2018.2.

컴퓨터 네트워킹

TCP(Transmission Control Protocol) ②

III. 트랜스포트 계층

5. 연결지향형 트랜스포트: TCP

- (5) 흐름 제어
- (6) 연결 관리

6. 혼잡 제어의 원리

- (2) 혼잡 제어에 대한 접근법

7. TCP 혼잡 제어

JEJU NATIONAL UNIVERSITY

김예찬 (think.computer@jejunu.ac.kr)

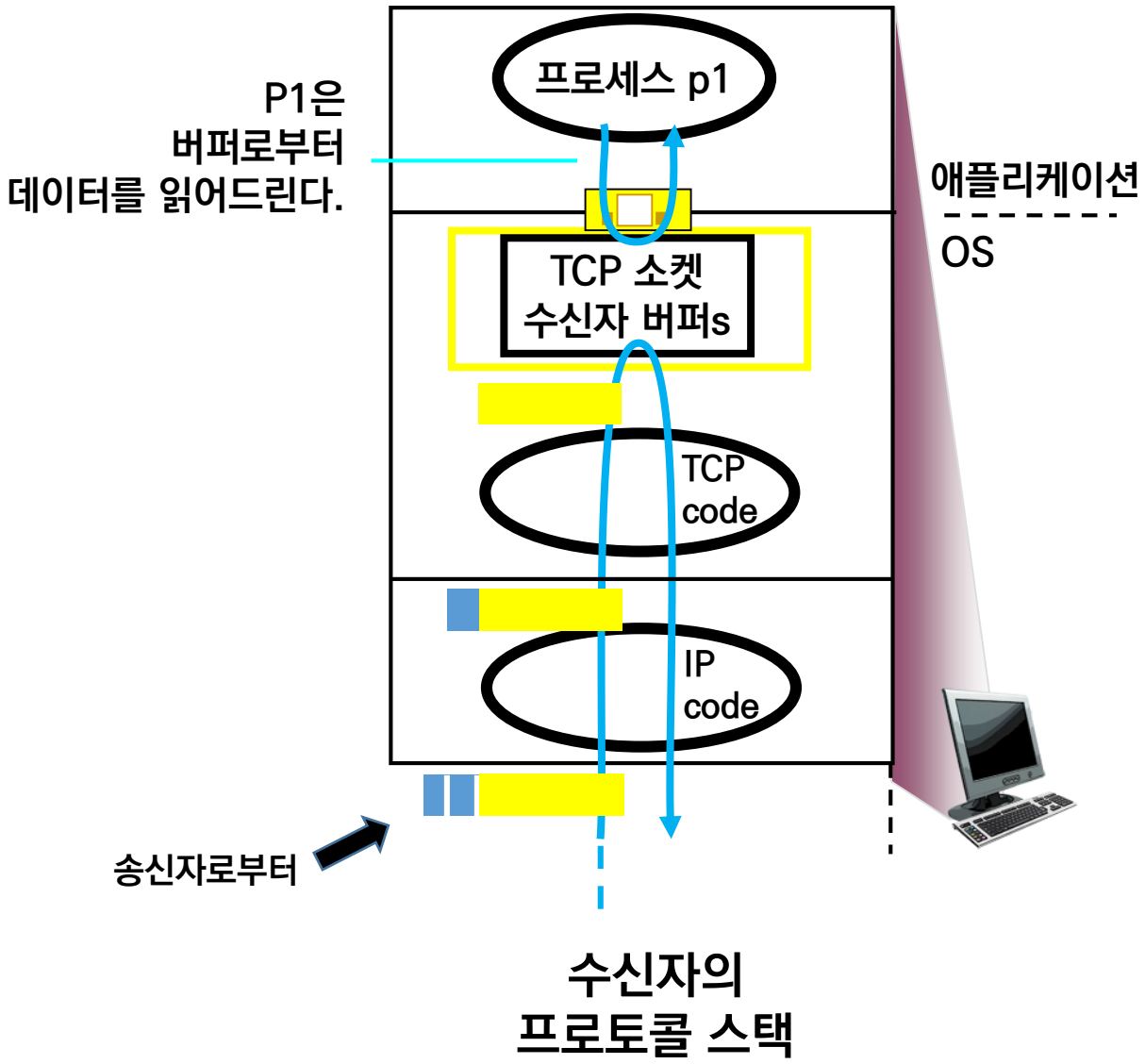
Goal :

- 흐름 제어의 필요성을 설명할 수 있다.
- 흐름 제어를 위한 수신자의 행동이 무엇인지 설명할 수 있다.
- 흐름 제어에서 송신자가 수신자의 버퍼가 다시 가용 상태인지를 확인하기 위해 윈도우 탐침 세그먼트를 사용한다는 것을 안다.

출발지 PORT#	목적지 PORT#
순서번호 필드	
확인응답번호 필드	
헤더 길이 / not used / U A P R S F	수신 윈도우
체크섬 필드	URG 데이터 포인터
옵션 필드 (선택적 / 가변적 길이) & Padding 필드	
페이로드(Message)	

〈TCP 세그먼트〉

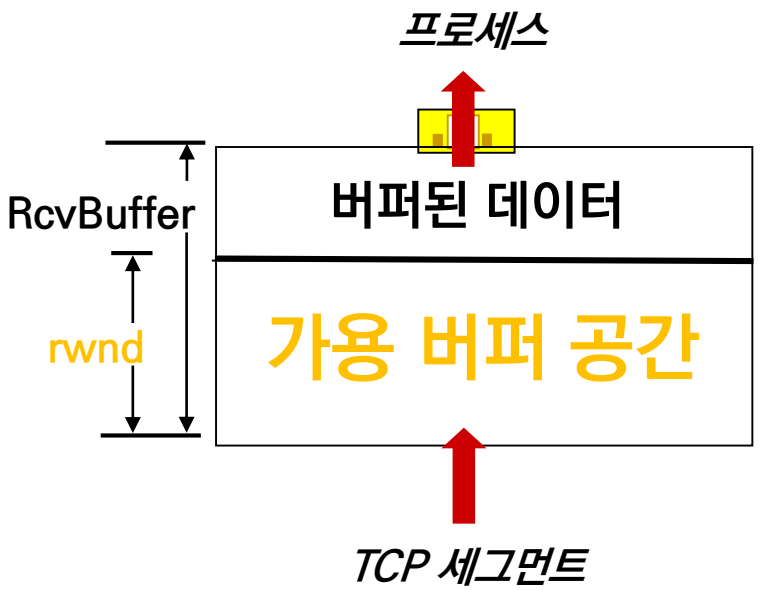
흐름 제어의 필요성



- P1은 데이터가 도달한 시점에 데이터를 읽어야 할 필요는 없다.
  - P1은 다른 작업으로 바쁠 경우, 버퍼엔 데이터가 점점 쌓인다.
  - 이때 만일 송신자가 계속 데이터를 전송할 경우,
    수신자의 버퍼는 오버플로우 (Overflow)가 발생할 수 있다. v

- 따라서 TCP는 이를 방지하기 위해 수신자 프로세스가 읽는 속도와 송신자의 전송 속도를 같게 한다.

수신 측 흐름 제어 action



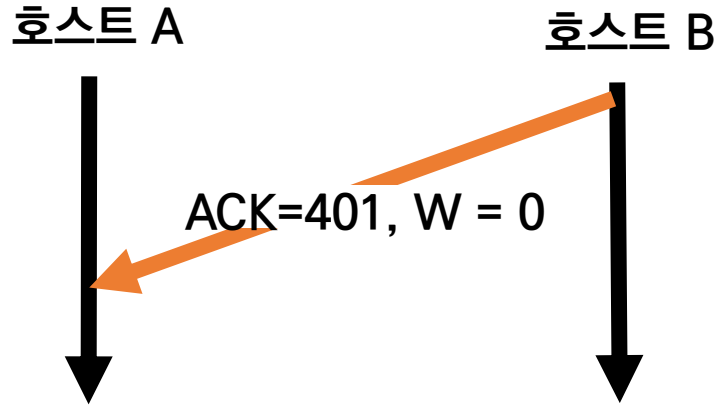
수신 측 버퍼링

- 수신 윈도우 크기 rwnd =
  RcvBuffer – [LastByteRcvd – LastByteRead]
  (rwnd는 시간에 따라 변하므로 동적이다.)

- 수신자는 수신할 때마다 송신자에게 rwnd값을 알려준다. (TCP 세그먼트에 담아서)
  - 예) 송신자가 수신자에게 100byte씩 보내고, 수신자는 계속 버퍼링한다고 할 때…
    - 초기 rwnd = RcvBuffer
    - 2nd rwnd = RcvBuffer – 100byte
    - 3rd rwnd = RcvBuffer – 200byte

  ⇒ 송신 측에서는 이를 통해 수신 버퍼 오버플로우를 막을 수 있다.

앞서 흐름 제어 방법의 문제점



- 호스트 B의 수신 버퍼가 가득 찼을 때의 rwnd?
  - rwnd = 0;

- 호스트 B가 호스트 A에게 rwnd = 0이라고 알린 후, 시간이 지나 호스트 B에 버퍼가 비워졌다고 하자. 호스트 A는 언제 새로운 데이터를 보낼 수 있는가?
  - 보낼 수 없다. 호스트 A는 호스트 B의 수신 버퍼 상태를 모르기 때문이다.

- 따라서, TCP 명세서는 호스트 A가 호스트 B의 수신 윈도우가 0일 때, 1byte 데이터로 세그먼트를 계속 전송할 것을 권고한다.

윈도우 탐침 세그먼트 (W. Probe Seg.) : 수신자에 의해 긍정 확인응답 받을 때까지 계속 이것을 전송하여, 수신자 버퍼의 상태를 조사(Probe)한다.

• TCP 연결 관리

Goal :

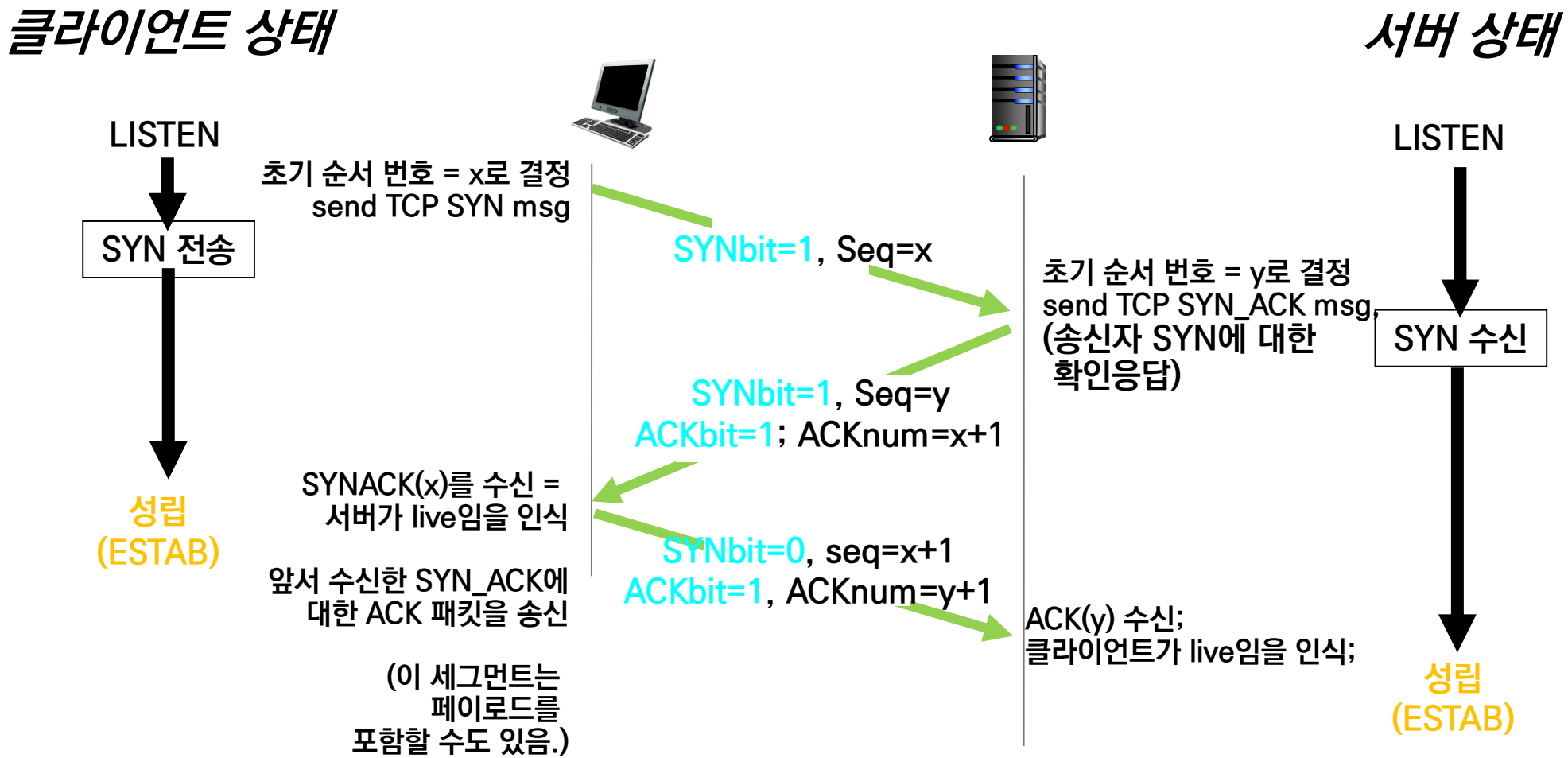
- TCP 연결이 어떻게 설정되고 해제되는지를 설명할 수 있다.

출발지 PORT#	목적지 PORT#
순서번호 필드	
확인응답번호 필드	
헤더 길이 / not used / U A P R S F	수신 윈도우
체크섬 필드	URG 데이터 포인터
옵션 필드 (선택적 / 가변적 길이) & Padding 필드	
페이로드(Message)	

〈TCP 세그먼트〉

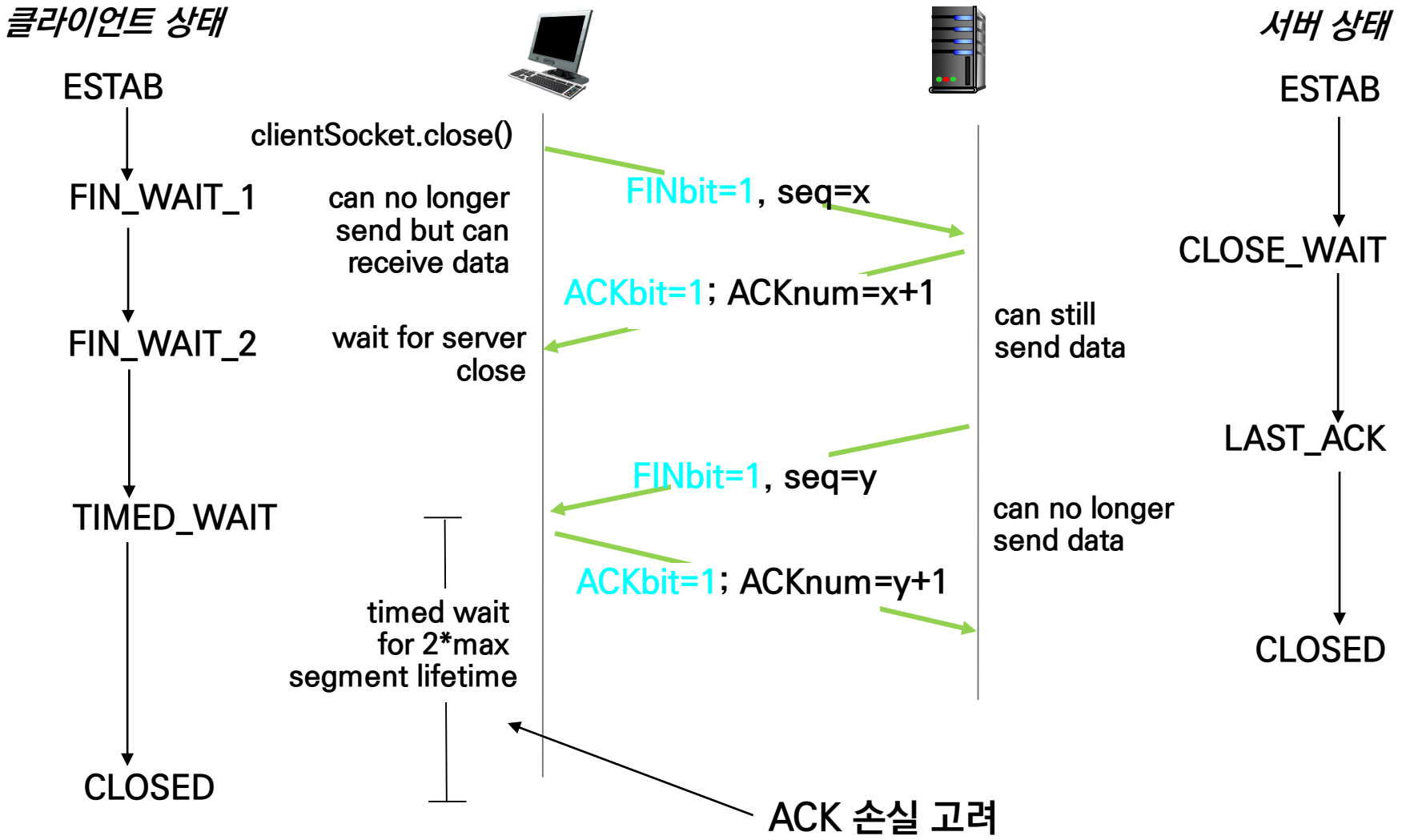
(초기) 연결 설정 시나리오

3-way Handshaking



연결 종료(FIN) 시나리오

클라이언트가 종료를 희망한다고 가정.



- 앞서 그림에서 왜 서버는 클라이언트에게 FIN 세그먼트를 보내는 걸까?
  - 이는 서버가 더 이상 보낼 데이터가 없으므로 연결을 종료해도 좋다는 의미이다.
  - 만일, 서버가 아직 클라이언트에게 보낼 데이터가 있었다면, 그 데이터를 보내고 연결을 종료한다.

만약 소켓과 관계 없는 세그먼트를 수신한다면?

- TCP
  - 만약 관계없는 목적지 포트 번호를 가진 TCP SYN 패킷 수신 시…
  - 수신 호스트는 출발지에 특별한 RST 세그먼트를 보낸다. (RST bit = 1)
    - 이는 출발지에게 그 세그먼트에 대한 소켓을 가지고 있지 않으니, 재전송하지 말 것을 뜻함.

- UDP
  - ICMP라는 특별한 데이터 그램을 전송한다.

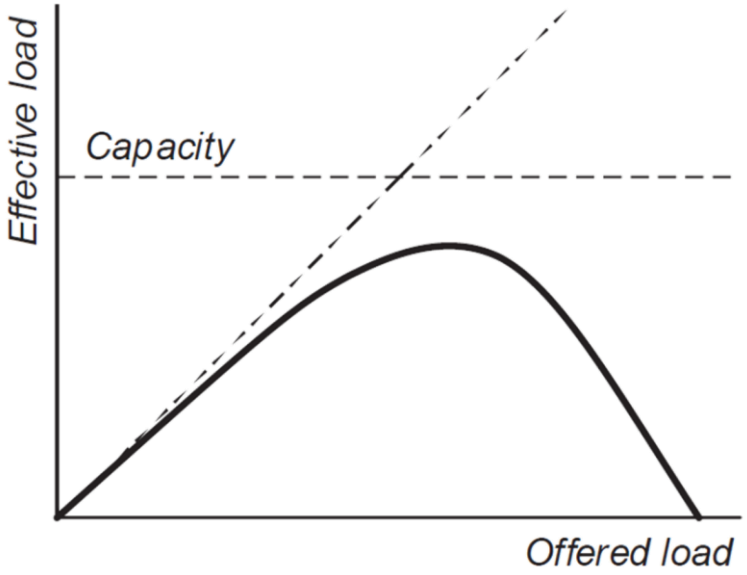
Goal :

- Congestion Collapse가 무엇인지 안다.
- 혼잡 제어와 흐름 제어를 비교 설명할 수 있다.
- AIMD 혼잡 제어 접근법을 설명할 수 있다.

Congestion Collapse

- 부하(트래픽)가 많아지면,
  실제 네트워크의 실 전송 용량은 0이 된다.
  - 쉽게 “연휴 날 꽉 막힌 고속도로”를 떠올려라.

- 혼잡(Congestion)의 정의
  - 너무 많은 노드들이 너무 많은 데이터를
    너무 빨리 전송하여 네트워크가 이를 처리하기
    곤란한 상태.
  - 비교
    - 흐름 제어에서는 End-to-End(Point-to-Point)
      간의 이슈.
    - 혼잡 제어에서는 Global한 이슈.



혼잡 제어에 대한 접근법

- End-to-end 접근법
  - Congestion은 각 노드(point)가 근시안적으로 판단하여 해결한다.
  - 일반적으로 TCP 등은 이 방식을 취한다.

- Network-assisted 접근법
  - 라우터 등이 각 종단 시스템(end system)에게 피드백을 제공한다.
  - 비용 측면에서 비효율적이다.
  - 명시적 혼잡 제어 관련.

AIMD 접근법

Additive Increase

Multiplicative Decrease

: 혼잡 윈도우(cwnd)의 크기를 AIMD 식으로 증감시킨다.

if 혼잡 윈도우 크기 증가 : 전송률을 높인다는 의미.

elif 혼잡 윈도우 크기 감소 : 전송률을 낮춘다는 의미.

참고 :

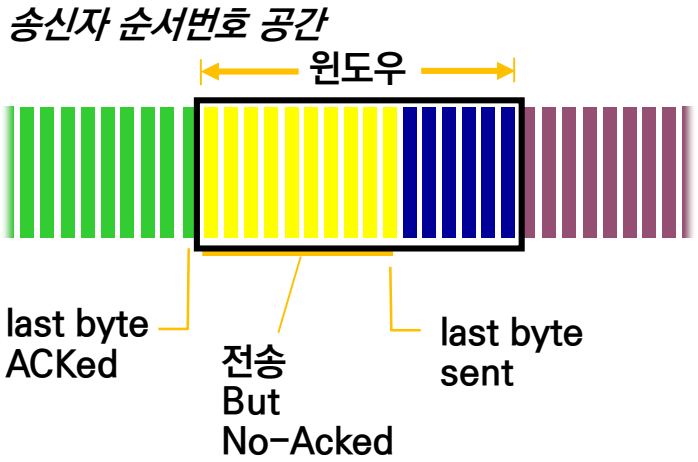
앞서 loss는 3 duplicate ACK 혹은 timeout으로 추정한다고 강의한 바 있음.

- 혼잡 윈도우(cwnd) :

  송신자가 네트워크로 트래픽을 전송할 수 있는 비율을 제한하기 위해 사용되는 추가적인 변수.
  (시간에 따라 크기가 동적.)

  호스트가 Congestion을 일으키지 않는 한도에서 최대한 한꺼번에 확인응답을 받지 않고 보낼 수 있는 데이터의 양.

AIMD 접근법



관련 공식

- 송신 윈도우 크기 ≥ LastByteSent − LastByteAcked
- LastByteSent − LastByteAcked ≤ min(cwnd, rwnd)
- 송신 윈도우 크기 ≤ min(cwnd, rwnd)

주목할 사실

TCP에서는 AIMD 접근 방식으로
혼잡 제어를 수행하는 것이 일반적이다.

Goal :

- TCP 혼잡 제어 알고리즘의 기본 원리를 설명할 수 있다.
- TCP 초기 버전인 Tahoe와 새로운 버전인 Reno를 비교 설명할 수 있다.
- Tahoe, Reno 그래프를 그려 혼잡 윈도우 크기 변화를 설명할 수 있다.

Quiz.

- O/X
  - 긍정 확인 응답은 혼잡 윈도우 크기를 증가시키는 촉매제 역할을 한다.
  - 확인 응답이 높은 속도로 도착하면 혼잡 윈도우는 더욱 빨리 증가한다.
  - TCP는 4개의 확인 응답 수신으로 패킷 손실을 감지할 수 있다.
  - 손실된 세그먼트의 존재는 네트워크의 혼잡을 의미할 수 있다.
  - AIMD는 묵시적 혼잡 제어 기술이다.

- 정답은 모두 O.
  - 참고
    - TCP는 확인 응답을 혼잡 윈도우 크기의 증가를 유발하는 트리거(Trigger) 혹은 클록(Clock)으로 사용하므로, TCP는 자체 클로킹(Self-clocking)이라고 함.
    - 묵시적 혼잡 제어 : Time-out, 3-duplicate ACK를 Loss로 간주, 한편 Loss의 원인을 혼잡(Congestion)으로 암묵적으로 유추.

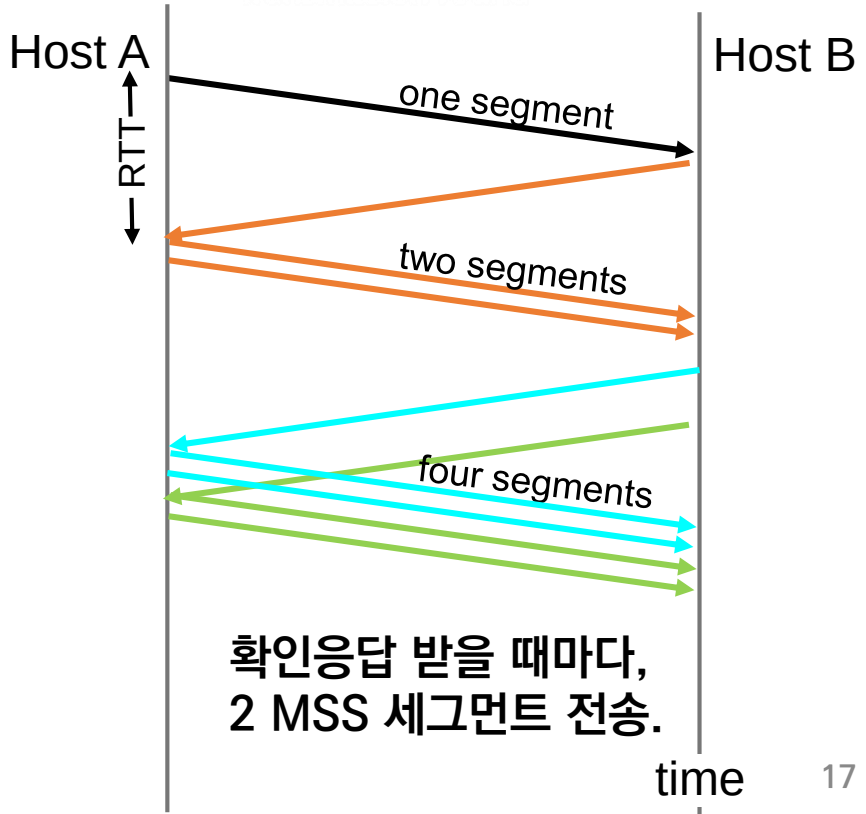
TCP Tahoe (1988)

- 주요 기능
  - 슬로 스타트(Slow Start)

```
cwnd = 1;
while ( 손실이 발생하지 않은 동안 ) {
        cwnd개의 packet 전송();
        if all ACKed :
                cwnd = cwnd * 2;
}

if '손실(Loss)' 발생 {
        cwnd = 1;     ssthresh = cwnd/2;
}

# ssthresh는 slow start threshold라는 뜻.
```



TCP Tahoe (1988)

- 혼잡 회피(Congestion Avoid)

  (한 차례 혼잡을 겪은 뒤, …)
  만약 cwnd가 지수적으로 증가하다
  cwnd 〉= ssthresh가 되면,
  cwnd값을 1씩 증가시킨다.
  → 이를 통해 혼잡을 회피하고자 시도함.

  그렇다면, 혼잡 회피에 따른
  선형적 cwnd 증가는 언제 끝날 것인가?
  → 또 다시 혼잡에 직면했을 때!
  (loss = 3-중복 ACK or timeout)
  (다시 slow-start 상태로 전이)

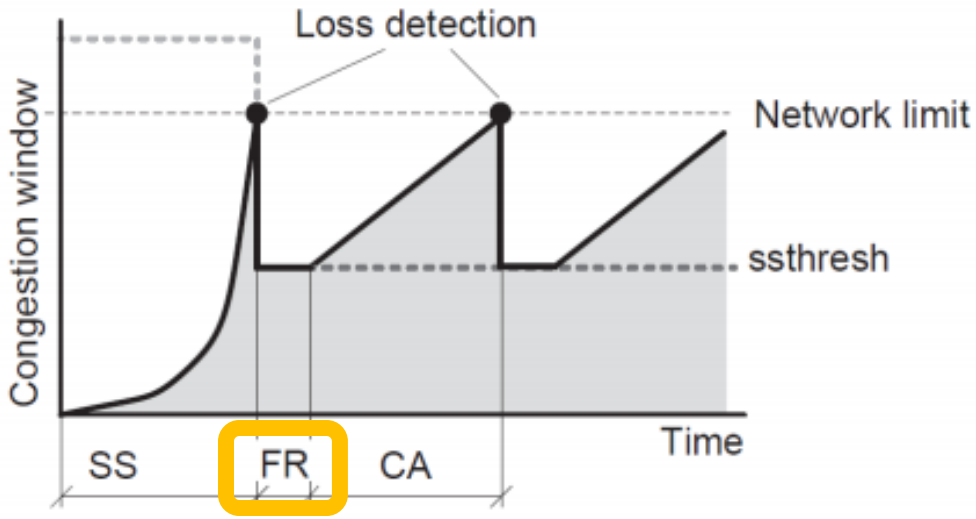
Tahoe의 한계 ★
3-duplicate ACK와 Timeout의 경우를 구분하지 않고
항상 cwnd = 1로 설정

Timeout

3-duplicate ACK는 한 패킷을 제외하고, 나머지 패킷들은 잘 전달되고 있다는 뜻.
⇒ Timeout보다 네트워크 혼잡이 덜 심하다는 의미.

TCP Reno (1990)

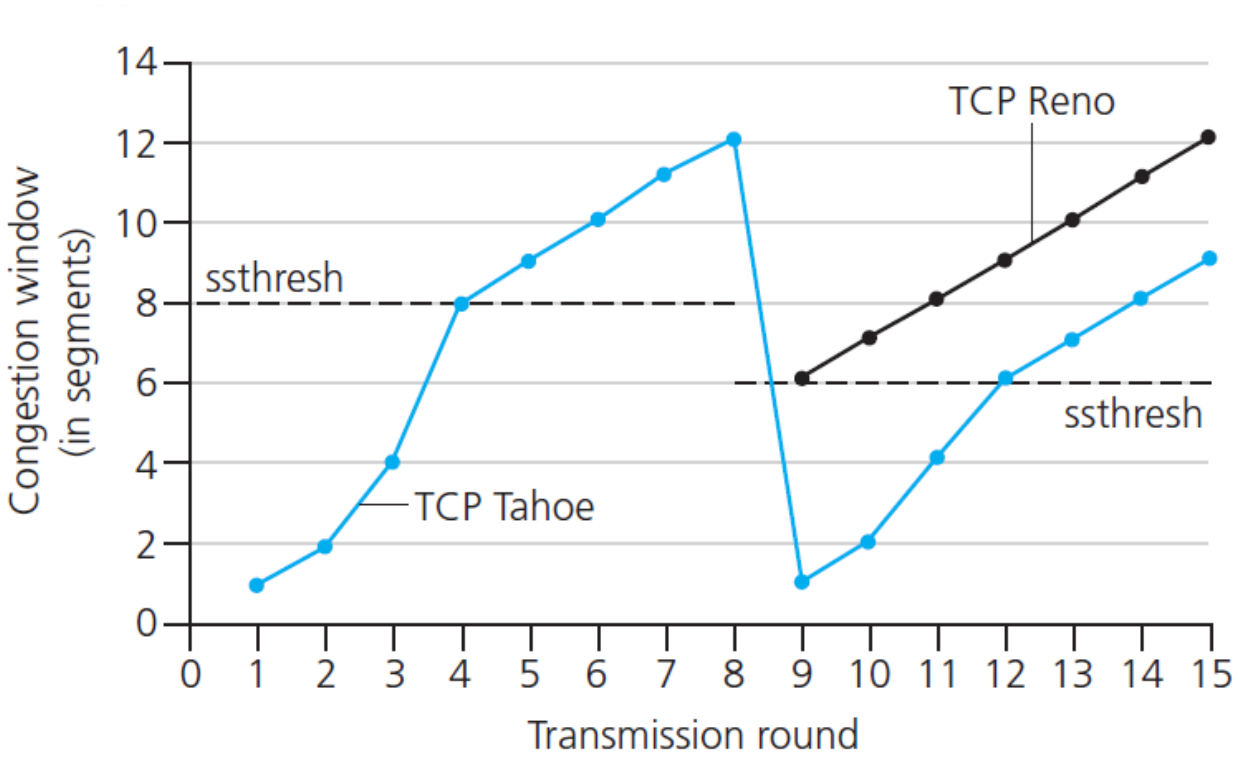
- Tahoe + ‘fast recovery (빠른 회복)’
  - 3-duplicate ACK에 따른 fast retransmit을 해야 할 경우, cwnd를 반으로만 줄이자.
    즉, cwnd = 1; (X) cwnd = cwnd / 2;



  - FR(Fast Recovery) 기간 :
    non-duplicate ACK를 수신할 때까지 cwnd 크기 유지.

단 FR 기간 동안 패킷 손실이 더 발견될 경우,
FR 기간 동안이더라도 cwnd 크기를 반씩 계속 줄여 나감.
⇨ TCP NewRENO(1999)에서는 cwnd 크기를 계속 같은 크기로 유지시키도록 알고리즘 수정!

TCP Tahoe vs Reno



- 슬로 스타트, 혼잡 회피
  TCP 혼잡 제어
  구현에 있어 필수 요소.

- 빠른 회복
  필수 요소는 아니지만,
  권고 사항.

- Reno의 경우, Tahoe와 달리, '혼잡 회피' 단계에서 cwnd가 항상 1씩 선형 증가함.