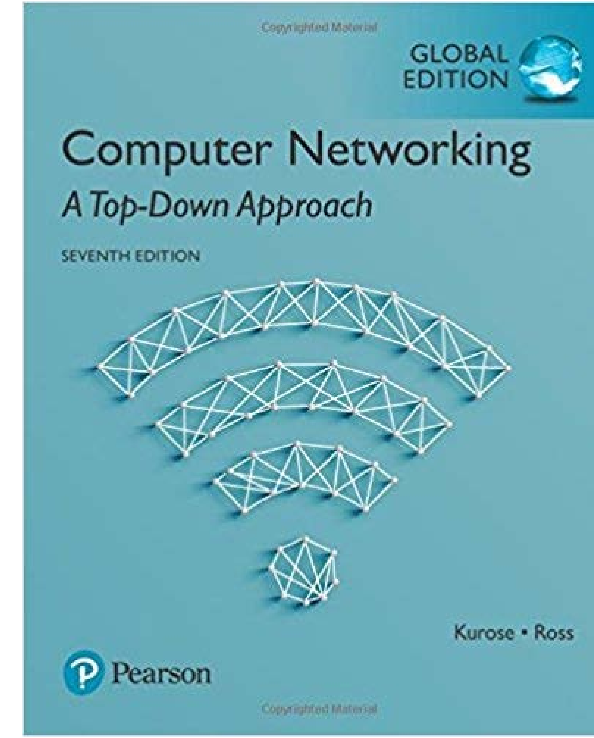
Chapter 3
Transport Layer part 4

School of Computing
Gachon Univ.
Joohyung Lee

Most of slides from J.F Kurose and K.W. Ross. And, some slides from Prof. Joon Yoo



Computer Networking: A Top Down Approach

7^{th} edition
Jim Kurose, Keith Ross
Pearson, 2017

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

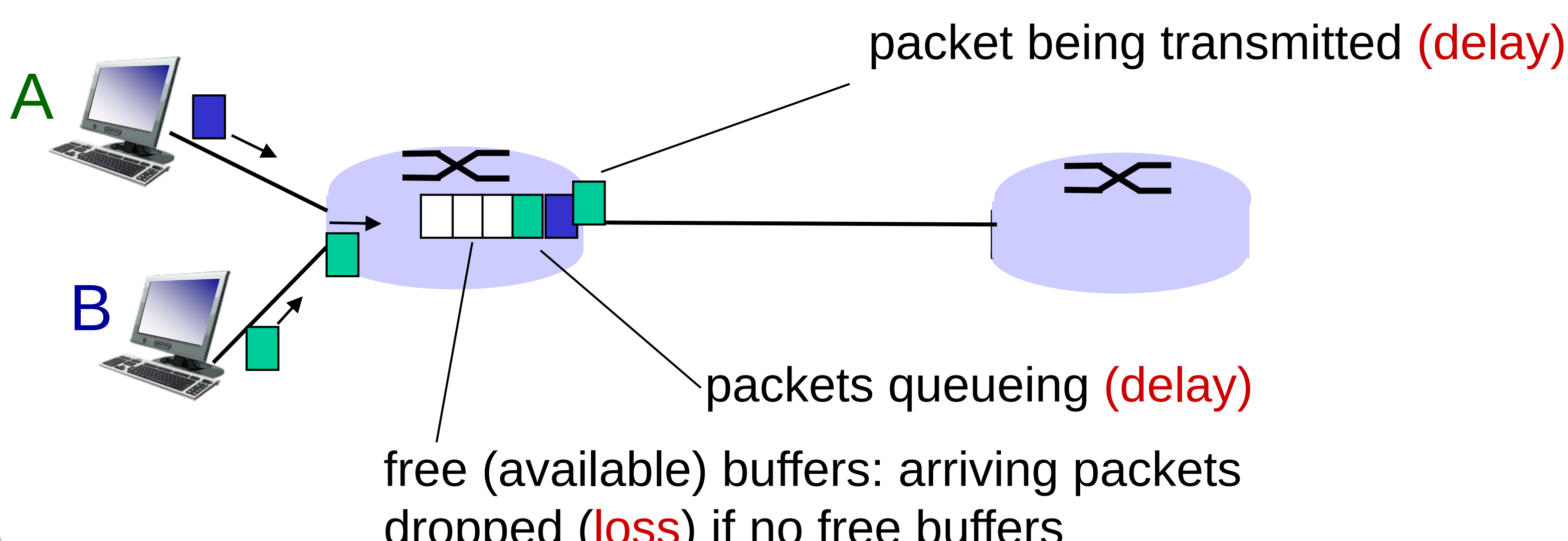
3.7 TCP congestion control

How do loss and delay occur?

packets *queue* in router buffers

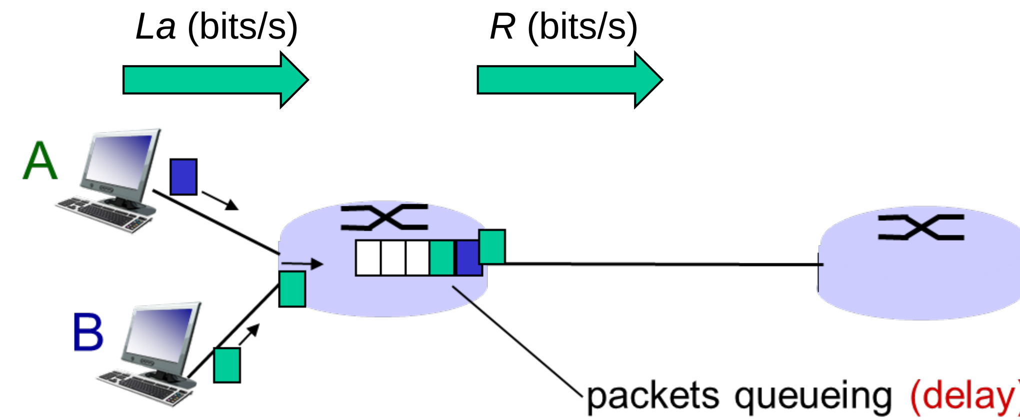
- packet arrival rate to link (temporarily) exceeds output link capacity
- packets queue, wait for turn
- Unlike other delays (d_{proc}, d_{trans}, and d_{prop}), queueing delay (d_{queue}) can vary from packet to packet

queueing delay : variation이 큼. 패킷마다도 다름
(첫 패킷과 마지막으로 들어오는 패킷)



Queueing delay (Ch. 1.4.2)

- *L:* packet length (bits)
- *a*: average packet arrival rate (packets/s)
- *R:* link bandwidth (bps)



- ***La/R* = *ρ* : Traffic Intensity**

link capacity보다 크니 queueing delay 발생

- *ρ* > 1: more "work" arriving than can be serviced; queue will tend to increase without bound; average delay infinite.
  - This should never happen!
- Design the system so that *ρ* ≤ 1

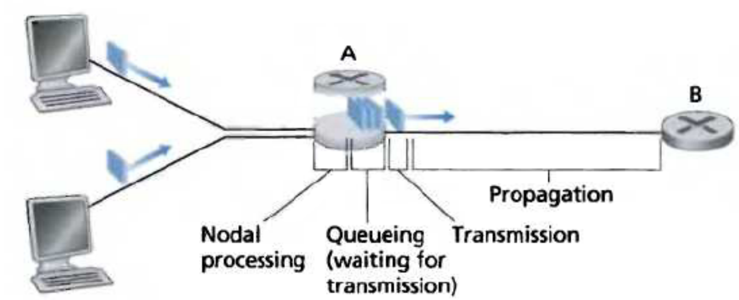
queue가 계속 늘어날 수밖에 없는 구조다

혼잡

Principles of congestion control

congestion:

- informally: "too many sources sending too much data too fast for ***network*** to handle"
- Symptoms (signs):
  - long delays (queueing in router buffers) **= queueing delay**
  - **lost packets** (buffer overflow at routers)
- **Congestion Control:**
  - ways to treat the cause of **network congestion**
  - different from **flow control**!
    - flow control considered the *receiver*'s capacity

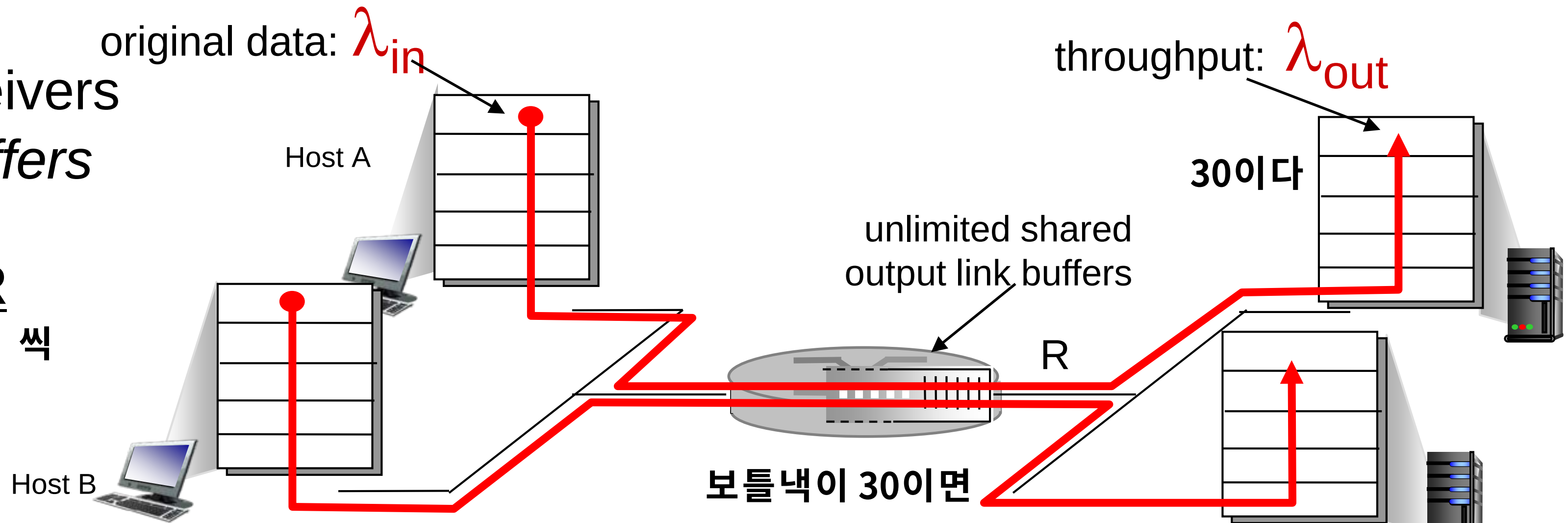


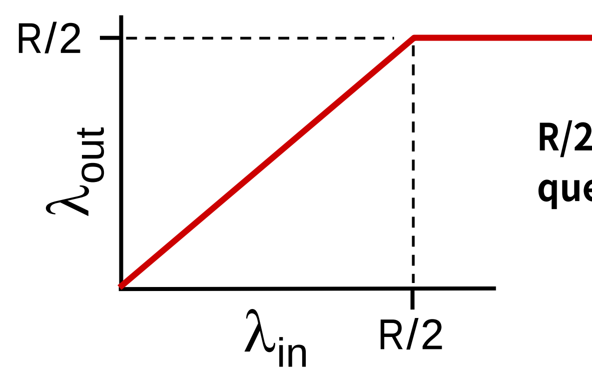
The nodal delay at router A

전송 속도를 줄이며

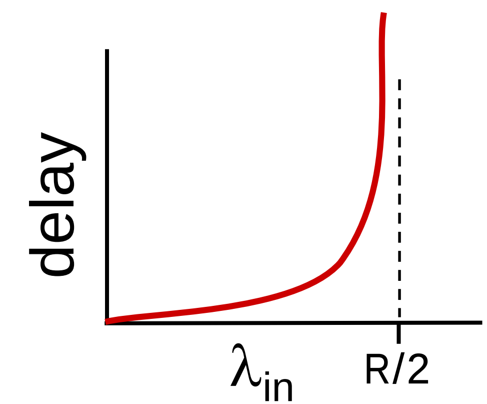
Causes/costs of congestion: scenario 1

- two senders, two receivers
- one router, *infinite buffers*
  - packet loss?
- output link capacity: **R**
  - each flow shares R/2 씩
- no retransmission





R/2을 넘으면
queue에 들어간다

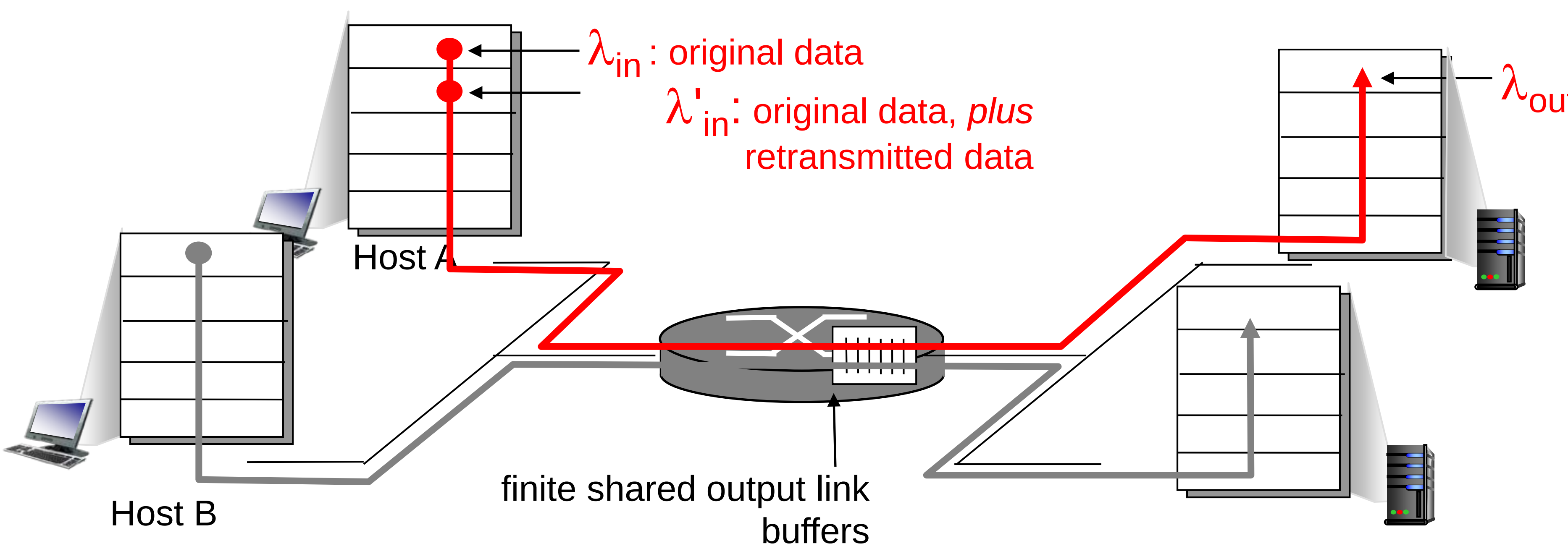


- maximum per-connection throughput: R/2
- **<u>large queuing delays</u>** as arrival rate, λ_{in}, approaches capacity

Causes/costs of congestion: scenario 2

- one router, *finite* buffers
  - packet loss?
- sender retransmission of timed-out packet
  - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
  - transport-layer input includes *retransmissions* : $\lambda'_{in} \geq \lambda_{in}$

패킷 로스를 고려하여 재전송 양을 합하니



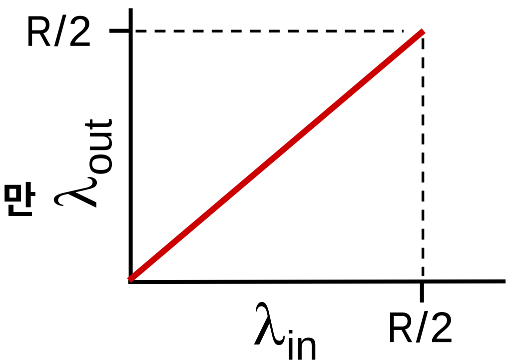
Causes/costs of congestion: scenario 2

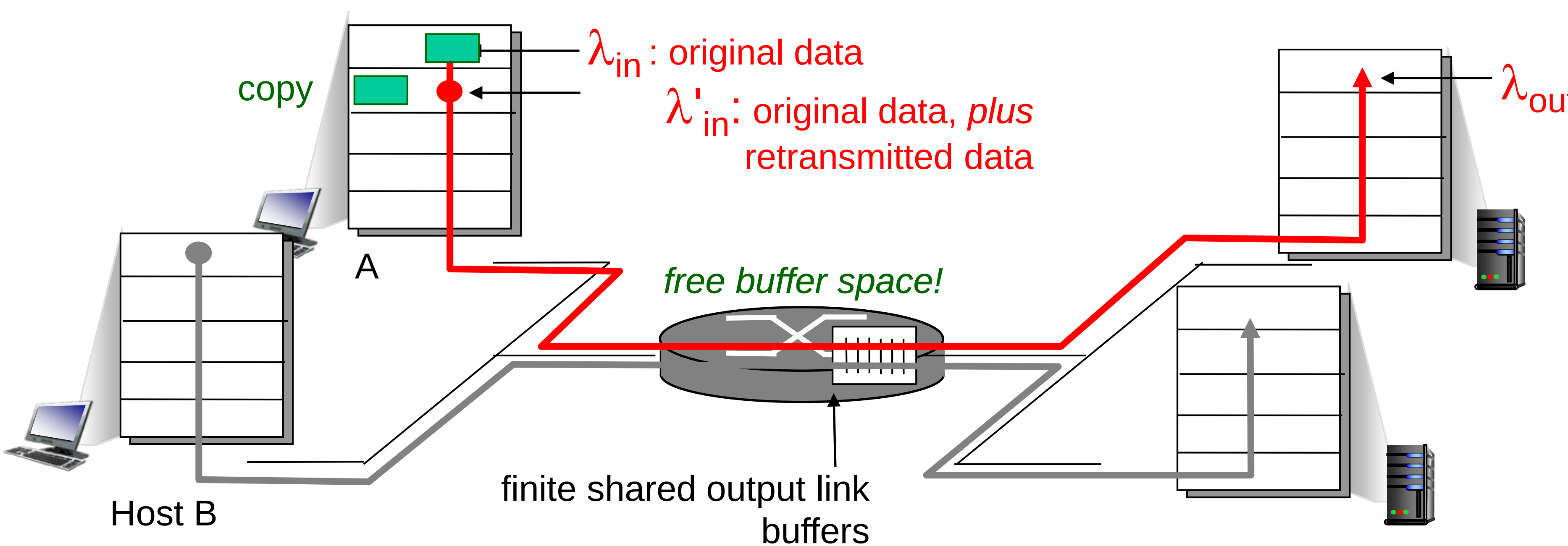
idealization1: perfect knowledge

- sender sends only when router buffers available
  - $\lambda'_{in} = \lambda_{in} = \lambda_{out}$

라우터 버퍼가 찼는지 알 수 없지만 안다고 가정

loss가 안생김





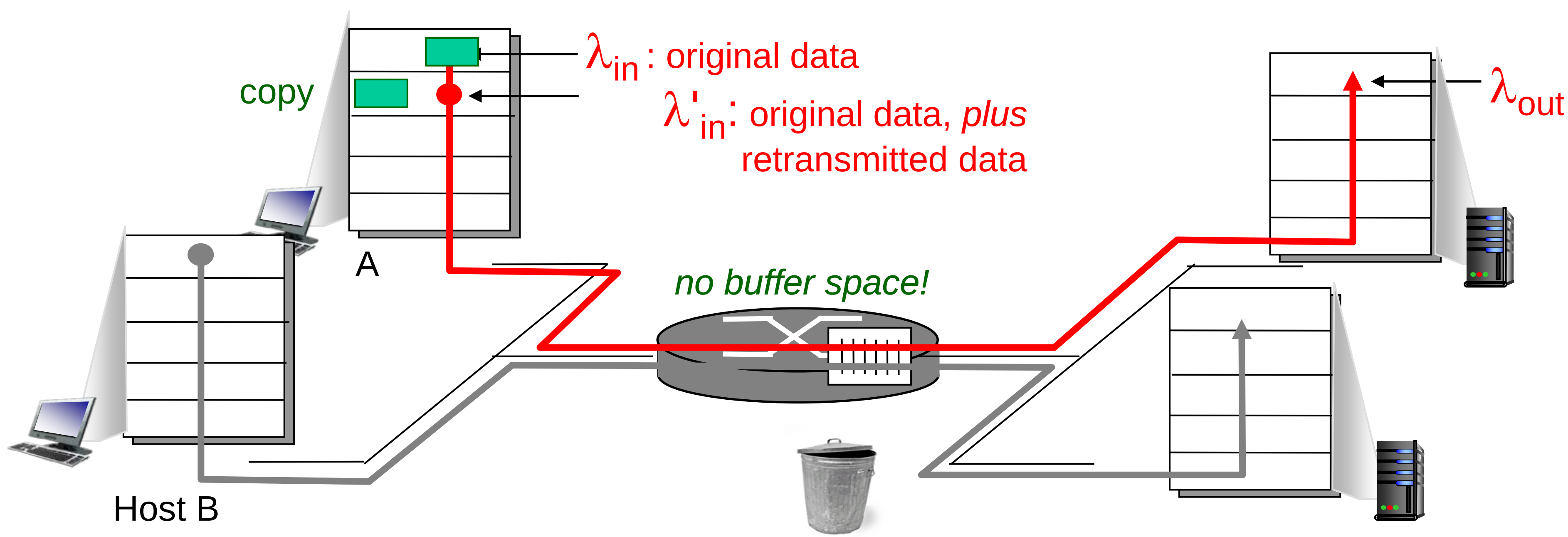
Causes/costs of congestion: scenario 2

Idealization2: known loss
packets can be lost, dropped at router due to full buffers

라우터 버퍼가 찼는지 알 수 없기에

- sender only resends if packet *known* to be lost
  - $\lambda'_{in} \geq \lambda_{in}$

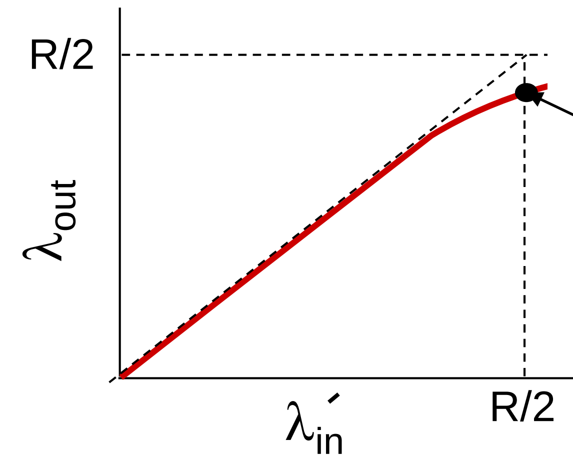
로스가 난 것들도 보내기 때문에



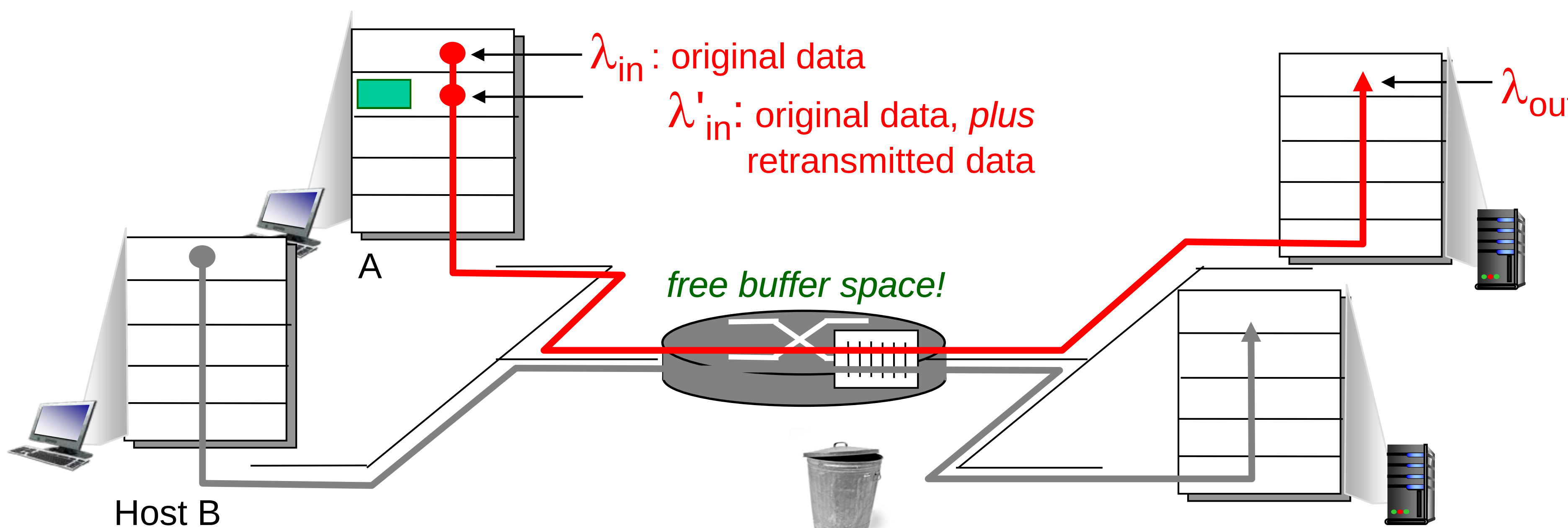
Causes/costs of congestion: scenario 2

Idealization2: known loss
packets can be lost, dropped at router due to full buffers

- sender only resends if packet *known* to be lost
  - $\lambda'_{in} \geq \lambda_{in}$



재전송하는만큼 네트워크를 씀

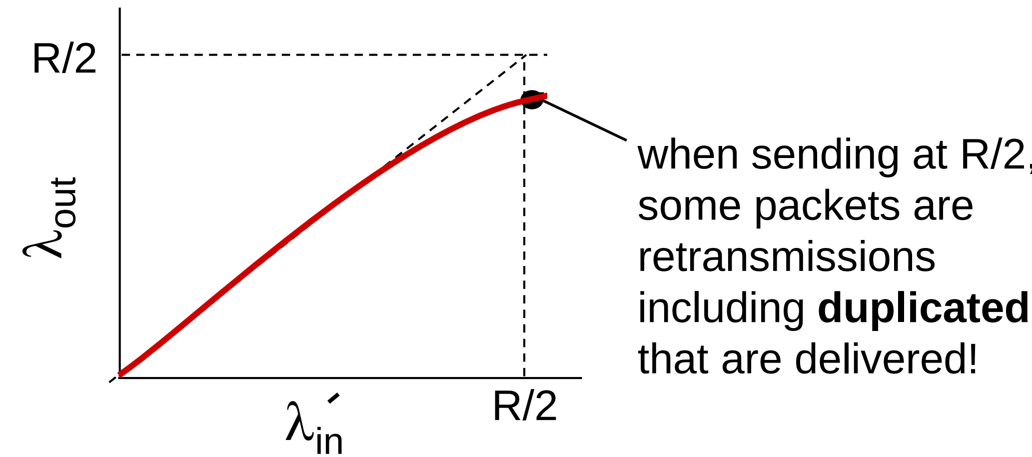


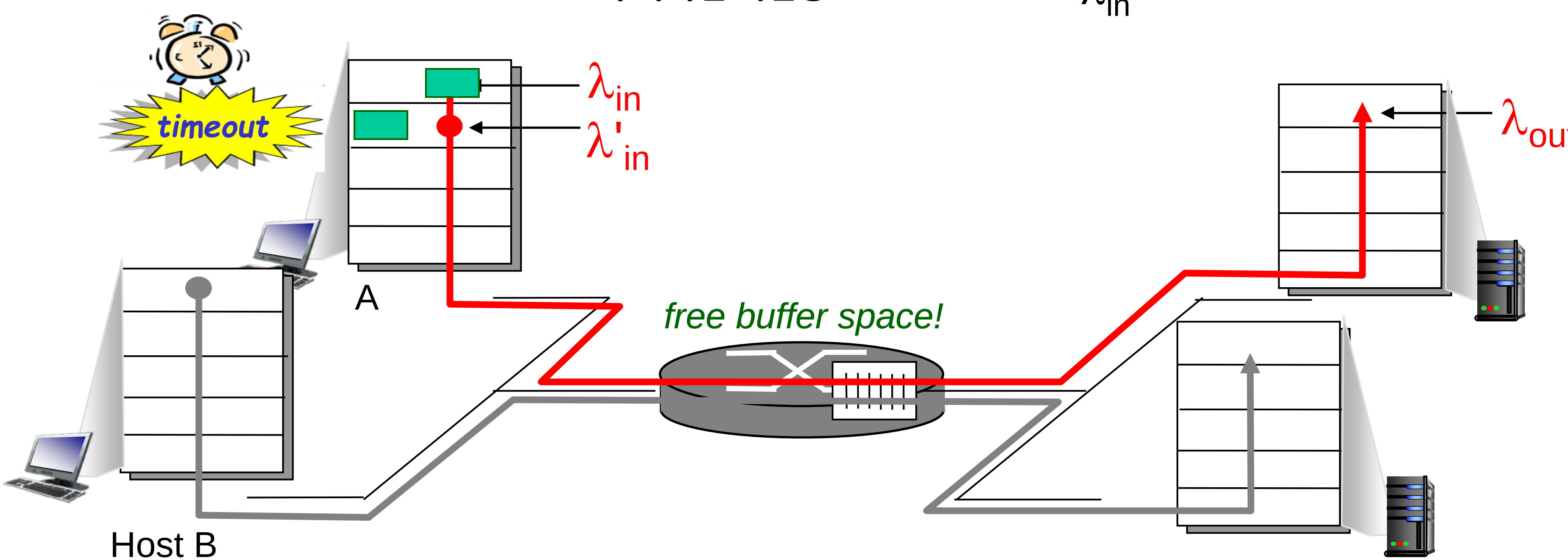
Causes/costs of congestion: scenario 2

Realistic: duplicates

- packets can be lost, dropped at router due to full buffers
- sender times out prematurely, sending *two* copies, both of which are delivered

timeout이 켜지면 재전송

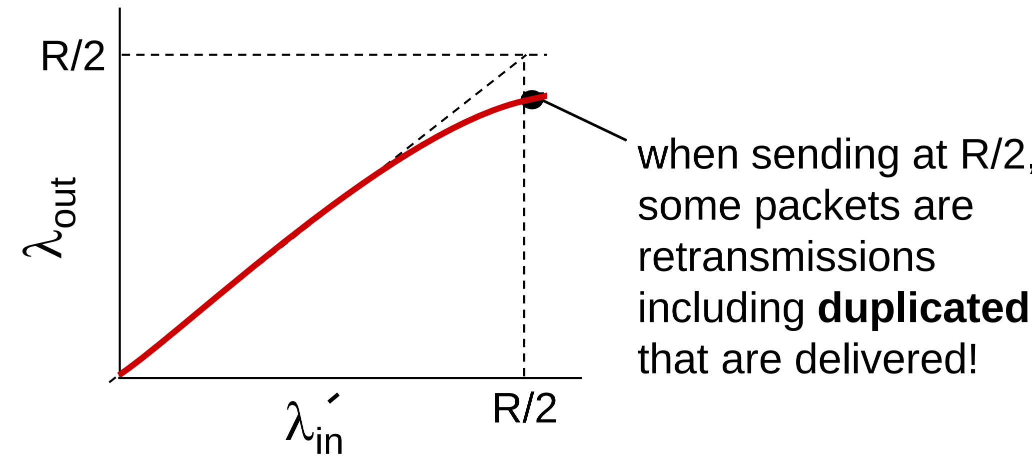




Causes/costs of congestion: scenario 2

Realistic: duplicates

- packets can be lost, dropped at router due to full buffers
- sender times out prematurely, sending *two* copies, both of which are delivered

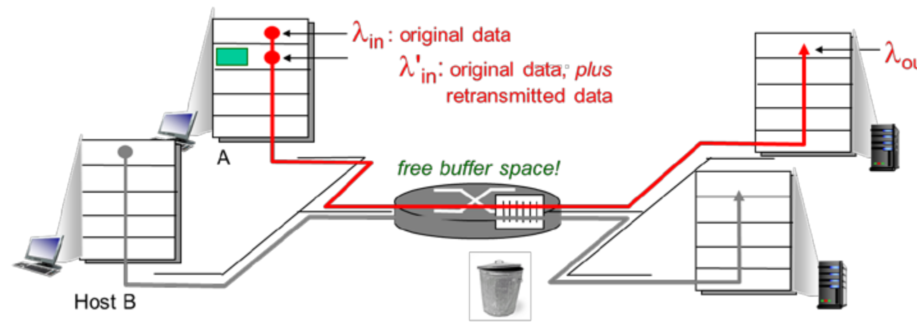


"costs" of congestion:

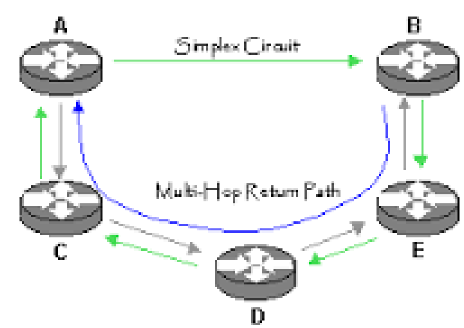
- large queueing delays
- more work (retrans) for given "goodput"
- unneeded retransmissions: link carries multiple copies of pkt
  - decreasing goodput

From single-hop to multi-hop path

- Until now (scenario 2), we only considered a **single-hop** path congestion

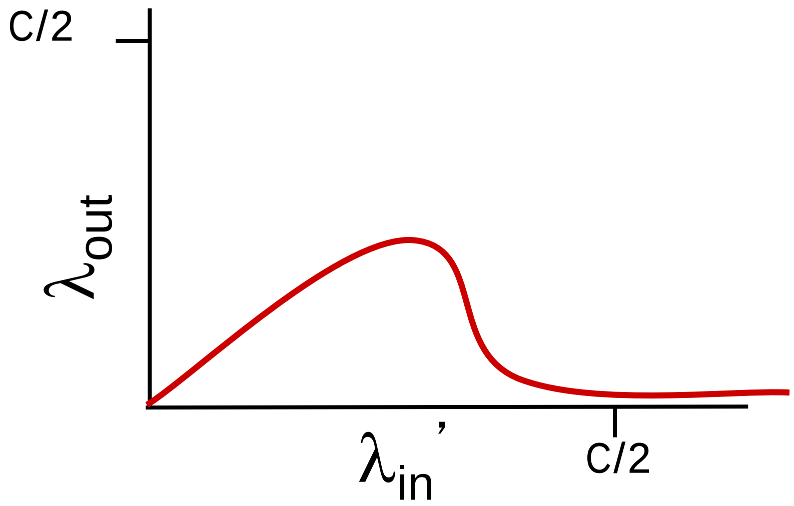


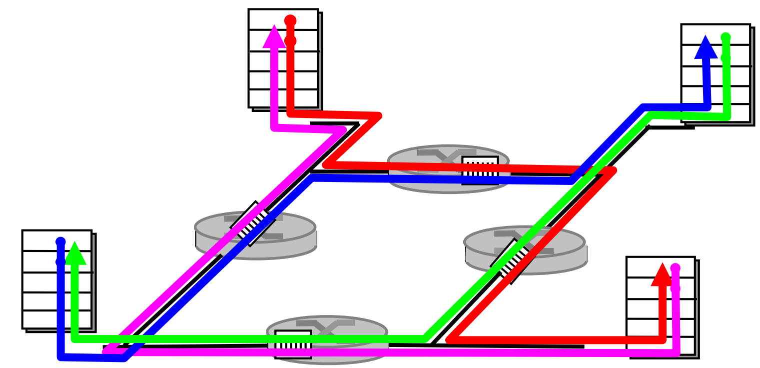
- But, generally in the Internet, the network path is **multi-hop** (Scenario 3)
- Congestion can happen at any link along the multi-hop path



한개라도 congestion이 발생하면 goodput이 줄어듦

Causes/costs of congestion: scenario 3





another "cost" of congestion:

- when packet dropped, any transmission capacity used along the path for that packet was wasted!

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

분산 알고리즘 : 각각의 단말기들이 congestion을 피하도록 함 (분산되어)

TCP congestion control

- TCP/IP protocol stack was operational in early 1980s
- But, Internet was suffering from congestion collapse
  - hosts would send their packets into the Internet as fast as received window would allow (only flow control)
  - congestion would occur at some router and packet drop happens, and hosts would retransmit their packets, resulting in even more congestion
- TCP congestion control introduced by Van Jacobson in later 1980s
  - Each host (end systems) determines how much capacity is available in the network

그때 당시 link capacity가 낮아서

sending rate를 조절

TCP congestion control

중간에 router는 개입하지 않고 서버와 client 사이

- End-to-end congestion control

link capacity와 현재 traffic intensity를 이용하여서 계속 바뀜

- Sender needs to determine **network capacity**, which changes over time

네트워크 상태에 맞게 n을 줄이거나 늘림 (n=1이면 1개 보내고 ack 받고)

- Sender limits the **sending rate** as a function of available network capacity (i.e., network congestion)
  - e.g., little congestion – increase **send rate**
  - e.g., congestion along path – decrease **send rate**
- The only feedback to sender: **ACK**

ack을 가지고 congestion을 추종해보자

  - ACK arrival: a packet has arrived safely at receiver
  - ACK timeout: has not arrived -> congestion이 의심된다

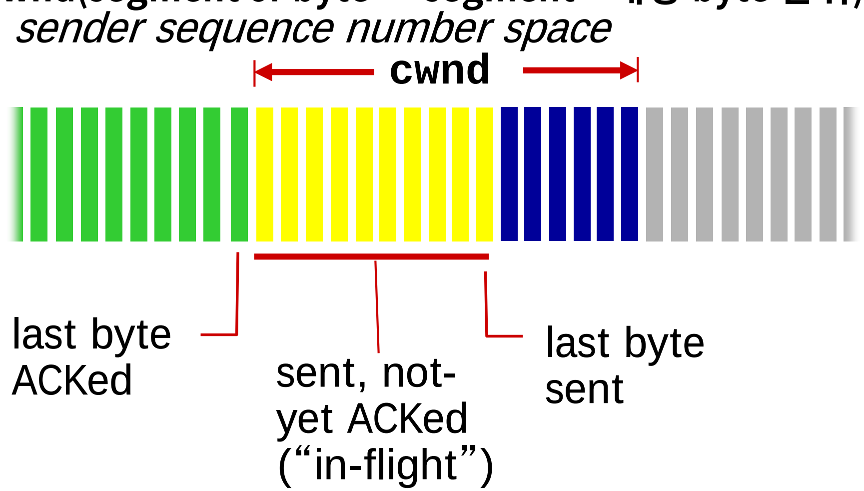
TCP congestion control

- Questions
  1. **How does the sender limit the rate?**
  2. How does the sender know there is congestion on path between itself and destination?
  3. What algorithm should sender use to change send rate as function of congestions?

n 값을 바꾸며 속도 조절

TCP Congestion Control: congestion window

고백앤에서만 n(segment 단위)을 쓰고 TCP에서는
cwnd(segment or byte -> segment * 개당 byte 단위)을 씀



❖ sender limits transmission:

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}$$

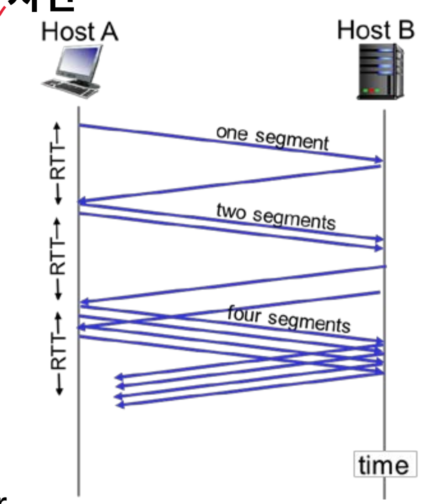
❖ **cwnd** is dynamic, function of perceived network congestion

TCP sending rate:

❖ *roughly:* send **cwnd** bytes, wait **RTT** for ACKS, then send more bytes

$$\boxed{\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \ \text{bytes/sec}}$$

시간



TCP Congestion Control: congestion window

Congestion control(cwnd) *and* ***Flow control (advertised wnd)*** *determine TCP sending rate*

네트워크 상황에 맞춰서 window 사이즈를 늘리거나 줄이거나

목적 : 리시버에서 손실을 막으려고

Send Window = MIN(flow control window, **congestion window**)

Note: advertised window is not so dynamic!

= advertised window

TCP congestion control

- Questions
  1. How does the sender limit the rate?
  2. **How does the sender know there is congestion on path between itself and destination?**
  3. What algorithm should sender use to change send rate as function of congestions?

Network Congestion

수많은 라우터들 중 congestion이 만들어지는지 알 수 는 없다

- How can the TCP sender know there is congestion?
  - Where does congestion actually happen?
  - However, network layer (e.g., router) does not send explicit message of congestion
- Loss event!
  - When there is excessive congestion, one (or more) router buffers along the path overflows, causing packet drop – **loss**!
  - Indication of loss
    - Timeout
    - Receipt of three duplicate ACKs

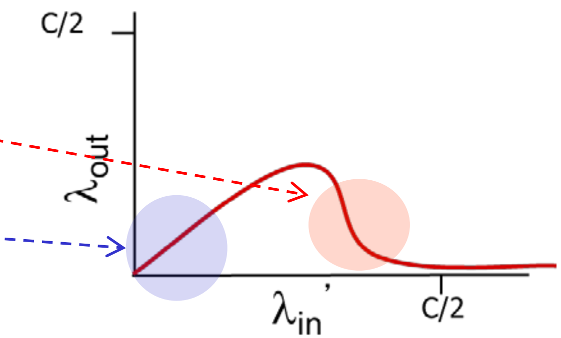
TCP congestion control

- Questions
  1. How does the sender limit the rate?
  2. How does the sender know there is congestion on path between itself and destination?
  3. **What algorithm should sender use to change send rate as function of congestions?**

TCP congestion control principles

- How should TCP sender determine send rate (or cwnd)?
  - If TCP senders collectively send too fast – network congestion may happen
  - If TCP senders cautiously send too slow – underutilize the network

    utilization이 안됨

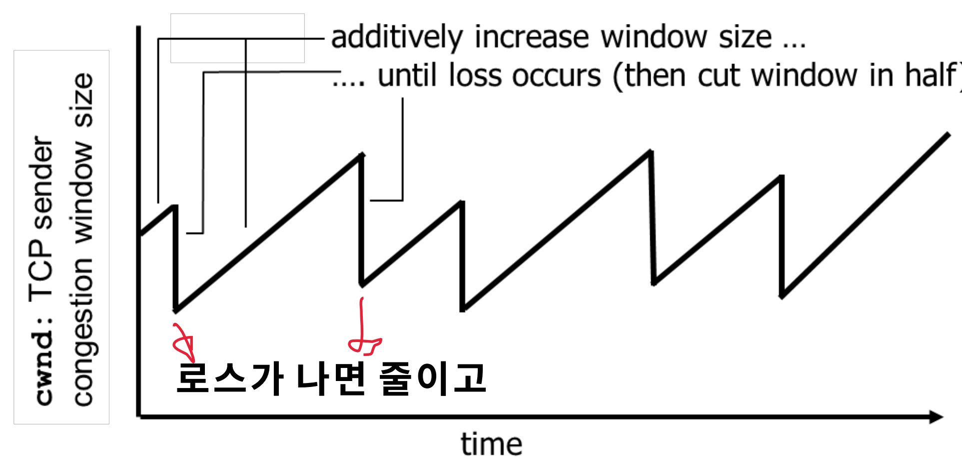


- TCP principles
  - A lost segment implies congestions, and hence, the TCP sender's rate should be decreased
  - An acknowledged segment indicates successful delivery, and hence, the sender's rate can be increases when ACK arrives

TCP congestion control principles

- Bandwidth probing
  - TCP sender increases its rate in response to arriving ACKs until loss event occurs – then decrease rate
  - TCP sender increases its rate to probe for rate at which congestion begins
  - Then backs off from that rate, and then begins probing again to see if the congestion rate has changed.

- TCP Congestion Control Algorithm
  - Slow start, congestion avoidance, fast recovery



TCP Congestion Control: Two modes

- TCP congestion control is governed by two parameters:
  - **Congestion Window (cwnd)**
  - **Slow-start threshold Value (ssthresh)**

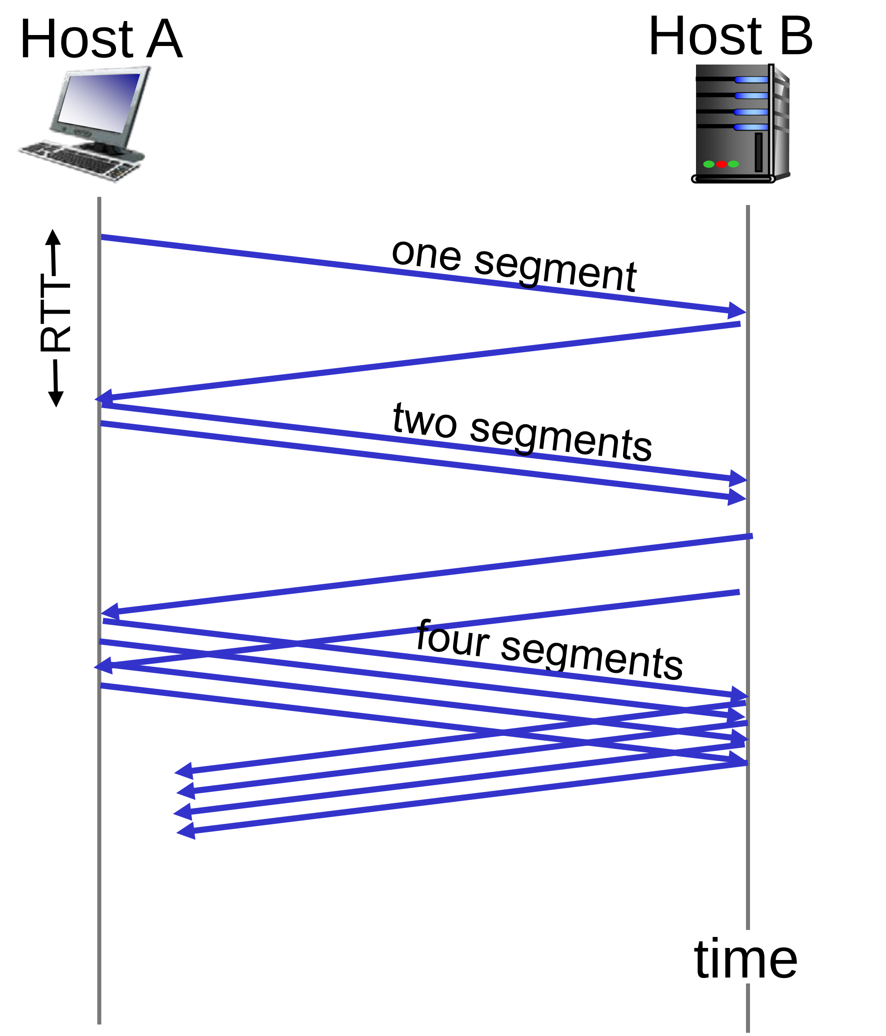
exponential로 *2 늘리다가
linear 하게 늘린다

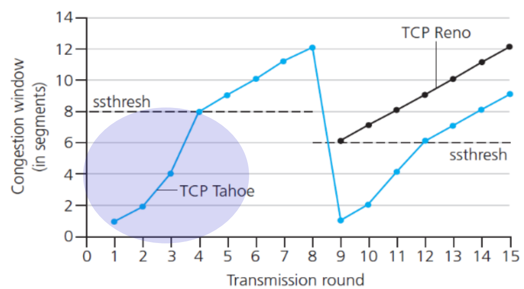
작은(slow) 윈도우에서 2배씩 늘린다

- Congestion control works in <u>two modes</u>:
  - **Slow Start** (cwnd < ssthresh)
  - **Congestion Avoidance** (cwnd ≥ ssthresh)
  - **(Fast Recovery)**

TCP "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





From Slow Start to Congestion Avoidance

TCP slows down the increase of **cwnd** if **cwnd** reaches the slow-start threshold value **ssthresh**

- If cwnd ≥ ssthresh then each time an ACK is received, increment cwnd as follows:
  - cwnd = cwnd + (1 MSS)/cwnd (linear increase per RTT)

1씩 증가

4개 -> 1/4 1/4 1/4 1/4인데
1/4을 더하니
1개씩 늘어나도록 함

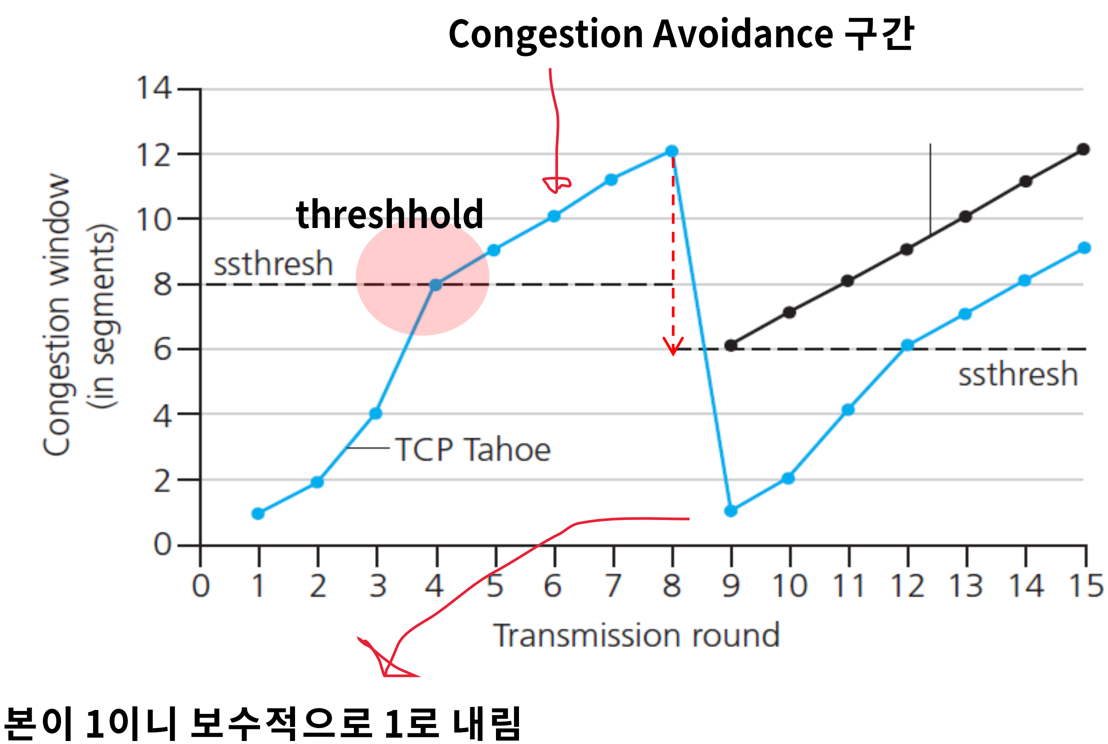
Setting of ssthresh:

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

TCP: switching from slow start to CA

Q: when should the exponential increase switch to linear?

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

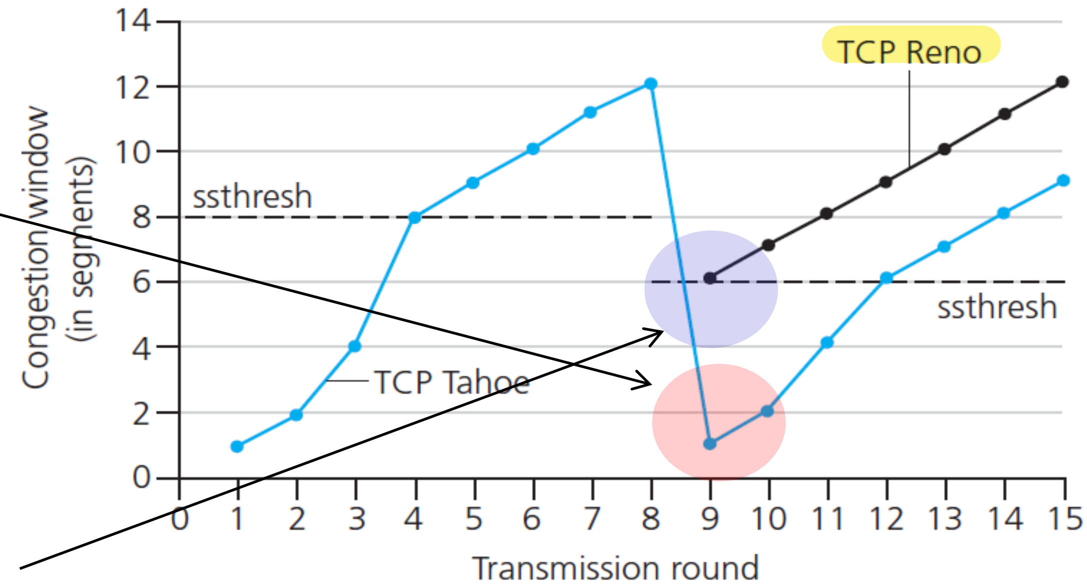


TCP Tahoe Algorithm

```
Initially:
    cwnd = 1 MSS;
    ssthresh = advertised window size;
New Ack received:
    if (cwnd < ssthresh)
        /* Slow Start*/
        cwnd = cwnd + 1 MSS;    => 2배씩
    else
        /* Congestion Avoidance */
        cwnd = cwnd + (1 MSS)/cwnd;    => 1씩 증가
Timeout:
    /* Multiplicative decrease */
    ssthresh = cwnd/2;
    cwnd = 1 MSS;
```

TCP: detecting, reacting to loss

- loss indicated by **timeout**:
  - **cwnd** set to 1 MSS;
  - window then grows exponentially (as in slow start) to threshold, then grows linearly
- **Fast Recovery (TCP Reno):** loss indicated by 3 duplicate ACKs
  - dup ACKs indicate network capable of delivering some segments (better than timeout)
  - **cwnd** is cut in half window then grows linearly

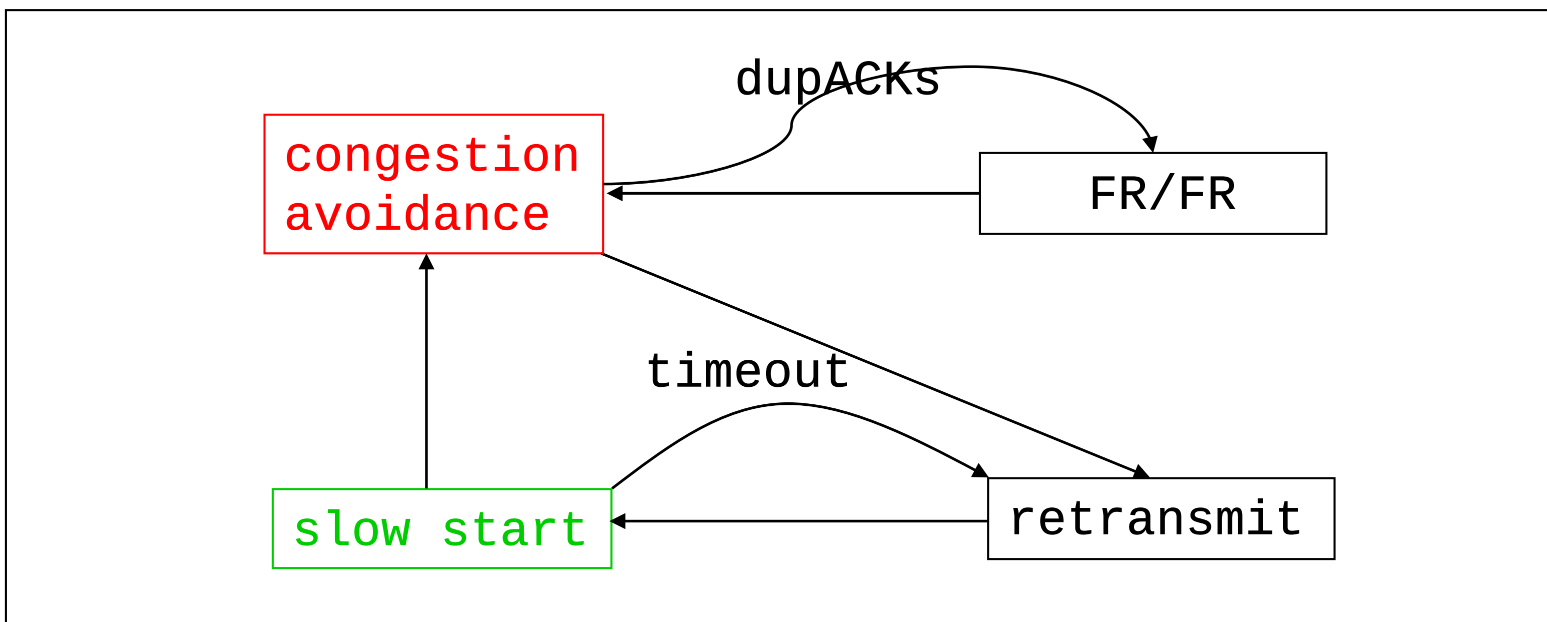


timeout에 대한 loss보다는 threshold가 안전하다고 생각하자

시험범위

TCP Reno Summary

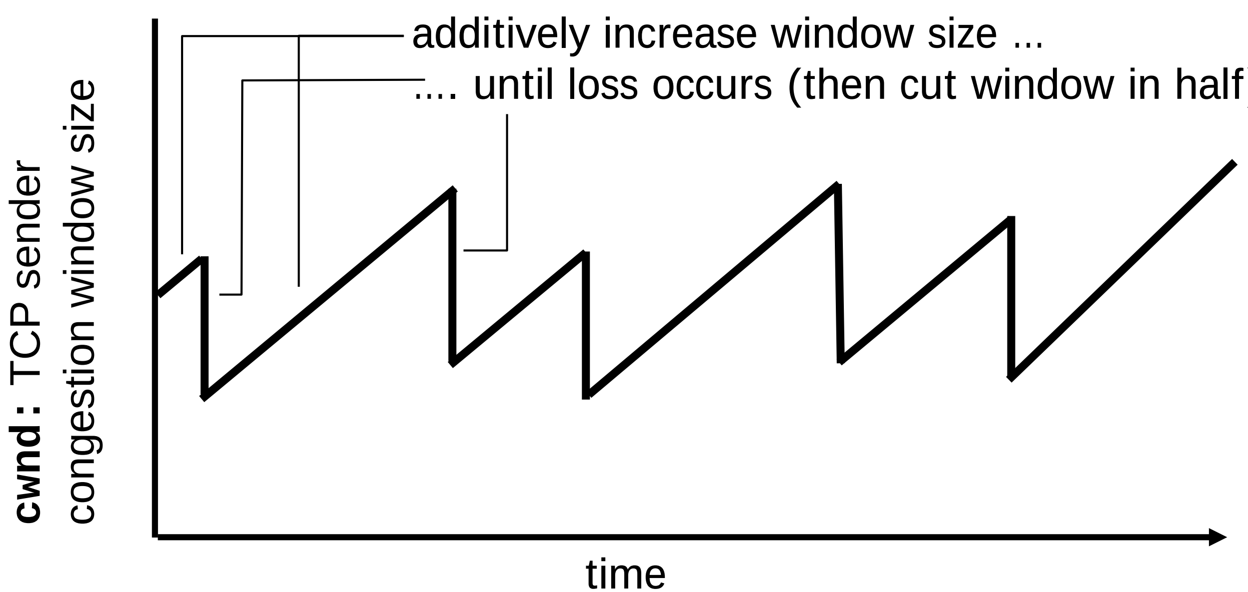
- Basic ideas
  - Fast recovery avoids slow start
  - dupACKs: fast retransmit + fast recovery
  - Timeout: fast retransmit + slow start



Additive Increase Multiplicative Decrease (AIMD)

- *approach:* sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **cwnd** by 1 MSS every RTT until loss detected
  - *multiplicative decrease*: cut **cwnd** in half after loss (3-dup acks)

AIMD saw tooth behavior: probing for bandwidth



Chapter 3: summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation, implementation in the Internet
  - UDP
  - TCP