Wireshark를 이용한 TCP 분석

목차

제출자

201323148

소프트웨어학과

이제호

1. 관찰 목표 및 예상 결과

1-(1) 관찰 목표

1차 과제는 application service가 어떤 application layer message를 발생시키는지 관찰하고 해당 service에 대한 이해도를 높이는 것이 목적인 반면에, 이번 과제는 application service를 사용할 때, communication하는 두 TCP endpoint 사이의 connection이 setup되고 close되기 까지 발생하는 TCP segment의 흐름 관찰을 통해 TCP가 어떻게 신뢰적인 데이터 전송을 이루어 내는지 확인하는 것이 목적이다.

본 보고서에서는 일련의 웹 브라우징 과정을 통해 다양한 contents를 로딩하는 과정을 거치며 얻은 capture 결과를 분석해 TCP connection setup and closing, end-to-end segment flow, TCP segment structure을 살펴 볼 것이다. 또한 재전송과 같은 특별한 상황들도 분석해볼 것이다.

1-(2) 예상 결과

본 보고서에서는 TCP segment와 TCP message라는 용어를 같이 사용하도록 하겠다.

우선 web server로부터 contents data를 전송 받아 브라우저에 로딩하기 위한 첫 단계로 client(browser)와 server간에 TCP connection을 설정해야 한다. 그 전에 DNS query와 response 메시지가 발생하지만 Transport layer에 집중하기 위해 고려하지 않겠다. TCP connection을 설정하기 위한 TCP 3-way handshake message가 먼저 관찰 될 것이고 connection이 설정되면 data를 주고받는 HTTP 요청과 응답에 대한 segment가 발생할 것이다. 이때 대부분의 segment는 server로부터 전송된 segment일 것으로 예측된다. 그 이유는 브라우징 과정이 대부분 server로부터 얻어온 data를 loading하는 과정이고, 보통 HTTP request 메시지에 비해 response 메시지의 크기가 훨씬 커서 MSS단위로 쪼개진 TCP segment들이 많이 발생할 것이기 때문이다. data 전송이 완료되면 connection을 종료하기 위한 4개의 TCP message를 관찰할 수 있을 것이다.

위와 같은 과정은 브라우저가 TCP/IP 기반 인터넷에서 HTTP를 사용해 data를 주고 받는 전형적인 과정인 반면, TCP segment loss가 발견되어 segment가 재전송되는 과정을 거치는 TCP connection도 존재할 것이다. 이러한 connection에서는 여러 duplicate ACK segment 이후에 TCP fast retransmission segment가 발생하거나 TCP retransmission segment가 발생할 것이다.

2. 관찰 방법

이번 과제에서는 1차 과제와 마찬가지로 웹 브라우징 과정을 거치며 capture를 진행할 것이다.
웹 서핑을 할 때 많은 TCP connection들이 생기고 없어질 것이며 특정 connection에서 발생하는 segment의 loss를 예측할 수 없고(예측 불가능한 네트워크 상황), 따라서 여러 가지 상황을 capture해놓는 것이 TCP의 기능을 관찰하는데 도움이 될 것이다. 그러므로 1차 과제와 같이 특정 application service의 메시지를 관찰하기 위한 specific scenario를 예측하여 진행하기 보다는, 의식의 흐름대로 인터넷을 사용하듯이 다양한 웹 사이트를 방문하고 많은 contents를 로딩하며 capture를 진행하도록 하겠다.
미리 실험하는 과정에서 페이지를 많이 이동하지 않았는데 재전송을 포함한 다양한 상황이 발생되는 것을 보아 capturing을 길게 할 필요는 없다고 판단하였다. 다만 쇼핑몰 페이지와 같이 많은 양의 contents가 포함된 무거운 페이지로 이동하며 TCP segment loss를 유도할 수는 있을 것이다.
대략적인 관찰 과정은 네이버의 웹툰, 뉴스와 아주대학교의 공지사항 페이지, 온라인 쇼핑몰 페이지 등을 방문하여 원하는 contents를 보는 과정으로 capture를 진행하겠다.
capture 결과 TCP message의 수는 매우 많을 것이고 가독성을 위해 TCP connection(stream)을 한 개씩 filtering하여 관찰할 것이다. Wireshark에서 특정 TCP stream index를 가진 message를 follow 하면 해당 stream index를 가진 message만 확인할 수 있다. 즉, 해당 index를 가진 TCP connection의 모든 TCP message들을 볼 수 있다. Traffic 분석에서는 stream follow filter를 이용해 각 파트를 잘 설명할 수 있는 대표적인 TCP connection을 선정하여 분석하도록 하겠다.

3. Traffic 분석

관찰 장소 : 아주대학교 중앙도서관 2층 커뮤니티 라운지
관찰 기간 : 약 245초

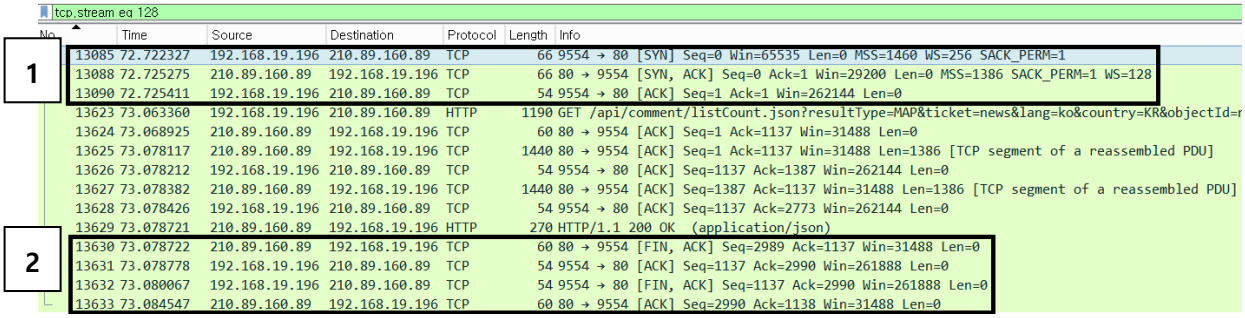
<Protocol Hierarchy>

Protocol	Percent Packets	Packets	Percent Bytes
Frame	100.0	49921	100.0
Ethernet	100.0	49921	1.6
Internet Protocol Version 6	0.0	7	0.0
Internet Protocol Version 4	99.9	49850	2.3
User Datagram Protocol	0.6	287	0.0
Transmission Control Protocol	99.3	49555	95.9

위 사진은 Protocol Hierarchy기능을 통해 살펴본 계층 구조로, 주목할 점은 웹 브라우저 응용 프로그램을 사용하는데 있어 Network에 발생한 bytes 흐름은 TCP/IP protocol이 96.5%로 대부분의 비율을 차지하고 있다는 점이다. 웹 서비스를 이용함에 있어 TCP/IP의 중요성을 실감할 수 있는 수치이다.

(1) TCP segment structure, TCP connection의 setup과 close

<TCP stream index = 128인 TCP connection>



위 사진은 TCP stream index가 128인 TCP connection에서 발생한 TCP message의 전부이다. 네이버 뉴스의 특정 기사 페이지로 이동했을 때 발생한 TCP connection이고 comment object에 대한 json object를 요청하기 위한 것으로 추정된다. 위와 같은 간결한 TCP communication을 통해 segment 구조와 setup, close 과정을 살펴보도록 하겠다.

(1) - 1. TCP segment structure

```
Transmission Control Protocol, Src Port: 9554, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 9554
    Destination Port: 80
    [Stream index: 128]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    Acknowledgment number: 0
    1000 .... = Header Length: 32 bytes (8)
  ▾ 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
    [Calculated window size: 65535]
    Checksum: 0xa59e [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▾ Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window sca
    ▸ TCP Option - Maximum segment size: 1460 bytes
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - Window scale: 8 (multiply by 256)
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - SACK permitted
```

```
Transmission Control Protocol, Src Port: 80, Dst Port: 9554, Seq: 0, Ack: 1, Len: 0
    Source Port: 80
    Destination Port: 9554
    [Stream index: 128]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    Acknowledgment number: 1    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  ▾ 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: 29200
    [Calculated window size: 29200]
    Checksum: 0x0f33 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▾ Options: (12 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP)
    ▸ TCP Option - Maximum segment size: 1386 bytes
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - SACK permitted
    ▸ TCP Option - No-Operation (NOP)
    ▸ TCP Option - Window scale: 7 (multiply by 128)
  ▾ [SEQ/ACK analysis]
      [This is an ACK to the segment in frame: 13085]
      [The RTT to ACK the segment was: 0.002948000 seconds]
      [iRTT: 0.003084000 seconds]
```

위 두 사진은 각각 SYN segment(왼쪽), SYNACK segment(오른쪽)을 나타낸다. 공통적인 segment의 구조를 살펴보도록 하겠다.

Source/Dest Port	출발지와 목적지 port 번호를 나타낸다. 웹 서비스를 이용할 때는 port 80로 dest port를 설정한다.
Sequence number	32bit의 순서번호 field로 전송하려는 data의 첫 byte의 순서번호를 가리킨다.
Acknowledge number	32bit의 ACK번호 field로 receiver로서 전송 받기를 기다리는 data의 순서번호를 가리킨다. Sequence#, Acknowledge#는 신뢰적 데이터 전송을 위한 중요한 요소이다.
Header Length	TCP segment의 헤더 길이를 나타낸다. 보통의 길이는 20byte이지만 SYN, SYNACK segment의 경우 options field 12byte가 포함되어 있기 때문에 32byte이다.
Flags	Urgent - 보통 쓰이지 않는 flag로 urgent한 data임을 나타내기 위한 flag이다. Acknowledgment - 1일 경우 Acknowledgment number field의 값이 유효하다는 것을 의미한다. Push - 1일 경우 receiver가 data를 상위 계층(application)으로 바로 전달해야 한다는 것을 나타낸다. Reset - Reset flag bit는 connection을 재설정하거나 즉시 연결을 끊고 싶을 때 설정하는 flag이다. Syn - Syn bit는 connection을 setup하기 원하는 TCP client측에서 설정하여 보내는 flag로 해당 flag를 1로 설정하여 보내면 해당 client는 SYN_SENT상태가 된다. Fin - Fin bit는 connection을 close하기 원하는 TCP client 또는 server에 의해 설정되는 flag이다.
Window size value	흐름제어에 사용되는 field로 receiver로서의 receiver buffer 용량을 상대편에게 reporting하기 위해 사용된다.
Checksum	data가 올바르게 전송되었는지 확인할 수 있는 field
Urgent pointer	Urg flag가 1로 설정된 경우, data의 어떤 부분이 Urgent한지 pointing 해준다.
Options	TCP Option을 포함하는 field로 다음과 같은 option이 있다. 1. Maximum segment size - MSS는 segment의 data field의 크기를 제한하는 값으로 해당하는 Network interface의 MTU(Max Transmission Unit)에 의해 결정된다. SYN segment와 SYNACK segment에 의해 TCP message를 한 개씩 주고 받으며 서로의 MSS 값을 공유하고 두 값 중 작은 MSS 값이 해당 TCP connection의 MSS가 된다. 2. Window Scale - 연결이 설정 된 후, data를 송수신하는 TCP segment의 Window size value는 아주 작다. 이 값은 실제 buffer size를 나타내는 것이 아니고 Window Scale에 따라 계산된 값이 실제 buffer size이다. 예를 들어, Window Scale이 8로 설정된 SYN segment를 서버 측에서 받으면 서버는 이후에 받는 segment의 Window size value에 256(=2^8)을 곱한 값이 client측의 실제 Window size value라고 인식한다. 이 옵션은 Window size value field가 2 byte로 밖에 표현할 수 없기 때문에 더 큰 값을 나타내기 위해 사용한다. 3. No-Operation - Options field의 의미 있는 option들을 구분하기 위해 사용한다.

(1) – 2. TCP connection setup and close

(128 stream) TCP connection 사진의 1번 박스는 setup 과정을 나타내고, 2번 박스는 close 과정을 나타낸다.

우선 1번 박스의 connection setup과정을 먼저 살펴보겠다.

<TCP connection setup>

13085	72.722327	192.168.19.196	210.89.160.89	TCP	66	9554 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
13088	72.725275	210.89.160.89	192.168.19.196	TCP	66	80 → 9554 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1386 SACK_PERM=1 WS=128
13090	72.725411	192.168.19.196	210.89.160.89	TCP	54	9554 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0

위와 같은 TCP connection setup 과정은 TCP 3-way handshake라고 불리며, data를 송수신하기 위해 선행되어야 한다. 우선 연결을 맺고자 하는 TCP client는 TCP server측에 SYN flag를 1로 설정하여 SYN segment를 전송한다. 이때 SYN segment에는 MSS와 Window Scale에 대한 정보도 포함된다. SYN segment를 받은 TCP server은 해당 client에 대한 자원을 할당해두고, 마찬가지로 MSS와 Window Scale에 대한 정보를 포함해 SYN ACK segment를 client측에 전송한다.

SYN segment와 SYNACK segment를 통해 TCP connection의 초기 정보들이 설정된다. 먼저 client와 server의 MSS 중 작은 MSS 값인 MSS = 1386이 선택되고, SYN segment가 보내진 시점에서부터 SYN ACK segment를 받는 시점까지의 RTT에 여유 값을 더해 initial RTT(iRTT)를 설정한다.

마지막으로 client측에서 ACK segment를 server에게 보내면 connection이 설정되고 data를 송수신 할 수 있게 된다. Client는 ACK segment에 data를 포함시켜 보낼 수 있고, receiver window size를 포함시켜 전송한다.

<TCP connection close>

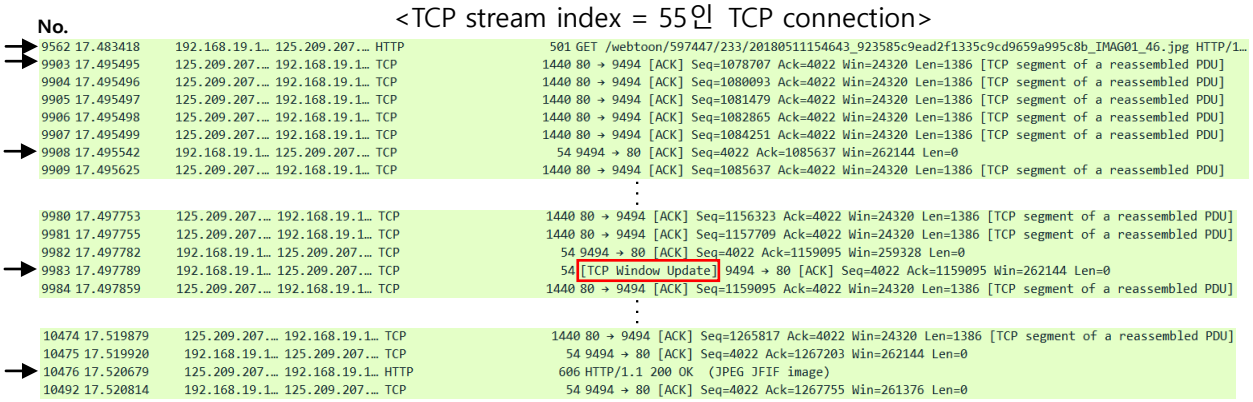
13630	73.078722	210.89.160.89	192.168.19.196	TCP	60	80 → 9554 [FIN, ACK] Seq=2989 Ack=1137 Win=31488 Len=0
13631	73.078778	192.168.19.196	210.89.160.89	TCP	54	9554 → 80 [ACK] Seq=1137 Ack=2990 Win=261888 Len=0
13632	73.080067	192.168.19.196	210.89.160.89	TCP	54	9554 → 80 [FIN, ACK] Seq=1137 Ack=2990 Win=261888 Len=0
13633	73.084547	210.89.160.89	192.168.19.196	TCP	60	80 → 9554 [ACK] Seq=2990 Ack=1138 Win=31488 Len=0

위 사진은 TCP connection close 과정으로 예상대로 4개의 메시지가 발생했다. 우선 connection을 종료하길 원하는 server측에서 FIN bit을 1로 설정한 FIN ACK segment를 client에게 보낸다. FIN ACK segment를 받은 client는 해당 segment에 대한 ACK segment를 전송한 후, server측에게 더 이상 보낼 data가 없는지 확인한다. 더 이상 전송할 data가 없다면 client는 server측에게 FIN ACK segment를 보내고, server에 의해 마지막으로 전송된 ACK segment를 받게 되면 해당 TCP connection은 close된다.

(2) TCP data segment flow between two endpoints

TCP data segment의 흐름을 설명할 때 Sequence, Acknowledge Number의 역할이 중요한데, 각 number는 relative number로 실제 순서번호가 아닌 상대적인 순서번호이다. 이는 Wireshark에서 제공하는 기본 기능으로 connection의 시작 segment인 SYN segment의 순서번호를 0으로 두어 가독성을 높이기 위함인데, 실제 순서번호는 0이 아닌 다른 값이다.

<TCP stream index = 55인 TCP connection>



No.					
9562	17.483418	192.168.19.1…	125.209.207.…	HTTP	501 GET /webtoon/597447/233/20180511154643_923585c9ead2f1335c9cd9659a995c8b_IMAG01_46.jpg HTTP/1…
9903	17.495495	125.209.207.…	192.168.19.1…	TCP	1440 80 → 9494 [ACK] Seq=1078707 Ack=4022 Win=24320 Len=1386 [TCP segment of a reassembled PDU]
9904	17.495496	125.209.207.…	192.168.19.1…	TCP	1440 80 → 9494 [ACK] Seq=1080093 Ack=4022 Win=24320 Len=1386 [TCP segment of a reassembled PDU]
9905	17.495497	125.209.207.…	192.168.19.1…	TCP	1440 80 → 9494 [ACK] Seq=1081479 Ack=4022 Win=24320 Len=1386 [TCP segment of a reassembled PDU]
9906	17.495498	125.209.207.…	192.168.19.1…	TCP	1440 80 → 9494 [ACK] Seq=1082865 Ack=4022 Win=24320 Len=1386 [TCP segment of a reassembled PDU]
9907	17.495499	125.209.207.…	192.168.19.1…	TCP	1440 80 → 9494 [ACK] Seq=1084251 Ack=4022 Win=24320 Len=1386 [TCP segment of a reassembled PDU]
9908	17.495542	192.168.19.1…	125.209.207.…	TCP	54 9494 → 80 [ACK] Seq=4022 Ack=1085637 Win=262144 Len=0
9909	17.495625	125.209.207.…	192.168.19.1…	TCP	1440 80 → 9494 [ACK] Seq=1085637 Ack=4022 Win=24320 Len=1386 [TCP segment of a reassembled PDU]
9980	17.497753	125.209.207.…	192.168.19.1…	TCP	1440 80 → 9494 [ACK] Seq=1156323 Ack=4022 Win=24320 Len=1386 [TCP segment of a reassembled PDU]
9981	17.497755	125.209.207.…	192.168.19.1…	TCP	1440 80 → 9494 [ACK] Seq=1157709 Ack=4022 Win=24320 Len=1386 [TCP segment of a reassembled PDU]
9982	17.497782	192.168.19.1…	125.209.207.…	TCP	54 9494 → 80 [ACK] Seq=4022 Ack=1159095 Win=259328 Len=0
9983	17.497789	192.168.19.1…	125.209.207.…	TCP	54 [TCP Window Update] 9494 → 80 [ACK] Seq=4022 Ack=1159095 Win=262144 Len=0
9984	17.497859	125.209.207.…	192.168.19.1…	TCP	1440 80 → 9494 [ACK] Seq=1159095 Ack=4022 Win=24320 Len=1386 [TCP segment of a reassembled PDU]
10474	17.519879	125.209.207.…	192.168.19.1…	TCP	1440 80 → 9494 [ACK] Seq=1265817 Ack=4022 Win=24320 Len=1386 [TCP segment of a reassembled PDU]
10475	17.519920	192.168.19.1…	125.209.207.…	TCP	54 9494 → 80 [ACK] Seq=4022 Ack=1267203 Win=262144 Len=0
10476	17.520679	125.209.207.…	192.168.19.1…	HTTP	606 HTTP/1.1 200 OK (JPEG JFIF image)
10492	17.520814	192.168.19.1…	125.209.207.…	TCP	54 9494 → 80 [ACK] Seq=4022 Ack=1267755 Win=261376 Len=0

위 사진은 TCP stream index가 55인 TCP connection에서 발생한 HTTP 요청과 응답 중 하나를 가져온 것이다. 네이버 웹툰의 image content를 요청하는 과정에서 발생하는 메시지들로 9562번부터 10492번까지의 메시지를 요약한 것이다.
Client가 보낸 HTTP 요청 메시지의 크기는 MSS보다 작기 때문에 여러 segment로 나누어 보내지 않고 하나의 TCP segment를 통해 전송한다. 요청을 받은 server는 image content를 client에게 전송하는데, 요청하는 image file의 크기는 188745 byte로 MSS의 약 137배의 크기이다. 따라서 server는 client에게 137개의 TCP segment를 전송했다.

- ➔ No.9562 TCP segment의 Acknowledge number는 1078707로 순서번호 1078707를 갖는 TCP segment를 받기를 원한다는 것을 의미한다. No.9903 TCP segment는 server가 client에게 보내는 segment로 순서번호 1078707을 갖고 있는 것을 확인했다. No.9908 TCP segment는 client가 server에게 보내는 ACK segment이고 Acknowledge number는 1085637이다. 즉, client는 순서번호 1085636까지 전송 받았고 순서번호 1085637의 segment를 기다리고 있다는 것을 의미한다. 역시 server가 보내는 다음 segment의 순서번호가 1085637임을 확인 할 수 있다. ACK segment를 받은 server는 sender window를 sliding하며 data를 전송한다.

- ➔ **No.9983 TCP segment**는 flow control의 좋은 예로써 바로 위의 segment와 다른 것은 window size value 밖에 없다. 위의 경우 window size가 update 되기 전에 이미 충분한 window size를 확보하고 있었기 때문에 해당 메시지가 큰 의미가 없지만, client가 Zero window인 경우에는 server가 보낸 data를 저장할 수 없기 때문에 client 측에서 server에게 window size update 정보를 포함하는 메시지를 보내는 것이 중요하다.

- ➔ **No.10476 TCP segment**는 HTTP 응답의 마지막 segment로 해당 segment까지 총 137개의 segment가 합쳐져서 application layer의 process에 전달된다.

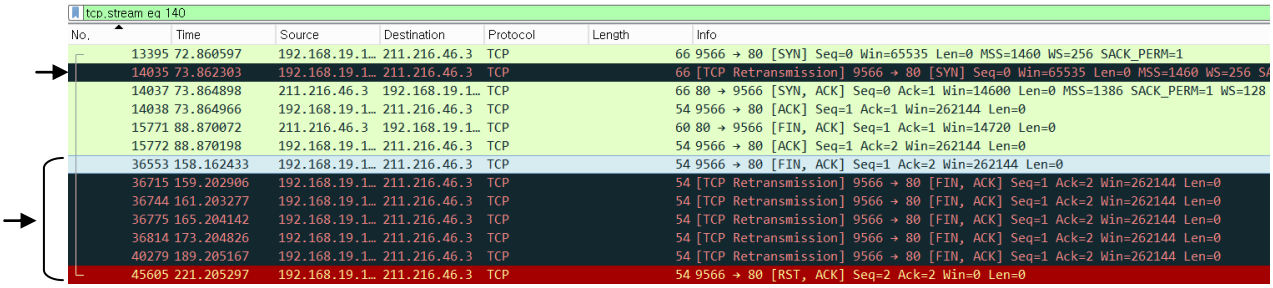
TCP는 순서에 맞지 않는 순서번호가 포착되는 즉시 그에 따른 대처를 할 것이고, 이러한 정교한 과정을 통해 신뢰적인 데이터 전송을 이뤄낸다.

(3) Abnormal segment flow cases

이번엔 duplication ACKs와 timeout에 의해 segment loss가 detect되었을 때 재전송하는 것과 같이, capture를 진행하며 발견된 특별한 상황들에 대해 살펴보도록 하겠다.

(3) – 1. Timeout에 의한 Retransmission

<TCP stream index = 140인 TCP connection>



위 사진은 TCP stream index가 140인 TCP connection에서 발생한 TCP 메시지를 나타낸다. TCP connection setup과 close 과정 모두 재전송이 발생한 특이 communication case로 연결이 설정되었는데도 특정 data 전송이 발생하지 않았다. Destination IP를 filter 값으로 추가한 결과 네이버 뉴스 메인 페이지의 이미지를 가져오기 위한 connection으로 추정되지만 요청이 발생하지는 않았다.

➔ **No. 14035 TCP segment**는 연결 설정을 위한 SYN segment이고 재전송 된 segment임을 알 수 있다. [TCP Retransmission]은 timeout에 의한 재전송을 의미하고, 이 경우는 SYN segment가 loss되거나 SYNACK segment가 loss되어 timeout이 발생한 것이다. 재전송이 발생하기까지 약 1초의 기간 동안 TCP client는 SYN_SENT상태로 SYNACK segment를 기다리고 있다.

➔ 다음으로 **No.36553부터 No.45605까지의 TCP segment**를 보면, 5번의 TCP Retransmission이 발생하였고 마지막에 RST flag가 1로 설정된 segment가 전송되었다. No.15771과 No.15772 TCP segment는 각각 server측에서 연결을 close하기 위해 보낸 FIN ACK segment와 이에 대한 client측의 ACK segment이다. 더 이상 보낼 data가 없는 client는 연결을 close하는데 동의하고 server에게 No.36553 FIN ACK segment를 보낸다.

주목할 점은 이후 5개의 TCP segment가 No.36553 FIN ACK segment에 대한 재전송 segment라는 것이다. 즉 같은 segment를 server측에 6개를 보낸 것인데, 이러한 case의 원인이 단순히 6개의 같은 segment가 모두 network상에서 loss되었기 때문이라고 보기엔 가능성이 매우 희박하다. 그렇다면 왜 5번의 재전송이 발생했으며, 마지막 RST segment의 의미는 무엇인지 알아보기 위해 구글 검색을 한 결과 어렵지 않게 그 이유를 찾을 수 있었다. 그 이유는 통신하는 network 자체가 문제가 있거나 server측에 문제가 있다는 것으로 해당

connection이 더 이상 유효하지 않기 때문이다. Endpoint간의 connection이 더 이상 유효하지 않기 때문에 연결을 종료하려는 client가 전송한 FIN ACK segment는 의미 없는 전송이고, 5번의 재전송에도 ACK segment가 오지 않을 때 client는 connection이 유효하지 않다고 판단하여 RST segment를 보낸다. 이러한 RST segment는 임의로 connection을 강제종료 하기 위해 전송하며, 이와 같은 과정은 **abortive close**이다.

또한 이 case에서 주목할 점은 **RTO timer management**에 대한 것이다(RFC 6289 - managing the RTO Timer 참고). segment의 RTO(Retransmission TimeOut) timer가 끝날 때까지 해당 segment에 대한 ACK segment가 도착하지 않는다면 TCP client는 RTO를 RTO * 2로 설정하고 같은 segment를 재전송한다. 또 다시 ACK segment가 도착하지 않는다면 RTO를 2배로 늘리고 재전송하게 된다. RTO timer는 이런 식으로 2의 지수 승 만큼 늘어난다. 이러한 timer management는 필수적이지만 그 방식에 있어서는 implementation dependent하다. RFC 6289에서는 위와 같은 방식을 recommand하고 있다. capture 결과를 보면 각각의 재전송이 발생하기까지 걸리는 시간이 1 -> 2 -> 4 -> 8-> 16 -> 32초로 증가하는 것을 확인 할 수 있다. 마지막으로 설정된 RTO timer 값인 32초가 지나도 ACK segment가 오지 않았고 이에 따라 RST segment를 보내어 connection을 종료시켰다.

위의 SYN segment와 FIN ACK segment 같은 TCP connection management segment들의 loss는 timer expires인 경우에만 발견할 수 있다는 것을 확인하였고, (3) – 2에서는 data 송수신을 위한 TCP segment의 loss detect에 대해 알아보겠다.

(3) – 2. Duplicate ACKs에 의한 Retransmission

<TCP stream index = 221인 TCP connection>

tcp.stream eq 221

No.	Time	Source	Destination	Protocol	Length	Info
34482	117.637350	23.67.53.65	192.168.19.1…	TCP	1440	80 → 9648 [ACK] Seq=2812593 Ack=182627 Win=178176 Len=1386 [TCP segment of a reassem…
34483	117.637389	192.168.19.1…	23.67.53.65	TCP	54	9648 → 80 [ACK] Seq=182627 Ack=2813979 Win=262144 Len=0
34486	117.770464	23.67.53.65	192.168.19.1…	TCP	1440	[TCP Previous segment not captured] 80 → 9648 [ACK] Seq=2815365 Ack=182627 Win=17817…
34487	117.770465	23.67.53.65	192.168.19.1…	TCP	1440	80 → 9648 [ACK] Seq=2816751 Ack=182627 Win=178176 Len=1386 [TCP segment of a reassem…
34488	117.770466	23.67.53.65	192.168.19.1…	TCP	1440	80 → 9648 [ACK] Seq=2818137 Ack=182627 Win=178176 Len=1386 [TCP segment of a reassem…
34489	117.770468	23.67.53.65	192.168.19.1…	TCP	1440	80 → 9648 [ACK] Seq=2819523 Ack=182627 Win=178176 Len=1386 [TCP segment of a reassem…
34490	117.770468	23.67.53.65	192.168.19.1…	TCP	430	80 → 9648 [PSH, ACK] Seq=2820909 Ack=182627 Win=178176 Len=376 [TCP segment of a rea…
34491	117.770469	23.67.53.65	192.168.19.1…	TCP	430	[TCP Retransmission] 80 → 9648 [PSH, ACK] Seq=2820909 Ack=182627 Win=178176 Len=376
34502	117.770536	192.168.19.1…	23.67.53.65	TCP	66	[TCP Dup ACK 34483#1] 9648 → 80 [ACK] Seq=182627 Ack=2813979 Win=262144 Len=0 SLE=28…
34503	117.770586	192.168.19.1…	23.67.53.65	TCP	66	[TCP Dup ACK 34483#2] 9648 → 80 [ACK] Seq=182627 Ack=2813979 Win=262144 Len=0 SLE=28…
34504	117.770610	192.168.19.1…	23.67.53.65	TCP	66	[TCP Dup ACK 34483#3] 9648 → 80 [ACK] Seq=182627 Ack=2813979 Win=262144 Len=0 SLE=28…
34505	117.770628	192.168.19.1…	23.67.53.65	TCP	66	[TCP Dup ACK 34483#4] 9648 → 80 [ACK] Seq=182627 Ack=2813979 Win=262144 Len=0 SLE=28…
34506	117.770644	192.168.19.1…	23.67.53.65	TCP	66	[TCP Dup ACK 34483#5] 9648 → 80 [ACK] Seq=182627 Ack=2813979 Win=262144 Len=0 SLE=28…
34507	117.770661	192.168.19.1…	23.67.53.65	TCP	74	[TCP Dup ACK 34483#6] 9648 → 80 [ACK] Seq=182627 Ack=2813979 Win=262144 Len=0 SLE=28…
34521	117.773708	23.67.53.65	192.168.19.1…	HTTP	1440	[TCP Fast Retransmission] HTTP/1.1 200 OK (JPEG JFIF image)
34522	117.773805	192.168.19.1…	23.67.53.65	TCP	54	9648 → 80 [ACK] Seq=182627 Ack=2821285 Win=262144 Len=0

위 사진은 TCP stream index가 221인 TCP connection에서 발생한 TCP 메시지를 나타낸다. 해당 connection은 쇼핑몰 상품 페이지로 이동했을 때 상품에 대한 image를 요청하기 위한 것으로 위 메시지들은 그 일부분이다.

No.34482와 No.34483 TCP segment는 server에서 보내는 data가 포함된 segment와 그에 대한 ACK segment이다. server에서 순서번호 2812593번의 TCP segment를 보냈고, 그에 대한 응답으로 client가 ACK number을 2813979로 설정한 ACK segment를 보냈다. 그 다음 No.34486 TCP segment는 순서번호 2815365를 가진 segment로 client가 기대하고 있는 순서번호와는 다르기 때

문에 [TCP Previous segment not captured]라는 문구를 확인할 수 있다. 이후 No.34487부터 No.34491까지 5개의 TCP segment 또한 기대하고 있는 순서번호의 segment가 아니다. 즉, 현재 TCP client는 순서번호 2813979를 가진 segment를 기다리고 있는데 해당 순서번호를 갖고 있지 않은 segment들이 6개 도착한 상황이다. 이에 따른 대처로 TCP는 duplicate ACKs를 보내도록 되어있는데, 이후 No.34502부터 No.34507까지 6개의 TCP segment가 duplicate ACK segment이다. 이런 duplicate ACK segment를 보내며 client는 server에게 순서번호 2813979의 segment를 기다리고 있다고 알린다. 3개 이상의 duplication ACKs를 받은 server는 해당 순서번호의 segment를 재전송하고 TCP client는 모든 data를 받았으므로 application layer process에게 재전송된 segment를 포함한 모든 segment들을 합쳐서 보낸다. 이 case는 순서번호 2813979의 TCP segment가 loss되었기 때문에 발생한 것이고, duplication ACKs를 통해 loss detect를 하고 재전송을 하는 과정을 **TCP fast retransmission**이라고 한다.

Loss가 발생한 시점부터 재전송 시점까지 불과 0.003초 밖에 걸리지 않았고, timeout에 의한 재전송보다 훨씬 빠르게 대처한다는 것을 확인할 수 있다. TCP connection에서 발생하는 모든 loss를 timeout에 의해 발견해야 한다면 신뢰적인 데이터 전송은 만족할 수 있으나 느린 서비스를 제공하게 될 것이다.

4. 새로 알게 된 사실

RST flag가 TCP connection을 강제로 close하기 위해 쓰일 수 있다는 것을 알았다. Network에서 router queue가 꽉 차서 TCP segment가 loss되는 상황을 예제로써 주로 접하였기 때문에 다른 원인에 대해 크게 생각해보지 않았는데, 유효하지 않은 TCP connection에 의해 loss가 발생하는 case를 직접 확인해 볼 수 있었다.

5. 관찰 후 소감

TCP가 제공하는 가장 중요한 기능인 신뢰적인 데이터 전송에 대해 수업시간을 통해 많이 들었지만, 마치 과거에 처음 제시되었을 때는 혁명이었던 아이디어들이 현재는 당연시 되고 있는 것처럼 크게 와닿지는 않았다. 하지만 이번 기회를 통해 TCP가 상당히 정교하게 그 기능을 수행하고 있다는 것을 느꼈고, 실제로 network에서 발생하는 문제는 많고 TCP는 그 문제들을 잘 처리한다는 것을 눈으로 확인할 수 있었다.

6. 참고

https://www.youtube.com/user/packetpioneer/videos - Top 10 wireshark filter 영상이 있는 유튜브 채널로 wireshark를 통해 TCP 분석을 하는 영상 여러 개가 업로드 되어 있다.