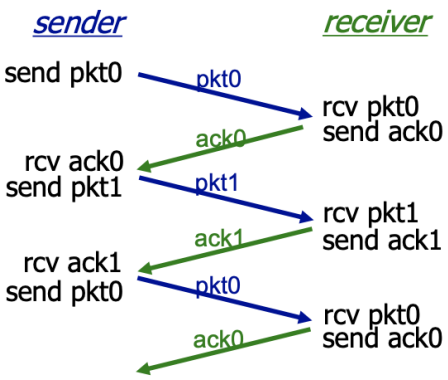
5일차

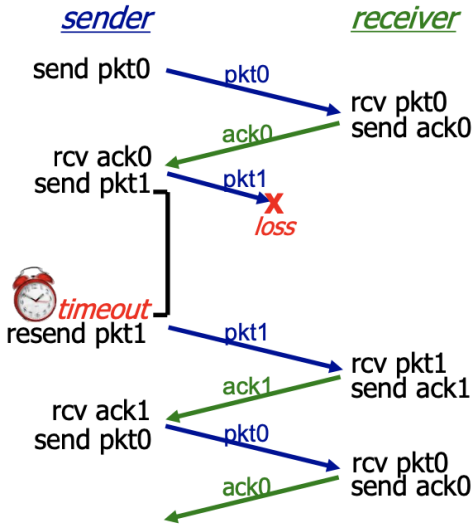
Principles of Reliable Data Transfer

- Application 계층에서 보기에는 transport 계층이 reliable channel 을 보내주는 것으로 보이지만 그 아래에서 여러과정을 통해서 reliable channel을 만들어주는 것이다. 이를 살펴보자!

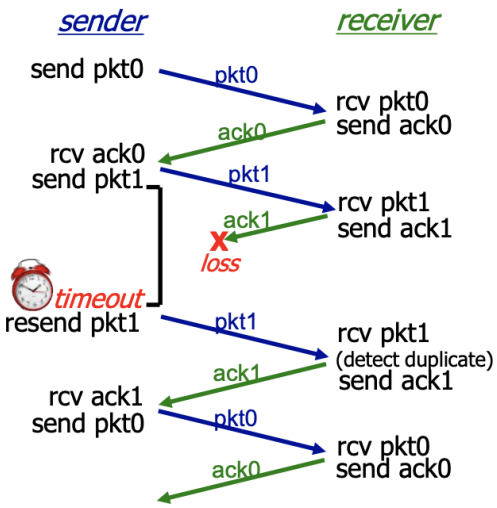
Stop-and-wait



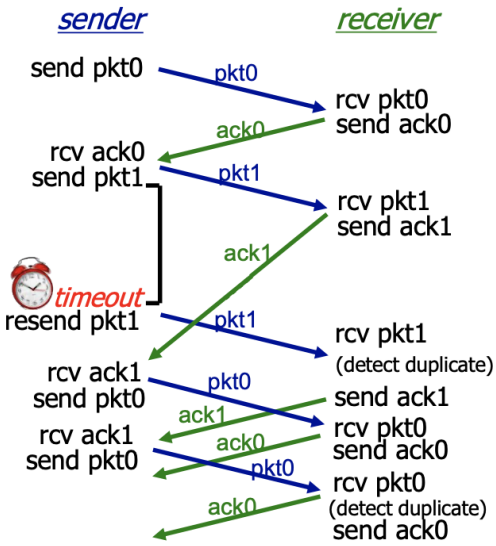
(a) no loss



(b) packet loss



(c) ACK loss



(d) premature timeout/ delayed ACK

기본적으로 sender는 pkt를 보내고 receiver는 pkt을 받고 ack(확인응답)를 다시 보내준다.

a. no loss : sender가 pkt을 보내고 receiver는 pkt을 받는다. pkt 받은 것을 다시 sender에게 알리기 위해 ack 를 보내준다.

b. packet loss : sender가 보낸 pk1이 loss 되었다. 따라서 receiver는 아무것도 보내지 않고 sender는 pkt1의 timeout이 되어서 pkt1 이 잘 도착하지 않았음을 인지하고 resend 한다.

c. ACK loss : 이번에는 sender에서 receiver로 pkt는 전달이 잘 되었지만 receiver에서 보내주는 ack 가 loss 되었다. 이때도 마찬가지로 sender에서 ack1를 수신하지 못했기 때문에 다시 한번 pkt1 을 보내준다.

d. premature timeout / delayed ACK : 송수신은 잘 되었으나 timeout 시간이 너무 짧거나 ack 가 delay 되어서 보내져서 sender 가 resend 한 이후에 ack가 도착하는 경우이다.

Piplelined protocols 2가지

→ 기존 Stop-and-wait 방식은 하나씩만 보내고 받기 때문에 이를 병렬적으로 해결하는 방법이 나온다.

1. Go-back-N
    a. sender 는 N개의 unACKed 패킷을 파이프라인에 가질 수 있다.

b. receiver는 cumulative ACK만 보낸다.
   cumulative ACK : 하나의 ACK 만 가지고 그 sequence num 이전 모든 packet 은 모두 받았다고 가정.

c. sender 는 가장 오래된 unACKed 패킷에 대한 timer 를 유지하고 있다. 이를 바탕으로 timeout 이 발생하면 unACKed packet들을 다시 보낸다.

2. Selective Repeat

a. sender 는 N개의 unACKed 패킷을 파이프라인에 가질 수 있다.

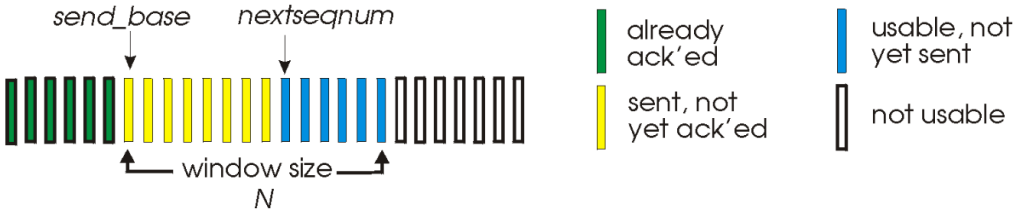
b. recevier는 각각의 ack 에 대한 packet 을 전달한다.

c. Go-back-N 방식과 다르게 timeout 이 발생하면 해당 unACKed packet 만 다시 보낸다.

Go-back-N

1. sender

a. window : sender 가 보낼 수 있는 양 (보내고 받는게 완료되면 하나씩 넘어감)

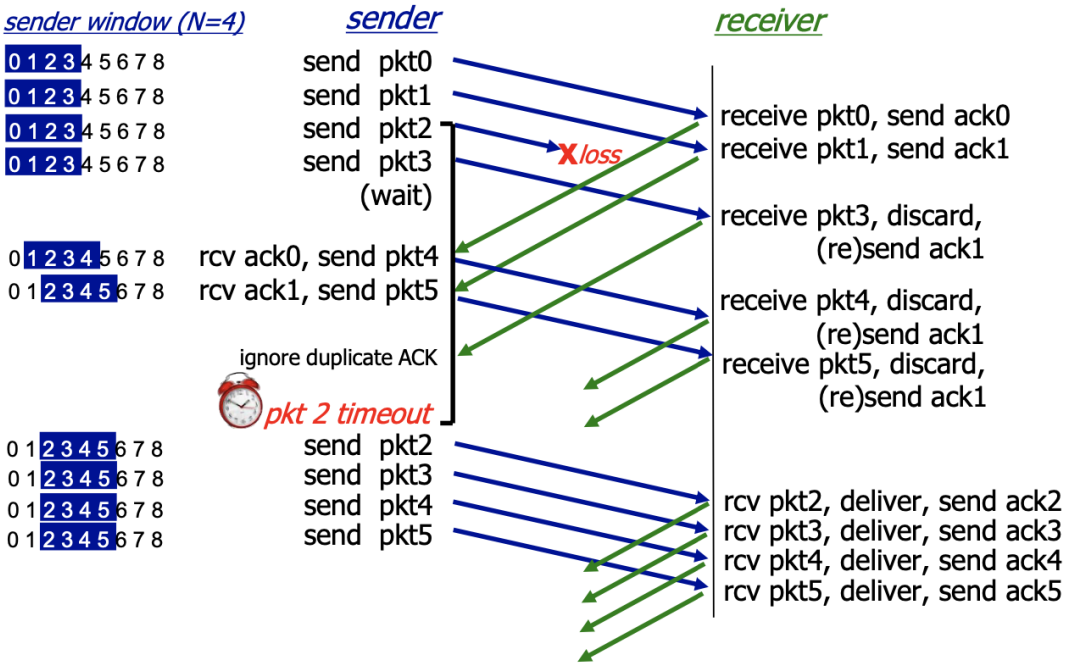


2. receiver

a. in-order로 보냄

b. out-of-order packet 는 discard 함. (buffer 안함)

3. 예시



sender 는 pkt 0,1,2,3을 보내는데 pkt2 에서 loss 발생

pkt3을 받은건 discard, ack1을 다시보낸다. (loss되기 직전 ack) ack0, ack1이 sender 에 다시 도착해서 sender 는 다음 pkt4,5 를 보낸다. 하지만 receiver는 이를 모두 discard 하고 ack1를 다시 보낸다.

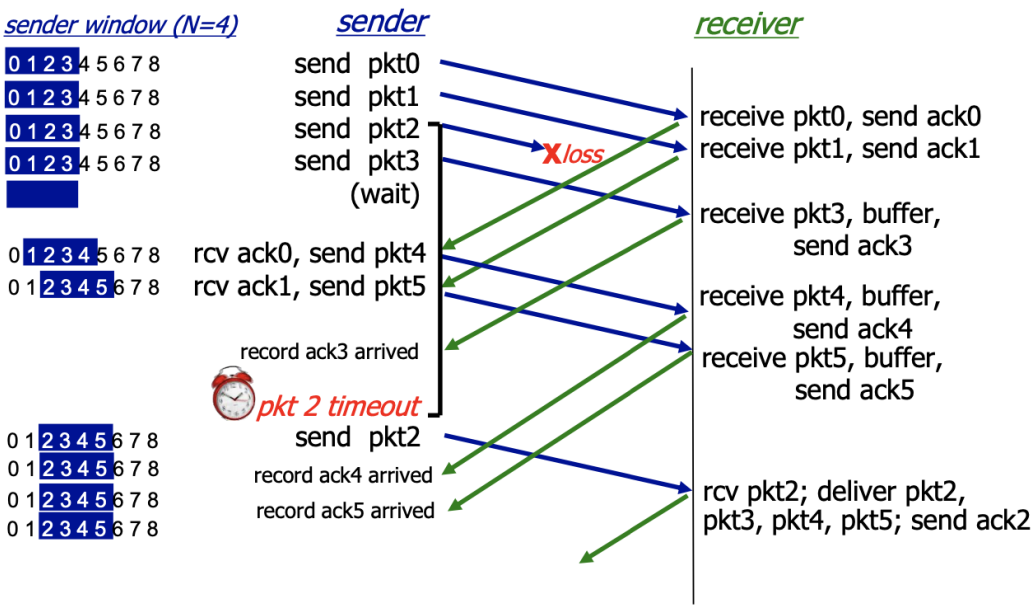
pk2의 timout이 지나고 sender 는 pk2부터 다시 보낸다.

→ 특징 : 순서가 안 맞으면 discard 해버림

Selective Repeat

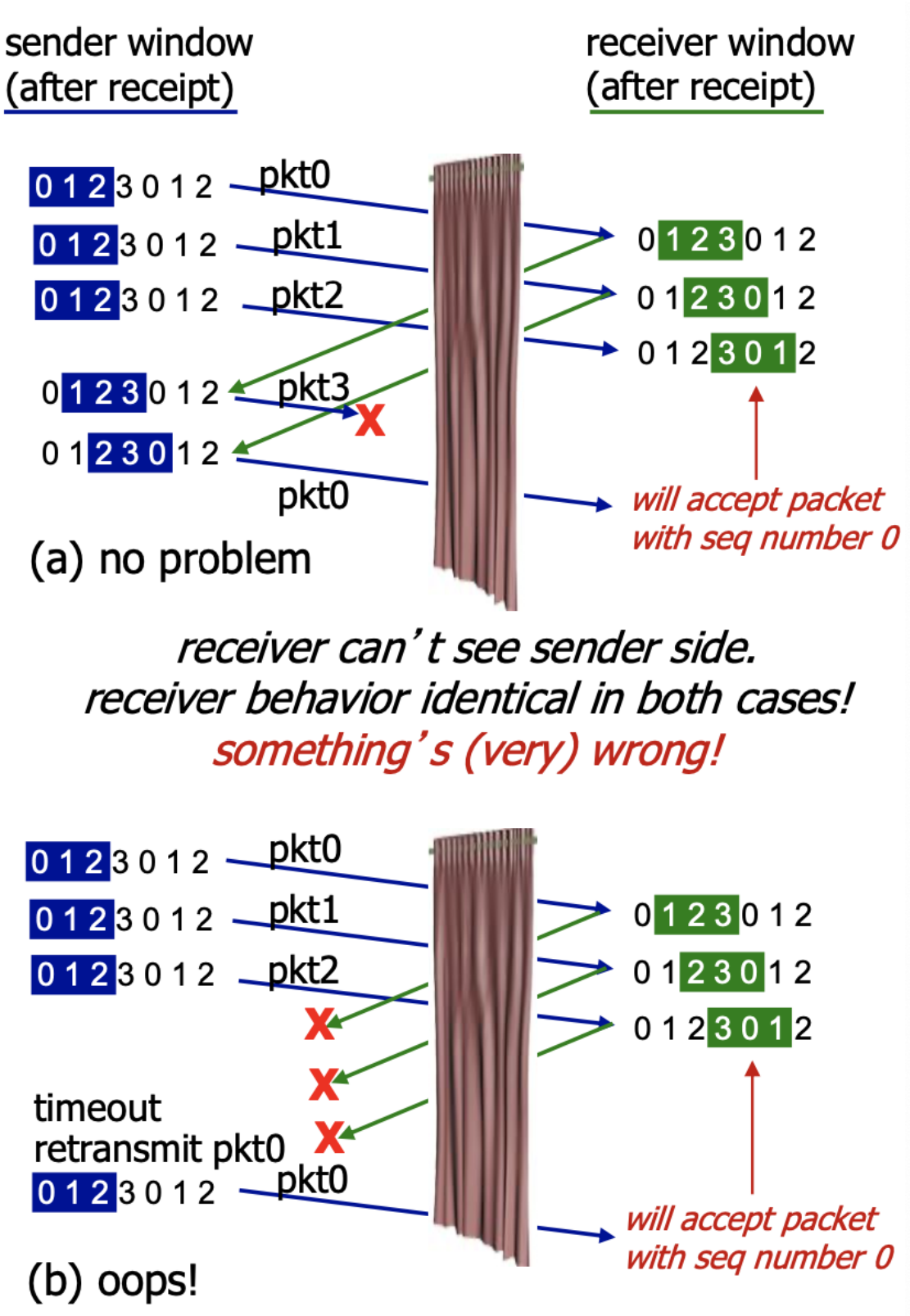
- Go-back-N에서는 순서에 안 맞으면 discard 해서 다시 보내야 되었었는데 selective repeat는 이를 보완한다.

1. sender
   a. window 안에 available seq 가 있으면 패킷을 보낸다.
   b. timeout
2. receiver
   a. out-of-order : buffer
   b. in-order : deliver
3. 예시


sender window (N=4)
sender
receiver
0 1 2 3 4 5 6 7 8
send pkt0
send pkt1
send pkt2
send pkt3
(wait)
loss
receive pkt0, send ack0
receive pkt1, send ack1
receive pkt3, buffer,
send ack3
rcv ack0, send pkt4
rcv ack1, send pkt5
receive pkt4, buffer,
send ack4
receive pkt5, buffer,
send ack5
record ack3 arrived
pkt 2 timeout
send pkt2
record ack4 arrived
record ack5 arrived
rcv pkt2; deliver pkt2,
pkt3, pkt4, pkt5; send ack2

4. dilemma

5.

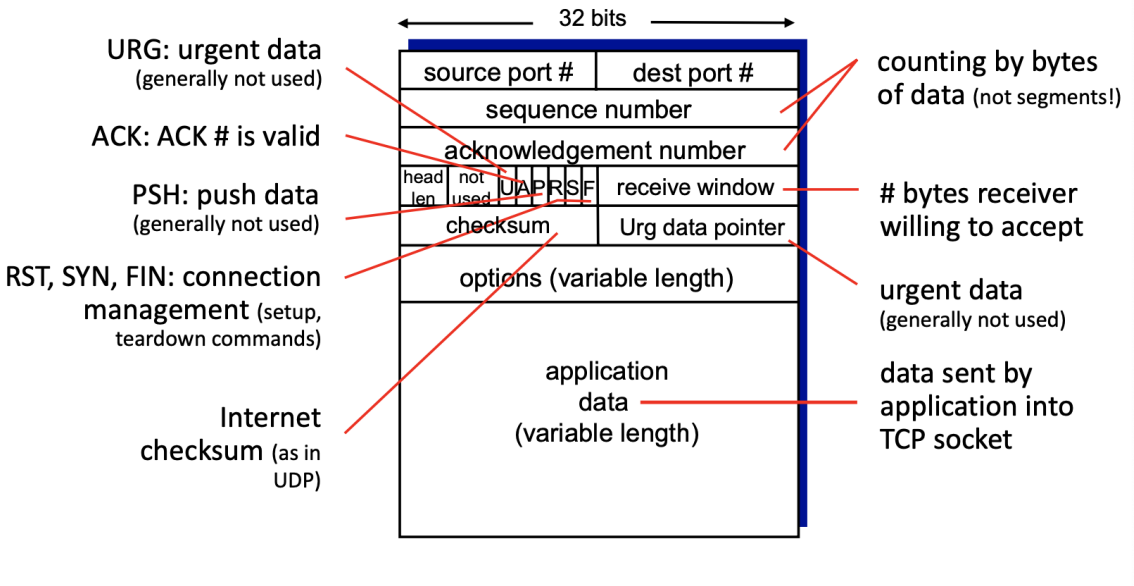


sender 와 receiver에서의 window 가 다르기 때문에 pkt0을 서로 다른 걸로 인식 할 수 있다.

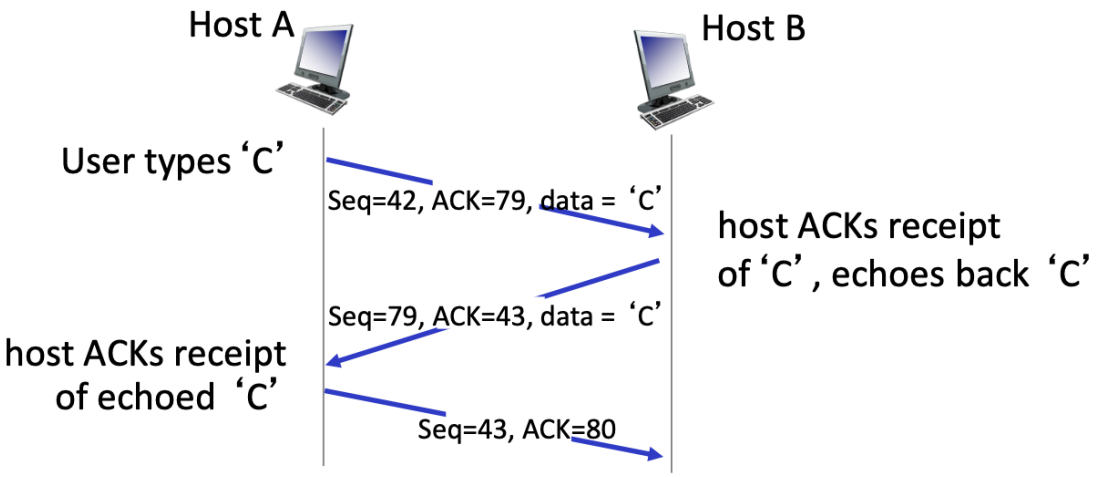
→ 이를 해결 하기 위해 sequence number / 2 ≥ window size 를 설정해준다.

TCP

1. 특징
   a. point-to-point : one sender, one receiver
   b. reliable, in-order
   c. pipelined : TCP congestion and flow control 이 window size 설정
   d. full duplex data : bi-directional data flow → TCP segment를 sender에서 receiver로 receiver에서 sender 로 보낸다.
   e. connection-oriented : three handshaking 초기설정
      → initializes sender, receiver state before data exchange
   f. flow controlled : sender will not overwhelm receiver
      → sender 에서 보내는 양이 recevier의 한계를 넘지 않는다.(데이터 무결성을 위해서)
2. segment 구조



   a. 기본 헤더는 20byte
   b. RST, SYN, FIN 을 통해 hankshaking 을 함(리셋, 시작, 끝)
3. sequence, ack



simple telnet scenario

bi-directional 이므로 host A, host B 는 모두 tcp segment 를 주고 받는다.

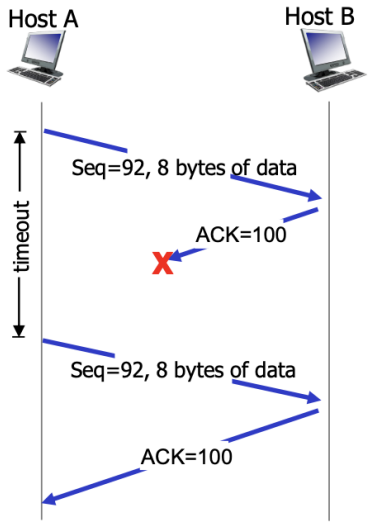
A, B가 모두 sender 이자 receiver 로 생각하면

a. A → B seq 42보낼게 79 보내줘

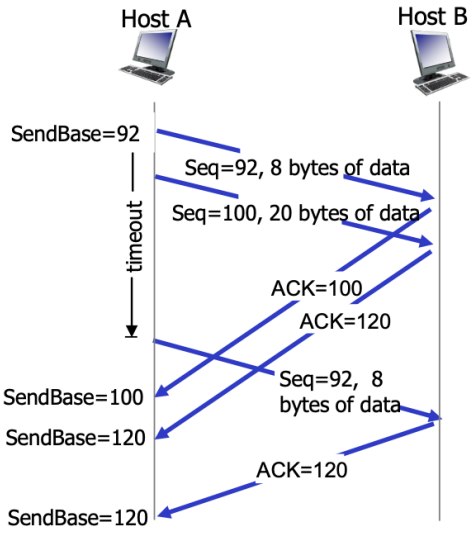
b. B → A seq 79 보낼게 43(다음꺼) 보내줘

c. A → B seq 43 보낼게 80 보내줘

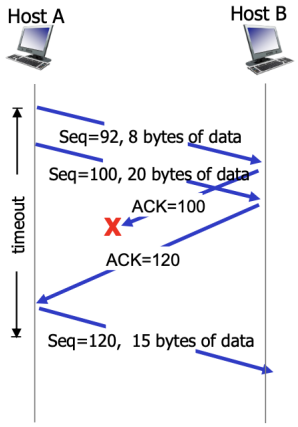
4. Retransmission (재전송)



lost ACK scenario



premature timeout



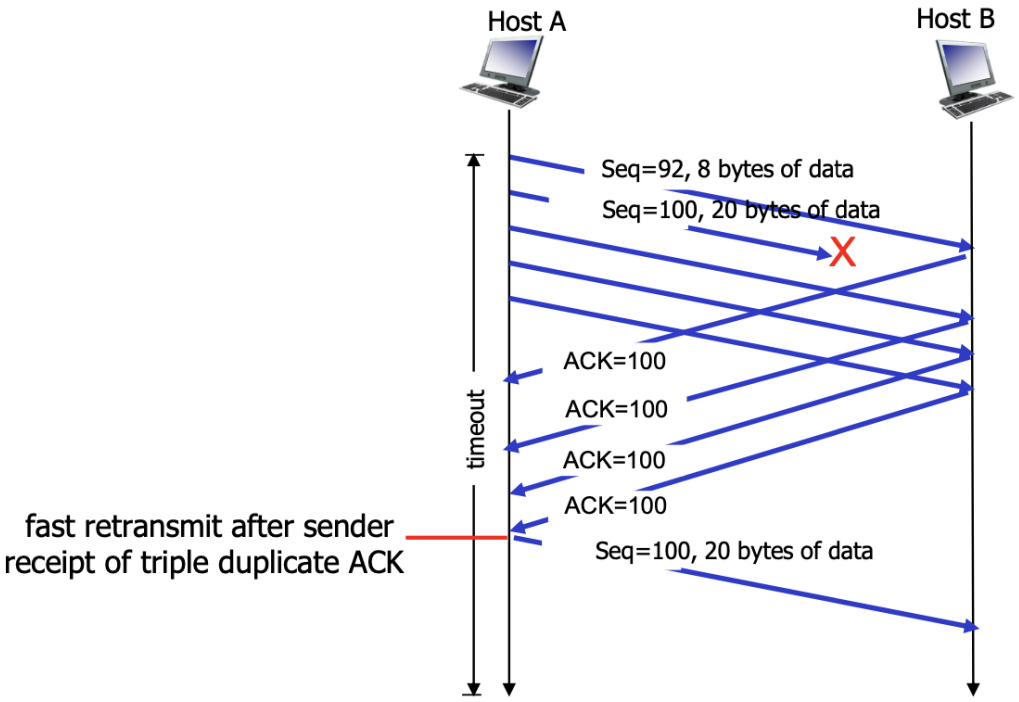
cumulative ACK

a. lost ACK scenario : ack가 도착하지 않아서 timeout 발생, sender 에서 재전송
b. premature timeout : timeout 시간이 너무 짧아서 재전송 후 도착함.
c. cumulative ACK : receiver에서 100을 못 받았다면 그 이후에 ack를 100을 보냈겠지만 120을 보냄으로써 그 이전것이 모두 정상적으로 도착했다는 것을 sender 에게 알려준다.(cumulative 은 마지막것만 보고 앞에것들은 모두 정상으로 도착했다 생각) 이로써 100을 다시 재전송 할 필요가 없음

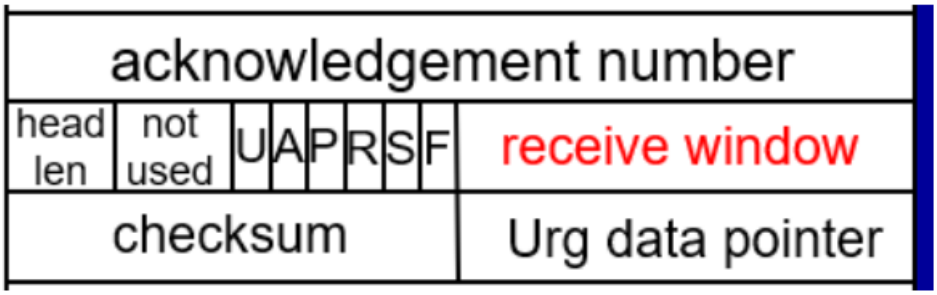
5. Round Trip Time (RTT), timeout
   a. Timeout value
      i. RTT 보다 길게!
         RTT : sender → receiver → sender 까지 걸리는 시간
      ii. too short : 너무 짧게 하면 다시 보내버리므로 너무 짧게 하면 안 됨
         1. too long : 너무 느리면 loss가 나도 반응을 느리게 한다.
6. Fast Retransmit

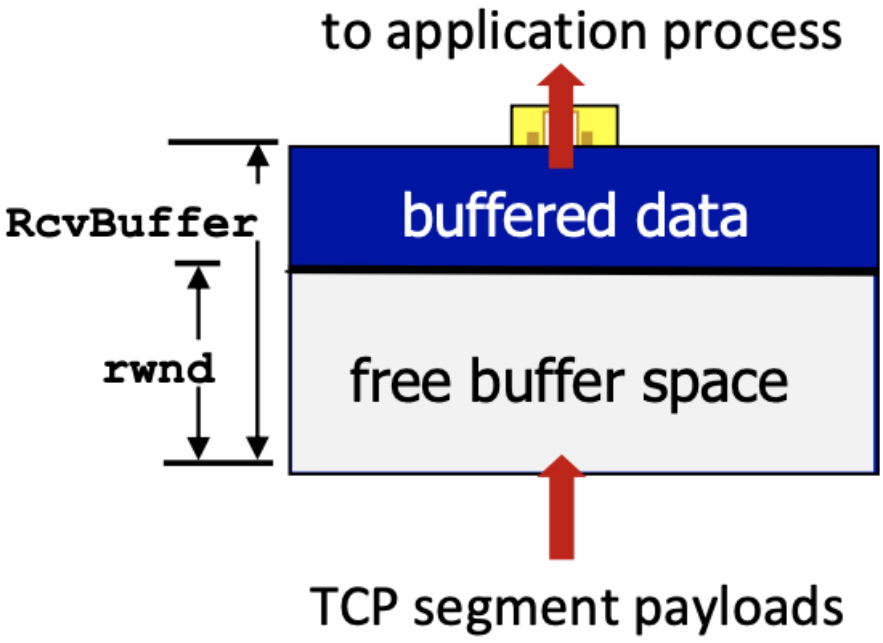
- 감기를 예로 들면 다 아프기 전에 병원을 빨리간다!

→ ACK가 3개가 반복된다면 timeout 이 지나지 않아도 도착하지 않았다는 것을 sender 가 빠르게 알고 재전송한다!



7. Flow Control





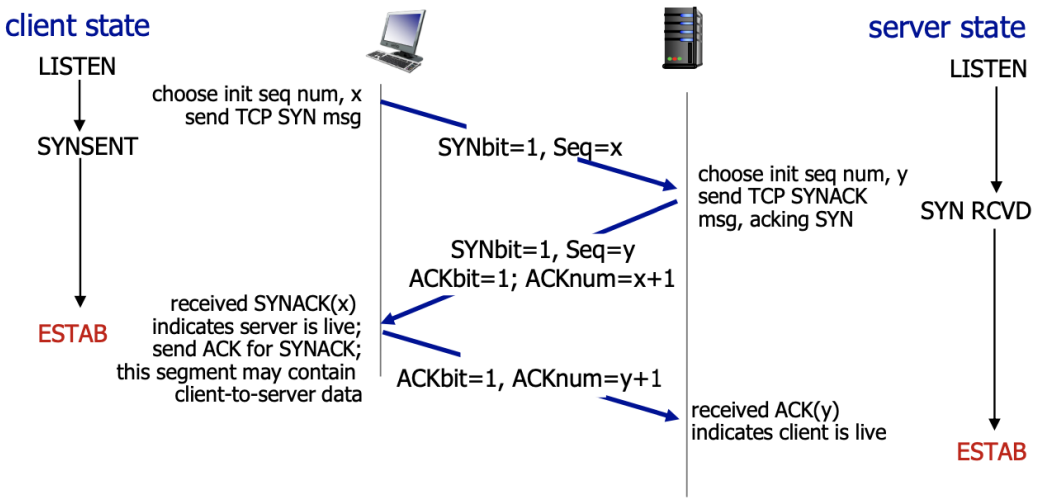
TCP receiver-side buffering

a. receiver의 free buffer space 를 header에 담아서 sender 에게 보내준다.

b. 따라서 sender 는 이를 보고 buffer 용량을 고려하여 데이터를 전송한다. (손실을 막기 위해)

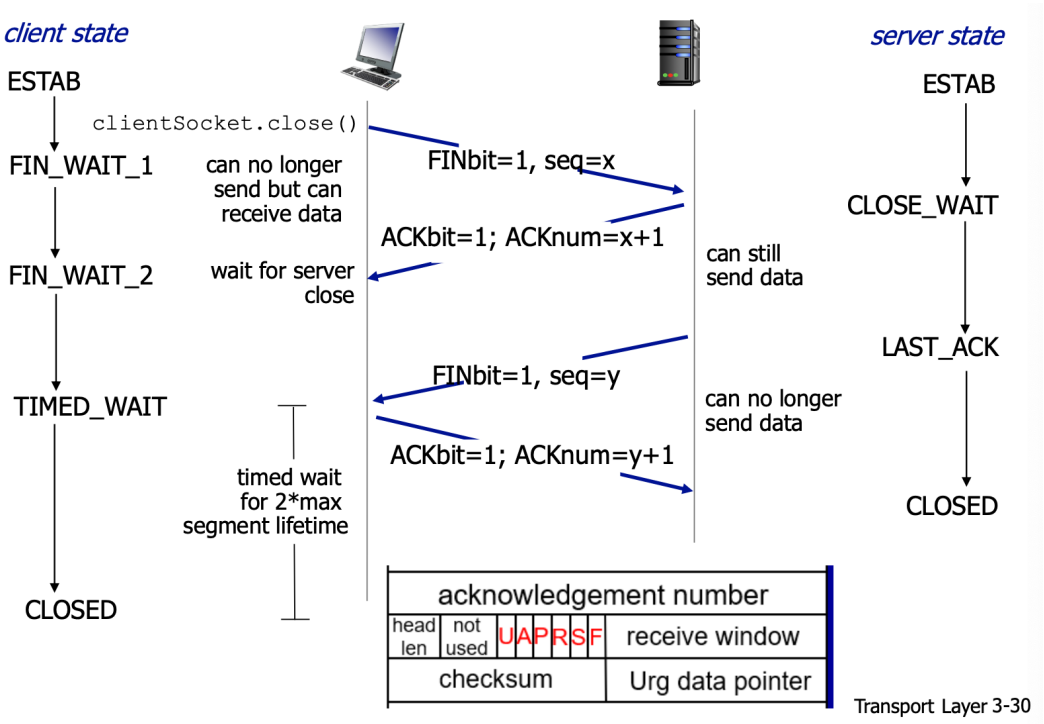
8. Connection Management

a. 3 way handshake



i. 시작할때는 SYN bit를 1로 설정

b. Closing



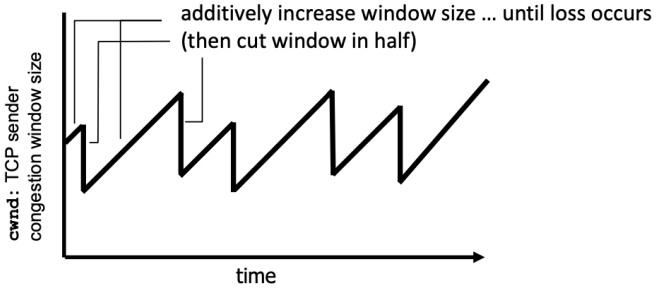
i. 끝날때는 FIN bit를 1로 설정 후 4단계로 close 함

9. Congestion control

- 네트워크가 관리하기에 너무 많은 데이터가 온다! 적당히 보내줘
- flow control 와 다름!

a. AIMD : Additive Increase Multiplicative Decrease

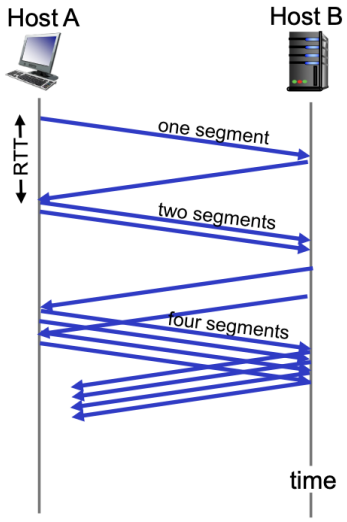
AIMD saw tooth behavior: *probing for bandwidth*



1. 천천히 증가, 아주 빠르게 감소
2. cwnd (sender 가 보낼 수 있는 window 용량)를 선형으로 조금씩 증가시키다가 최댜 용량을 만나면 확 떨어트린다.

b. Slow Start

- when connection begins, increase rate exponentially until first loss event:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT
  - done by incrementing **cwnd** for every ACK received
- *summary:* initial rate is slow but ramps up exponentially fast

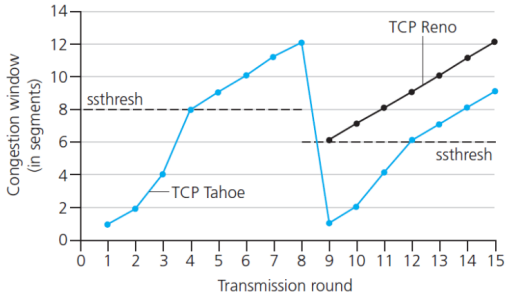


Q: when should the exponential increase switch to linear?

A: when **cwnd** gets to 1/2 of its value before timeout.

Implementation:

- variable **ssthresh**
- on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event



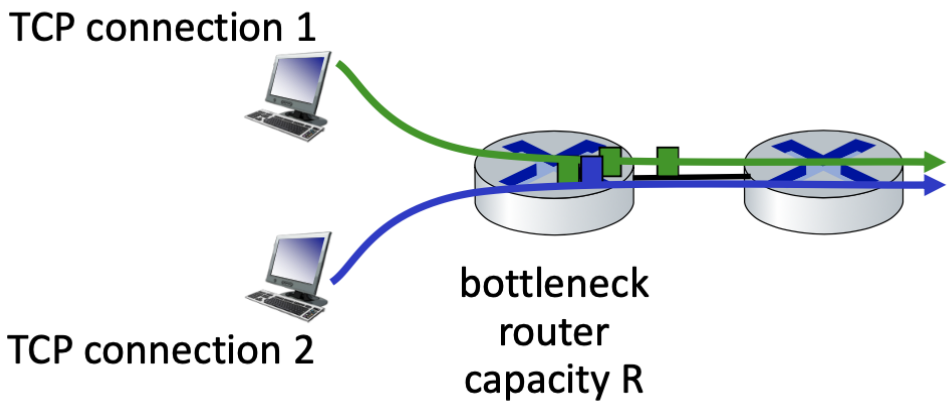
증가할때 2배씩 증가 하다가 ssthrest 를 만나면 선형으로 증가 이후 한계를 만나면 빠르게 감소.

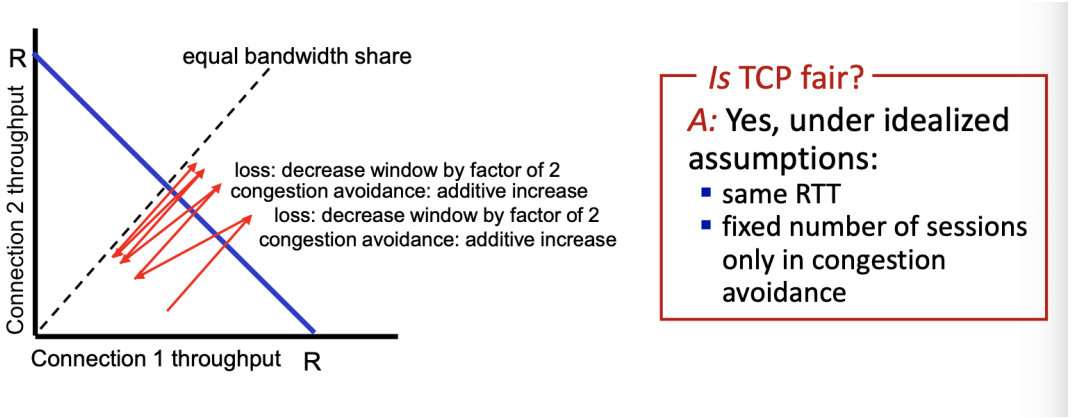
이때 최고점의 절반으로 ssthresh 설정 이후 반복

TCP Reno 는 맨 아래까지 갔다가 다시 올라오기에 시간이 오래걸리므로 ssthresh 를 기준으로 바로 선형으로 올라가도록 함.

10. Fairness

a. K개의 TCP session 들이 같은 라우터의 링크에 R의 bandwidth를 가지고 들어가면 똑같이 R/K 의 속도를 가진다. 공평하게!





b.

y = x 함수로 최대한 가깝게 만들어진다 (congestion avoidance를 통해서)

Explicit Congestion Notification (ECN)

IP header의 2비트가 network router에 의해서 congestion시 mark된다.

ECN 비트의 한 비트는 라우터가 정체를 겪고 있음을 나타내기 위해 사용되게 된다. 손실이 발생하기 전에 혼잡 시작을 송신자에게 알리는 비트를 설정할 수 있다. 나머지 한 비트는 발신 호스트가 라우터에게 송신자와 발신자 모두 ECN을 사용할 수 있음을 알리게 된다.