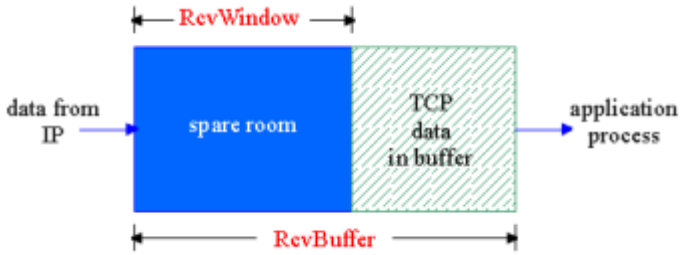
20230406_TIL - Transmission Layer

TCP FLOW CONTROL

- TCP Connection의 리시버는 receive buffer라는걸 갖고 있다.



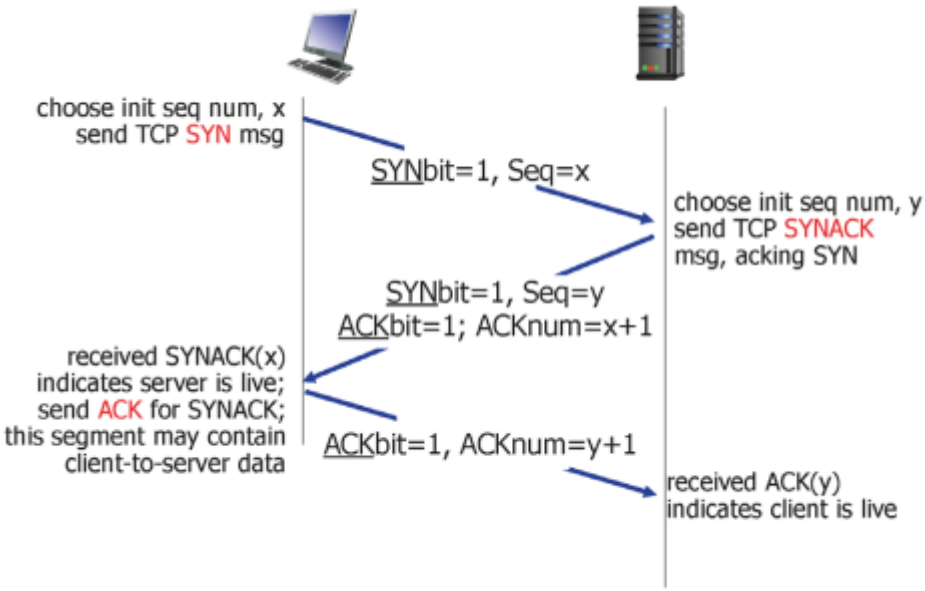
- 어플리케이션 프로세스가 버퍼로부터 읽어오는데 느릴 수 있다(may라는 표현으로 보아 느린가봄)
- 그래서 sender 역시 receiver의 buffer를 넘어서지 않게, 너무 빠르거나 많이 보내지 않는다.
- speed-matching service를 제공하는데, 이는 어플리케이션 계층의 drain rate에 맞춘다는 말이다.
- 즉 sender의 flow control은 receiver의 버퍼의 크기에 맞추는데, 이는 결국 어플리케이션 계층의 drain rate에 의해 결정된다.
- `spare room`: receive window와 같은 크기, 즉 `마지막으로 받은 데이터의 바이트 - 마지막으로 읽은 바이트`의 크기를 갖는다.

- receiver는 spare room의 존재를 window의 크기를 `segment`에 넣어줌으로써 알린다.
- sender는 또 receiver의 버퍼가 터지지 않도록 ack를 받지 못한 데이터의 수를 제한한다(???)

TCP CONNECTION MANAGEMENT

- TCP의 SENDER와 RECEIVER는 `SEGMENT`를 교환하기 전에 연결을 확보한다. → SEQ#, FLOW CONTROL 정보 등
- 클라이언트 측에서 소켓을 열면서 통신이 시작됨(서버는 기다리는 상태)

3-WAY HANDSHAKE



1. 클라이언트 호스트가 TCP SYN `SEGMENT`를 서버에 보냄
    a. SEQ#로 구분
    b. 데이터 없음
2. 서버 호스트는 `SYN`을 받고, `SYN ACK` 를 보낸다.
    a. 서버가 버퍼를 할당

b. 서버의 초기 SEQ#로 보냄

3. 클라이언트가 `SYN ACK`를 받고, 데이터를 담은 세그먼트를 보내면서 응답한다.

TCP 연결 닫기

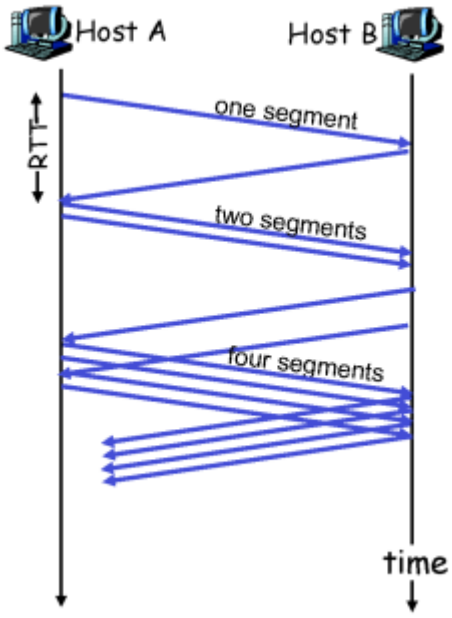
- 클라이언트가 연결 종료

1. `TCP FIN` `SEGMENT`를 서버에 전달
2. 서버가 `FIN`을 받고 데이터를 보낸 다음에 `ACK` 전달, 연결 닫고 `FIN`을 보낸다.
3. 클라이언트가 서버가 보내는 `FIN`을 받고 `ACK` 전송, 타이머 터짐
4. 서버는 `ACK`를 받으면 연결 종료. 클라이언트는 타이머 끝나면 연결 닫음

TCP CONGESTION CONTROL

- keyword: congestion window (aka CongWin)

SLOW START



- bottleneck의 너비를 알지 못함

- 그래서 0부터 시작해서 사이즈를 빠르게 늘려나감
- 초기 window size = 1mss (maximum segment size)
- 첫 유실이 발생할 때까지 기하급수적으로 증가

ADDICTIVE INCREASE

- 용량(threshold)에 가까워짐
- 천천히 늘리기 시작
- 보내는 양이 linear하게 증가
  - 매 rtt (rtt: 데이터 송신 후 수신까지 시간)마다 1mss씩 증가

MULTIPLICATIVE DECREASE

- 네트워크에 과부하가 생기면 다른 사용자가 영향이 있음
- 패킷 드랍이 발생하면 congwin을 절반으로 줄여줌

- sender의 전송속도: 대략 congestion window / rtt
  - 네트워크의 혼잡도에 따라 결정
- `congwin ≥ LastByteSent-LastByteAcked`
- timeout 혹은 같은 `ack` 4번 → 1 loss event
  - loss event 발생시 threshold는 congestion window의 절반으로
  - timeout이 발생하면
- loss event가 한번 일어나면 tcp sender가 rate를 감소시킨다
  - 어떻게? → slow start로 회귀, AIMD, time event 발생 후 조치(threshold, congwin 조절)