Wireshark를 이용한 transport layer traffic 분석

제출자 : 미디어학과 201821048 이서영

1. 개요

UDP와 TCP 프로토콜에 대해 관찰하고 분석했다. UDP는 신뢰성을 보장하지 않고 TCP는 신뢰성을 보장한다. 각 패킷에서 어떤 필드가 있는지, 그 필드가 어떤 역할을 하는지 살펴보았다. 그리고 동영상 재생 서비스에서 UDP가 사용될 때와 TCP가 사용될 때의 차이에 대해 알아보았다.

2. 관찰 방법

유튜브에서 영상을 보면서 wireshark를 동작시켰다. UDP의 특성에 맞는 영상 스트리밍 서비스에서 작동시켰다. 캡처기능과 통계기능을 이용해 분석했다.

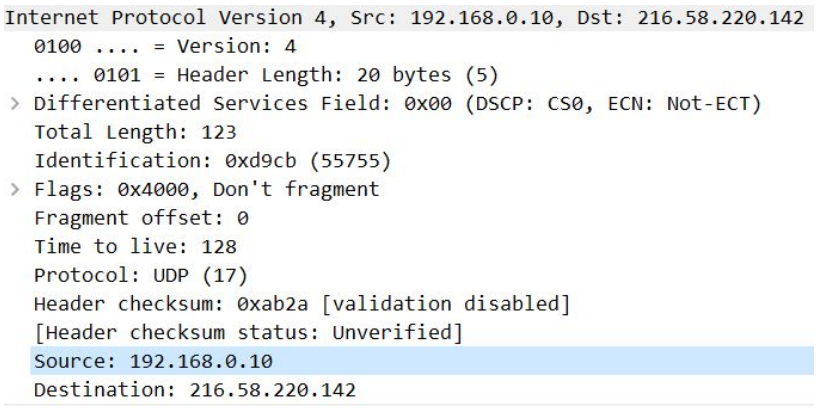
3. Traffic 분석

1) UDP 분석

유튜브에서 영상을 재생한 후 패킷을 캡쳐했다. 그 중 UDP프로토콜 분석을 위해 UDP만 필터링해서 캡쳐를 했다.

udp

No.	Time	Source	Destination	Protocol	Length	Info
9369	58.484266	192.168.0.10	216.58.220.142	UDP	137	53088 → 443 Len=95
9362	54.933224	108.177.125.189	192.168.0.10	UDP	67	443 → 57309 Len=25
9361	54.837900	192.168.0.10	108.177.125.189	UDP	75	57309 → 443 Len=33
9358	52.935858	192.168.0.10	172.217.26.46	UDP	75	59663 → 443 Len=33
9357	52.917152	172.217.26.46	192.168.0.10	UDP	81	443 → 59663 Len=39
9356	52.917147	172.217.26.46	192.168.0.10	UDP	302	443 → 59663 Len=260
9355	52.888797	192.168.0.10	172.217.26.46	UDP	75	59663 → 443 Len=33
9354	52.887696	172.217.26.46	192.168.0.10	UDP	68	443 → 59663 Len=26
9353	52.870581	172.217.26.46	192.168.0.10	UDP	81	443 → 59663 Len=39
9351	52.801071	192.168.0.10	172.217.26.46	UDP	1185	59663 → 443 Len=1143
9350	52.800976	192.168.0.10	172.217.26.46	UDP	1392	59663 → 443 Len=1350
9349	52.800881	192.168.0.10	172.217.26.46	UDP	1392	59663 → 443 Len=1350
9348	52.800697	192.168.0.10	172.217.26.46	UDP	1392	59663 → 443 Len=1350
9347	52.721460	172.217.26.3	192.168.0.10	UDP	67	443 → 64226 Len=25
9346	52.651796	192.168.0.10	172.217.26.3	UDP	75	64226 → 443 Len=33
9327	48.479554	192.168.0.10	172.217.175.106	UDP	137	61237 → 443 Len=95
9320	45.581469	216.58.197.182	192.168.0.10	UDP	68	443 → 50073 Len=26
9319	45.544653	192.168.0.10	216.58.197.182	UDP	75	50073 → 443 Len=33
9318	45.531856	192.168.0.10	216.58.197.182	UDP	75	50073 → 443 Len=33
9317	45.531733	216.58.197.182	192.168.0.10	UDP	576	443 → 50073 Len=534
9316	45.531730	216.58.197.182	192.168.0.10	UDP	1392	443 → 50073 Len=1350
9315	45.531481	216.58.197.182	192.168.0.10	UDP	1392	443 → 50073 Len=1350
9314	45.531478	216.58.197.182	192.168.0.10	UDP	1392	443 → 50073 Len=1350
9313	45.530749	216.58.197.182	192.168.0.10	UDP	1392	443 → 50073 Len=1350
9312	45.530745	216.58.197.182	192.168.0.10	UDP	1392	443 → 50073 Len=1350
9311	45.530367	216.58.197.182	192.168.0.10	UDP	1392	443 → 50073 Len=1350
9310	45.530298	216.58.197.182	192.168.0.10	UDP	1392	443 → 50073 Len=1350
9309	45.530294	216.58.197.182	192.168.0.10	UDP	1392	443 → 50073 Len=1350
9308	45.530154	216.58.197.182	192.168.0.10	UDP	1392	443 → 50073 Len=1350
9307	45.528541	216.58.197.182	192.168.0.10	UDP	1392	443 → 50073 Len=1350
9306	45.528036	192.168.0.10	216.58.197.182	UDP	75	50073 → 443 Len=33
9305	45.527813	216.58.197.182	192.168.0.10	UDP	1392	443 → 50073 Len=1350
9304	45.527702	216.58.197.182	192.168.0.10	UDP	1392	443 → 50073 Len=1350



source port, destination port, length, checksum 4개의 필드로 되어있다. 다른 프로토콜에 비해 단순하게 구성되어 있다. tcp헤더와는 다르게 수신 확인 여부를 확인하는 정보가 없고 패킷을 그냥 보낸다.

source port : 2byte 출발지 포트번호를 나타낸다. 클라이언트는 임의의 포트 번호를 사용하고 서버는 잘 알려진 포트 번호를 사용한다. 192.168.0.10이 나의 IP이다.

destination port : 2byte 응답 패킷에서 수신중인 포트를 열기 위해서, 전송하는 어플리케이션이나 프로토콜을 정의하기 위한 목적이다. 216.58.220.142이다.

length : 2byte UDP헤더에서 유효한 데이터의 끝까지 패킷의 길이를 정의한다.

checksum : 2byte 오류를 확인하기 위해 사용한다. UDP에서는 항상 체크섬을 요구하지 않는다. validation disabled로 확인하지 않아도 된다.

flags : 분할되었을 때 몇 번째인지 알려준다. 위에서는 분할되지 않았다. don't fragment라고 되어있다.

time to live : 살아있을 수 있는 기간, 몇 홉을 걸칠때까지 살 수 있을 지 나타낸다. 128개의 라우터를 거칠 수 있다.

UDP프로토콜은 신뢰성을 보장하지 않는다. 하지만 불필요한 메시지에 대한 정보를 제외하고 필요한 핵심 데이터만 전달한다는 점이 장점이다. 동영상 서비스를 할 때 UDP가 사용될 것이라고 추측하고 wireshark 패킷 캡쳐를 했다. 예상에 맞게 UDP를 이용해 대용량의 데이터를 빠르게 서비스하는데 사용되었다.

2) TCP 분석

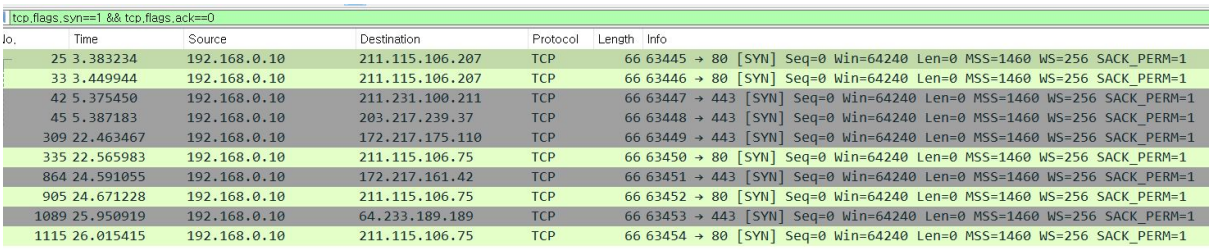
3 Way Handshake : 클라이언트와 서버는 데이터를 주고 받을 준비가 되었다는 것을 서로에게 알려주고 이후 데이터 전송에 필요한 시퀀스 번호를 알 수 있게 된다.

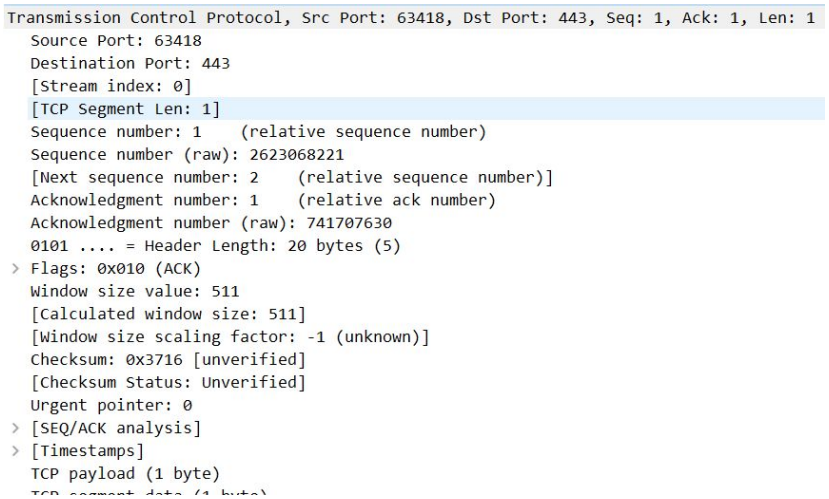
1번 라인: 클라이언트가 서버로 시퀀스 번호를 seq 필드에 담아 보냄

2번 라인: 서버는 클라이언트가 보내준 시퀀스 번호를 1 증가시켜서 ack 필드에 담아 보냄

3번 라인: 클라이언트는 다시 서버로부터 받은 시퀀스 번호를 1 증가시켜서 자신의 ack 필드에 담아 보냄

- connection setup 단계에서 교환하는 TCP segment header 분석





Source port : 발신지 포트필드는 발신지에서 오픈된 포트이다. 위의 패킷을 보면 63418인 것을 볼 수 있다.

destination port : 목적지 포트필드는 수신지에서 오픈된 포트이다. 443이다.

Sequence number : 순차번호필드는 고유한 번호를 가지며 이 값으로 TCP세그먼트에 대한 식별값을 제공한다. 신뢰성을 보장하는 효과가 있다. 패킷을 보내다 일부가 분실되면 확인을 위해 수신자를 사용가능하게 한다. 이 순차번호는 패킷에 포함되어있는 데이터만큼 증가한다. 3way handshaking에서 사용하며 보안을 위해 랜덤하게 생성한다.

Acknowledgement number : 확인 응답번호필드는 다음번에 올 것이라고 생각하는 순차번호를 표시한다.

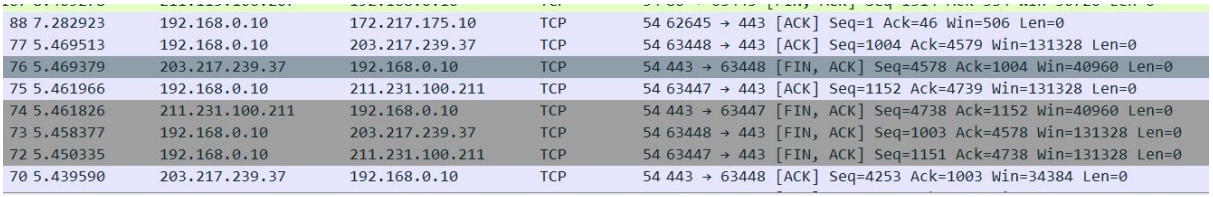
Header length : 데이터 오프셋 필드는 TCP헤더의 길이를 정의한다. 길이는 4byte씩 증가하고 값이 5라는 뜻은 20byte 길이를 갖는다는 의미이다.

Flags : 9개의 비트 플래그이다. 이 플래그들은 현재 세그먼트의 속성을 나타낸다. 기존에는 6개의 플래그만을 사용했지만, 혼잡 제어 기능의 향상을 위해 Reserved 필드를 사용하여 NS, CWR, ECE 플래그가 추가되었다.

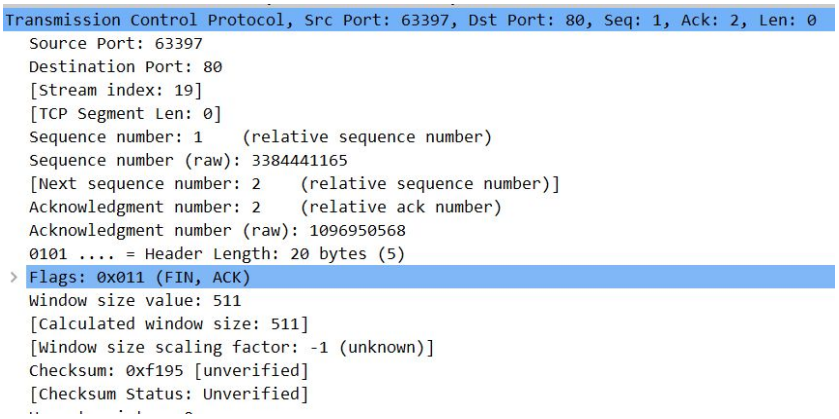
Data Offset : 데이터 오프셋 필드에는 전체 세그먼트 중에서 헤더가 아닌 데이터가 시작되는 위치가 어디부터인지를 표시한다.
window size : 윈도우 크기는 511이다. 수신자로부터 응답을 기다리지 않고 연속적으로 보낼 수 있는 바이트 수이다. 윈도로 한번에 보낼 수 있는 크기를 조정한다.

Checksum : 체크섬은 데이터를 송신하는 중에 발생할 수 있는 오류를 검출하기 위한 값이다. TCP의 체크섬은 전송할 데이터를 16 Bits씩 나눠서 차례대로 더해가는 방법으로 생성한다.

- connection 종료 단계에서 교환하는 TCP segment header 분석



88	7.282923	192.168.0.10	172.217.175.10	TCP	54 62645 → 443 [ACK] Seq=1 Ack=46 Win=506 Len=0
77	5.469513	192.168.0.10	203.217.239.37	TCP	54 63448 → 443 [ACK] Seq=1004 Ack=4579 Win=131328 Len=0
76	5.469379	203.217.239.37	192.168.0.10	TCP	54 443 → 63448 [FIN, ACK] Seq=4578 Ack=1004 Win=40960 Len=0
75	5.461966	192.168.0.10	211.231.100.211	TCP	54 63447 → 443 [ACK] Seq=1152 Ack=4739 Win=131328 Len=0
74	5.461826	211.231.100.211	192.168.0.10	TCP	54 443 → 63447 [FIN, ACK] Seq=4738 Ack=1152 Win=40960 Len=0
73	5.458377	192.168.0.10	203.217.239.37	TCP	54 63448 → 443 [FIN, ACK] Seq=1003 Ack=4578 Win=131328 Len=0
72	5.450335	192.168.0.10	211.231.100.211	TCP	54 63447 → 443 [FIN, ACK] Seq=1151 Ack=4738 Win=131328 Len=0
70	5.439590	203.217.239.37	192.168.0.10	TCP	54 443 → 63448 [ACK] Seq=4253 Ack=1003 Win=34384 Len=0



```
Transmission Control Protocol, Src Port: 63397, Dst Port: 80, Seq: 1, Ack: 2, Len: 0
  Source Port: 63397
  Destination Port: 80
  [Stream index: 19]
  [TCP Segment Len: 0]
  Sequence number: 1    (relative sequence number)
  Sequence number (raw): 3384441165
  [Next sequence number: 2    (relative sequence number)]
  Acknowledgment number: 2    (relative ack number)
  Acknowledgment number (raw): 1096950568
  0101 .... = Header Length: 20 bytes (5)
> Flags: 0x011 (FIN, ACK)
  Window size value: 511
  [Calculated window size: 511]
  [Window size scaling factor: -1 (unknown)]
  Checksum: 0xf195 [unverified]
  [Checksum Status: Unverified]
```

flag : FIN플래그를 보내 연결이 끝난 것을 나타낸다.

4 Way Handshake

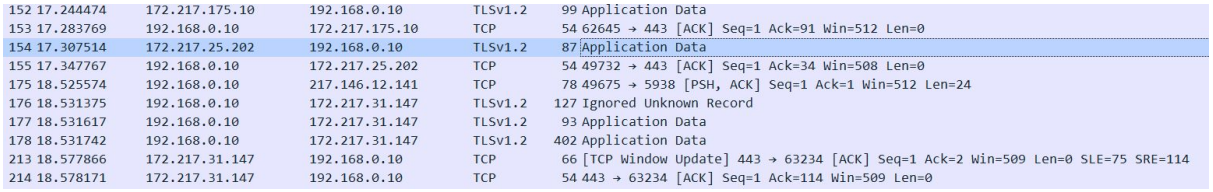
연결을 생성할 때와 마찬가지로, 연결을 종료할 때도 특정한 과정을 거쳐서 연결을 종료해야한다. 한 쪽에서 일방적으로 연결을 끊어버리면 다른 한 쪽은 연결이 끊어졌는지 지속되고 있는지 알 방법이 없다. 또한 연결을 종료하기 전에 아직 다 처리하지 못한 데이터가 있을 수도 있기 때문에 양 쪽이 다 정상적으로 연결을 종료할 준비가 되었는 지를 확인하는 과정이 필요한 것이다.

요청자 ---SEQ: 1---> 수신자

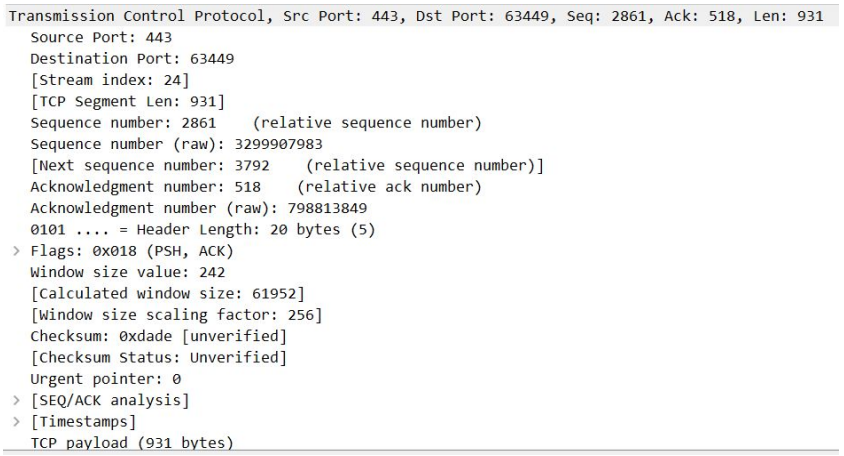
요청자 <---ACK: 2--- 수신자

요청자 ---FIN: 2---> 수신자

- application data를 포함한 TCP segment의 header 분석



```
152 17.244474    172.217.175.10    192.168.0.10      TLSv1.2    99 Application Data
153 17.283769    192.168.0.10      172.217.175.10    TCP        54 62645 → 443 [ACK] Seq=1 Ack=91 Win=512 Len=0
154 17.307514    172.217.25.202    192.168.0.10      TLSv1.2    87 Application Data
155 17.347767    192.168.0.10      172.217.25.202    TCP        54 49732 → 443 [ACK] Seq=1 Ack=34 Win=508 Len=0
175 18.525574    192.168.0.10      217.146.12.141    TCP        78 49675 → 5938 [PSH, ACK] Seq=1 Ack=1 Win=512 Len=24
176 18.531375    192.168.0.10      172.217.31.147    TLSv1.2   127 Ignored Unknown Record
177 18.531617    192.168.0.10      172.217.31.147    TLSv1.2    93 Application Data
178 18.531742    192.168.0.10      172.217.31.147    TLSv1.2   402 Application Data
213 18.577866    172.217.31.147    192.168.0.10      TCP        66 [TCP Window Update] 443 → 63234 [ACK] Seq=1 Ack=2 Win=509 Len=0 SLE=75 SRE=114
214 18.578171    172.217.31.147    192.168.0.10      TCP        54 443 → 63234 [ACK] Seq=1 Ack=114 Win=509 Len=0
```



```
Transmission Control Protocol, Src Port: 443, Dst Port: 63449, Seq: 2861, Ack: 518, Len: 931
  Source Port: 443
  Destination Port: 63449
  [Stream index: 24]
  [TCP Segment Len: 931]
  Sequence number: 2861    (relative sequence number)
  Sequence number (raw): 3299907983
  [Next sequence number: 3792    (relative sequence number)]
  Acknowledgment number: 518    (relative ack number)
  Acknowledgment number (raw): 798813849
  0101 .... = Header Length: 20 bytes (5)
> Flags: 0x018 (PSH, ACK)
  Window size value: 242
  [Calculated window size: 61952]
  [Window size scaling factor: 256]
  Checksum: 0xdade [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
> [SEQ/ACK analysis]
> [Timestamps]
  TCP payload (931 bytes)
```

Next sequence number : 3792

sequence number : 2861

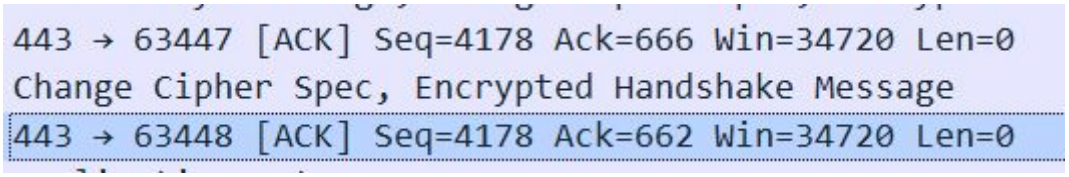
TCP segment length : 931

3792 - 2861 = 931인 것을 확인할 수 있다.

3) TCP 분석 심화

sliding window : 수신 측이 한 번에 처리할 수 있는 데이터를 정해놓고 그때 그때 수신 측의 데이터 처리 상황을 송신 측에 알려줘서 데이터의 흐름을 제어하는 방식이다. 송신 측이 처리할 수 있는 데이터의 양을 알고 있다. 이 정보를 알고있기 때문에 수신 측에서 처리 가능이라는 대답을 다 하지 않아도 데이터를 보내기 전에 예측이 가능해진다. 송신 측과 수신 측은 각각 데이터를 담을 수 있는 버퍼를 가지고 있고 송신측에서는 윈도우에 들어있는 데이터를 수신쪽에서 응답이 없어도 연속적으로 보낼 수 있다.

ARQ : automatic repeat request 로 재전송 기반의 오류 제어를 사용한다. 통신 중에 오류가 발생하면 송신 측에서 수신 측에게 해당 데이터를 다시 제공한다. 통신 중에 뭔가 오류가 발생하면 송신 측이 수신 측에게 해당 데이터를 다시 전송해야한다. 하지만 이 재전송이라는 작업 자체가 했던 일을 또 해야하는 비효율적인 작업이기 때문에, 이 재전송 과정을 최대한 줄일 수 있는 여러가지 방법을 사용하게 된다.



Go Bank N : 데이터를 연속적으로 보내다가 그 중 어느 데이터부터 오류가 발생했는지를 검사하는 방식이다. Go Back N 방식을 사용하면 데이터를 연속적으로 보낸 후 한 개의 ACK나 NACK만을 사용하여 수신 측의 처리 상황을 파악할 수 있으므로, 연속적으로 데이터를 보낼 수 있는 흐름 제어 방식인 슬라이딩 윈도우와 아주 잘 들어맞는다. 예를 들어 송신 측은 수신 측으로 NACK를 받고나면 오류가 발생한 4번 데이터와 그 이후 전송했던 모든 데이터를 다시 전송해줘야 한다는 말이 된다. 이때 송신 측은 비록 5번까지 전송했지만 오류가 발생하면, 오류가 발생한 4번 데이터로 되돌아가서 다시 전송해야하므로 Go Back N이라고 부르는 것이다.

Selective Repeat은 말 그대로 선택적인 재전송을 의미한다. Stop and Wait와 Go Back N 방식과 다르게, 이 방식을 사용하는 수신 측의 버퍼에 쌓인 데이터가 연속적이지 않다는 단점이 존재한다. 송신 측이 데이터를 재전송하게되면 수신 측은 이 데이터를 버퍼 중간 어딘가에 끼워넣어서 데이터를 정렬해야한다. 이때 같은 버퍼 안에서 데이터를 정렬할 수는 없으니, 별도의 버퍼가 필요하게 된다. 결국 재전송이라는 과정이 빠진 대신 재정렬이라는 과정이 추가된 것인데, 이 둘 중에 재전송이 좀 더 이득인 상황에서는 Go Bank N 방식을, 재정렬이 좀 더 이득인 상황에서는 Selective Repeat 방식을 사용하면된다.

4. 새로 알게 된 지식

HOLB(Head of line Blocking) :
기존의 HTTP에는 HOLB(Head of Line Blocking)이라고 하는 문제가 있다. HTTP 레벨에서의 HOLB와 TCP 레벨에서의 HOLB는 다른 의미이기는 하나 결국 어떤 요청에 병목이 생겨서 전체적인 레이턴시가 늘어난다는 맥락으로 본다면 동일하다고 할 수 있다.

TCP를 사용한 통신에서 패킷은 무조건 정확한 순서대로 처리되어야 한다. 수신 측은 송신 측과 주고받은 시퀀스 번호를 참고하여 패킷을 재조립해야하기 때문이다. 그래서 통신 중간에 패킷이 손실되면 완전한 데이터로 다시 조립할 수 없기 때문에 절대로 그냥 넘어가지 않는다. 무조건 송신 측은 수신 측이 패킷을 제대로 다 받았다는 것을 확인한 후, 만약 수신 측이 제대로 패킷을 받지 못했으면 해당 패킷을 다시 보내야 한다.

또한 패킷이 처리되는 순서 또한 정해져있으므로 이전에 받은 패킷을 파싱하기 전까지는 다음 패킷을 처리할 수도 없다. 이렇게 패킷이 중간에 유실되거나 수신 측의 패킷 파싱 속도가 느리다면 통신에 병목이 발생하게 되는 현상을 HOLB라고 부르는 것이다. 이건 TCP 자체의 문제이므로 HTTP/1 뿐만 아니라 HTTP/2도 가지고 있는 문제이다.