석 – 2nd Edition』. 에이콘, 2012.

Jim Kurose and Keith Ross. *Computer Networking: A Top-Down Approach,* 2017