

Lecture 5-3

Congestion Control

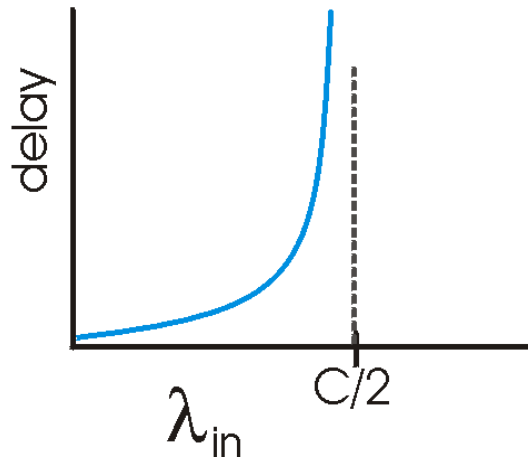
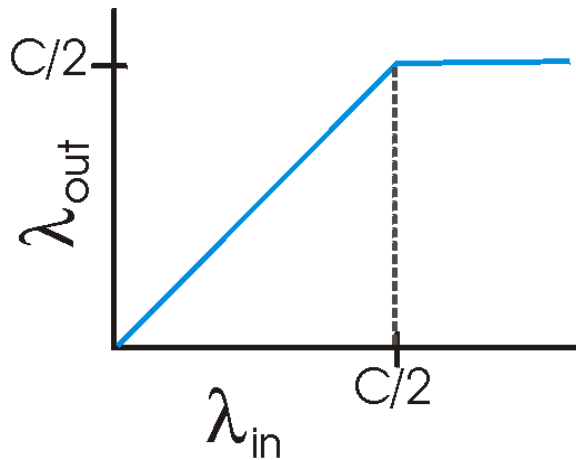
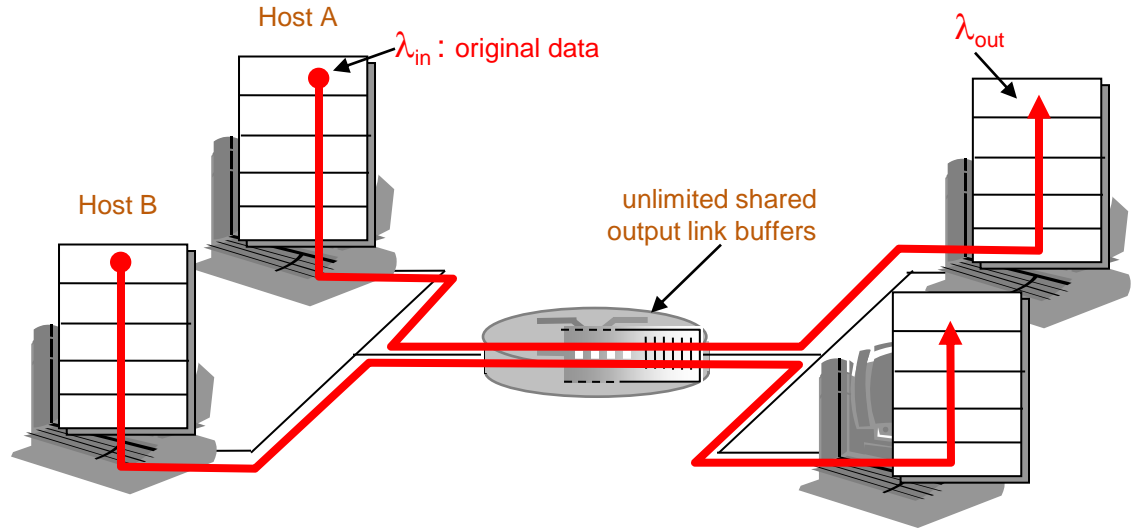
Principles of Congestion Control

Congestion:

- informally: “too many sources sending too much data too fast for *network* to handle”
- different from flow control!
- manifestations:
 - lost packets (buffer overflow at routers)
 - long delays (queueing in router buffers)
- a top-10 problem!

Causes/costs of congestion: scenario 1

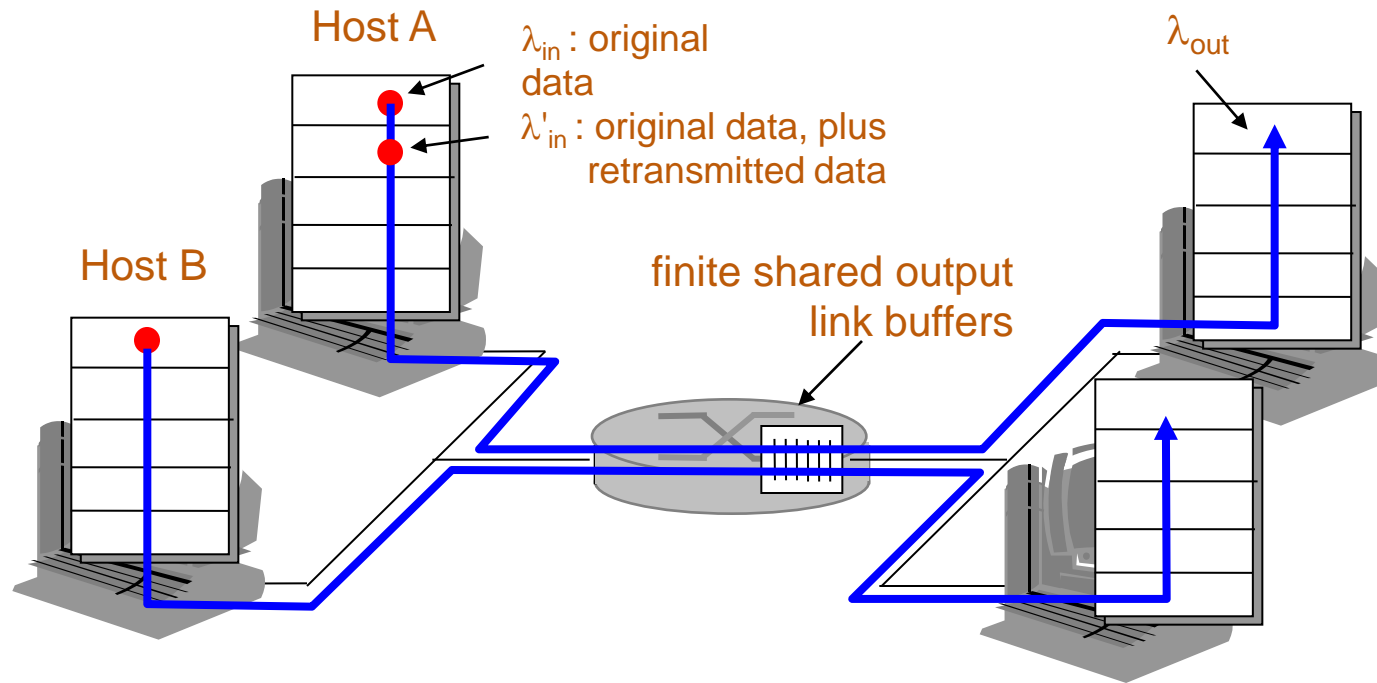
- two senders, two receivers
- one router, infinite buffers
- no retransmission



- large delays when congested
- maximum achievable throughput

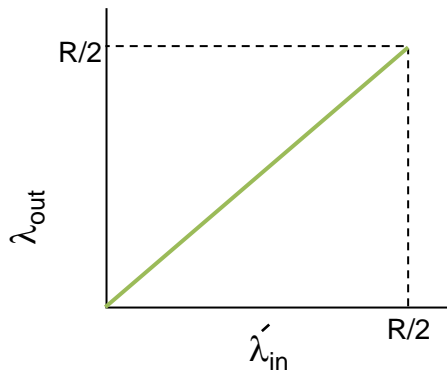
Causes/costs of congestion: scenario 2

- one router, *finite* buffers
- sender retransmission of lost packet

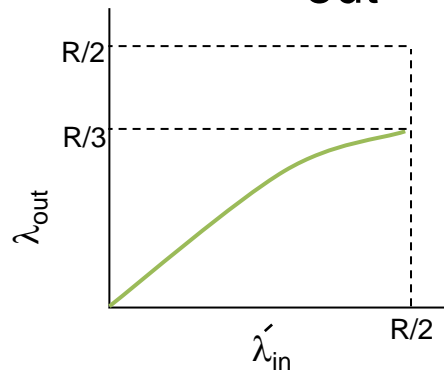


Causes/costs of congestion: scenario 2

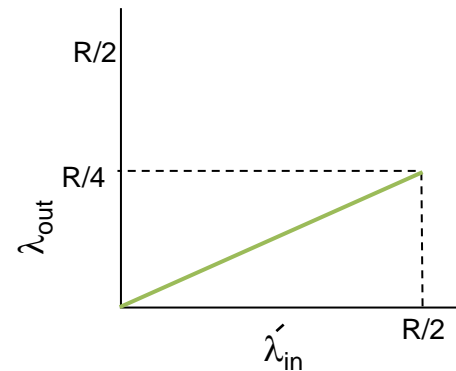
- always: $\lambda_{in} = \lambda_{out}$ (goodput)
- “perfect” retransmission only when loss: $\lambda'_{in} > \lambda_{out}$
- retransmission of delayed (not lost) packet makes λ'_{in} larger (than perfect case) for same λ_{out}



a.



b.



c.

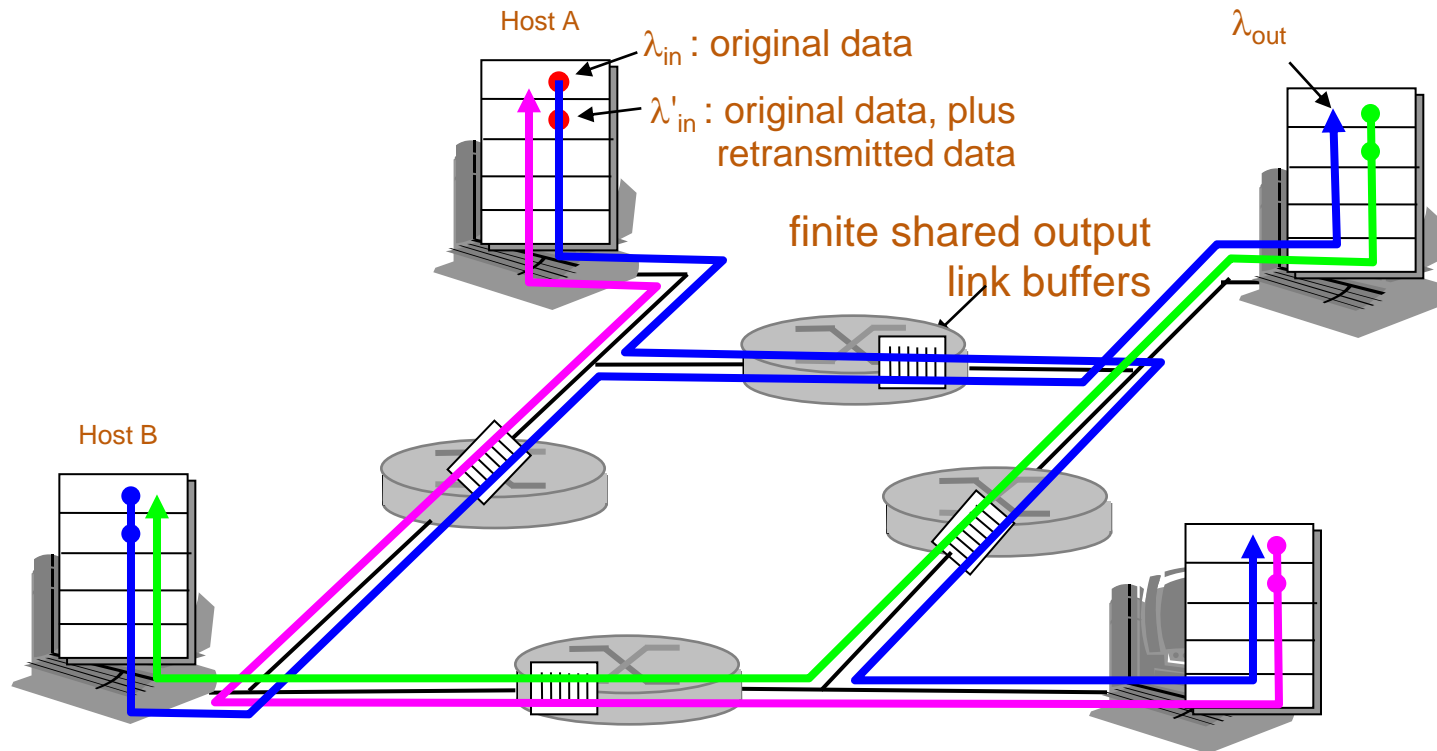
“costs” of congestion:

- more work (retrans) for given “goodput”
- unneeded retransmissions: link carries multiple copies of pkt

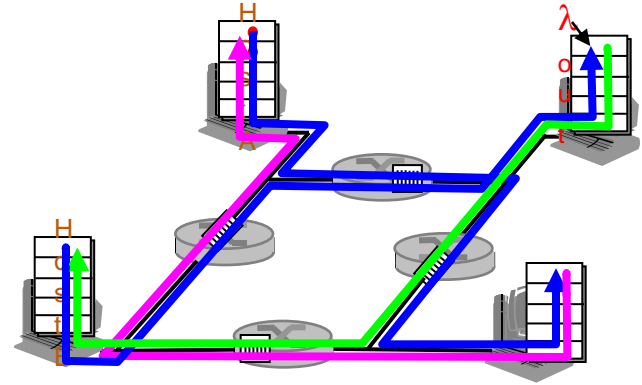
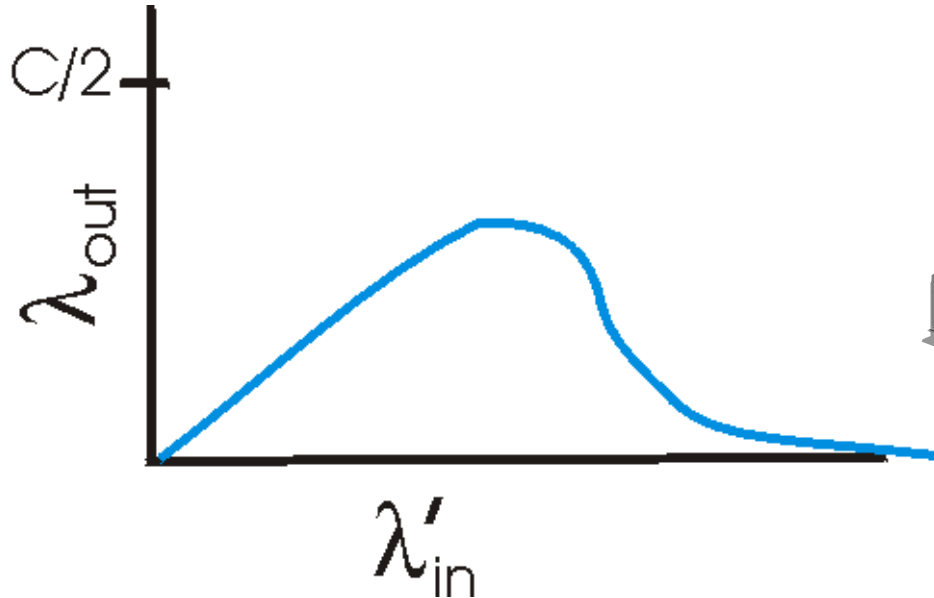
Causes/costs of congestion: scenario 3

- four senders
- multihop paths
- timeout/retransmit

Q: what happens as λ_{in}
and λ'_{in} increase ?



Causes/costs of congestion: scenario 3



Another “cost” of congestion:

- when packet dropped, any “upstream transmission capacity used for that packet was wasted!

Approaches towards congestion control

Two broad approaches towards congestion control:

End-end congestion control:

- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- approach taken by TCP

Network-assisted congestion control:

- routers provide feedback to end systems
 - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
 - explicit rate sender should send at

TCP Congestion Control

- end-end control (no network assistance)
- sender limits transmission:
 $\text{LastByteSent} - \text{LastByteAcked} \leq \text{CongWin}$
- Roughly,

$$\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}$$

- **CongWin** is dynamic, function of perceived network congestion

How does sender perceive congestion?

- loss event = timeout or 3 duplicate acks
- TCP sender reduces rate (**CongWin**) after loss event

three mechanisms:

- AIMD
- slow start
- conservative after timeout events

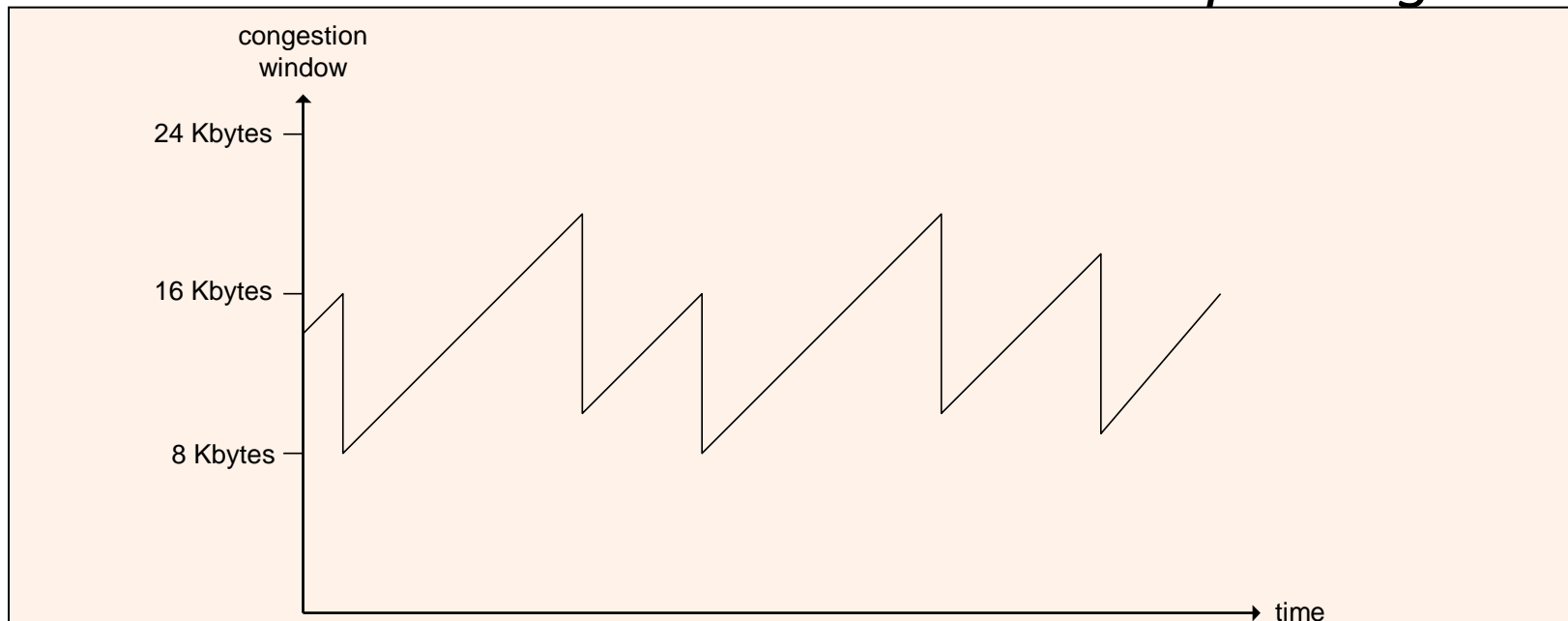
TCP AIMD

multiplicative decrease:

cut **CongWin** in half
after loss event

additive increase:

increase **CongWin** by
1 MSS every RTT in
the absence of loss
events: *probing*

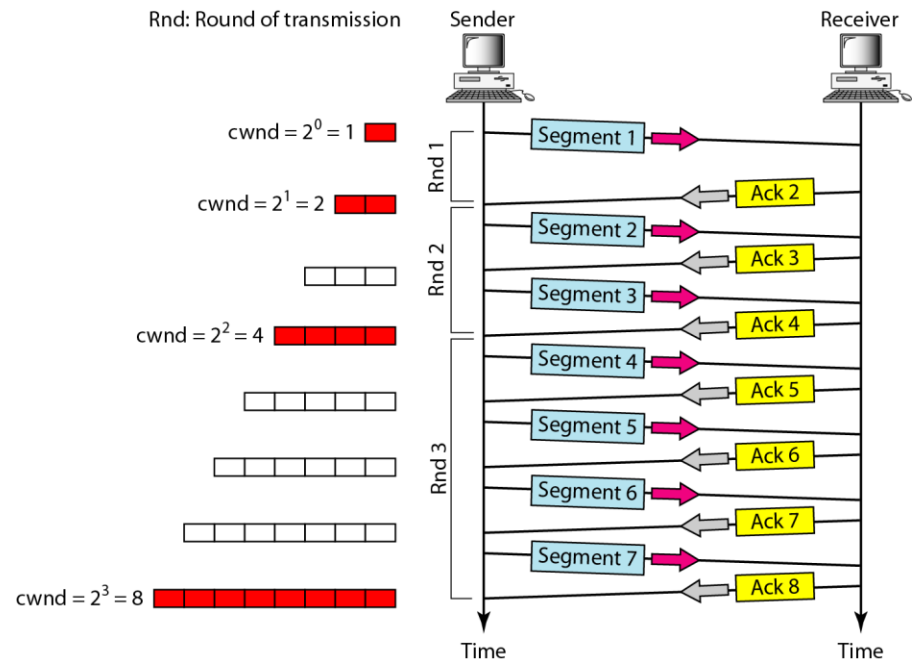


Long-lived TCP connection

TCP Slow Start

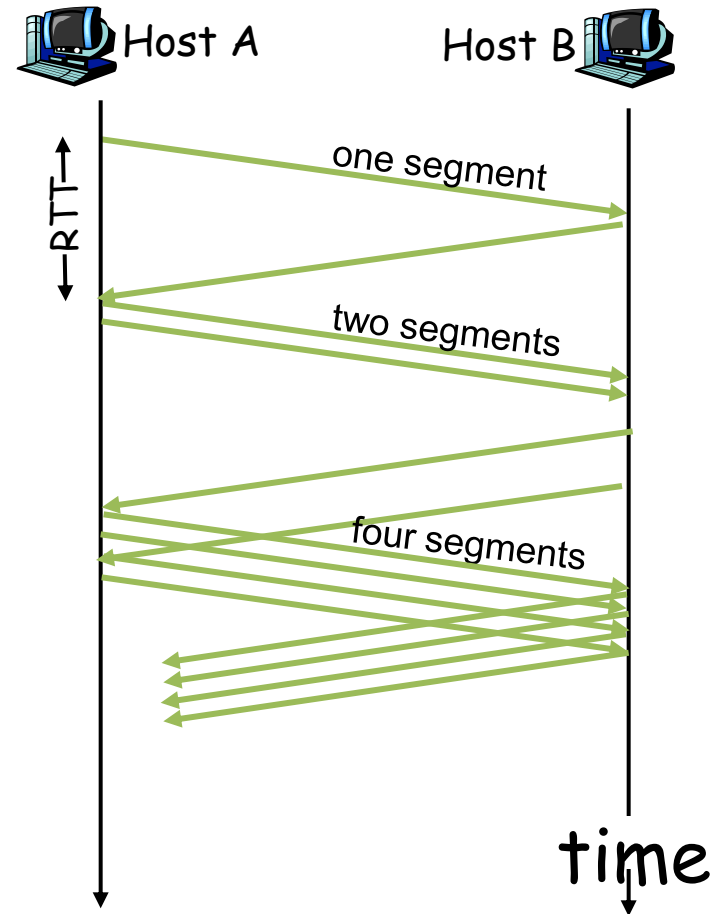
- When connection begins, **CongWin** = 1 MSS
 - Example: MSS = 500 bytes & RTT = 200 msec
 - initial rate = 20 kbps
- available bandwidth may be \gg MSS/RTT
 - desirable to quickly ramp up to respectable rate

- When connection begins, increase rate exponentially fast until first loss event



TCP Slow Start (more)

- When connection begins, increase rate exponentially until first loss event:
 - double **CongWin** every RTT
 - done by incrementing **CongWin** for every ACK received
- Summary: initial rate is slow but ramps up exponentially fast



Refinement

- After 3 dup ACKs:
 - **CongWin** is cut in half
 - window then grows linearly
- But after timeout event:
 - **CongWin** instead set to 1 MSS;
 - window then grows exponentially
 - to a threshold, then grows linearly

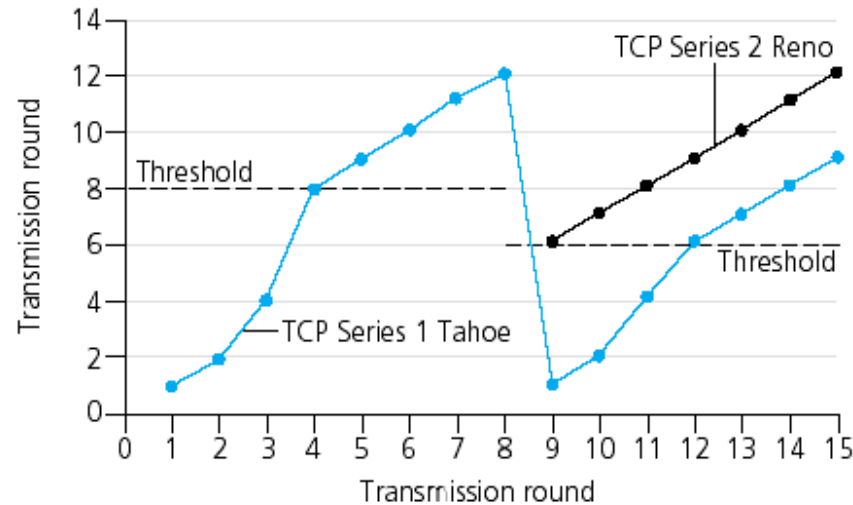
Philosophy:

- 3 dup ACKs indicates network capable of delivering some segments
- timeout before 3 dup ACKs is “more alarming”

Refinement (more)

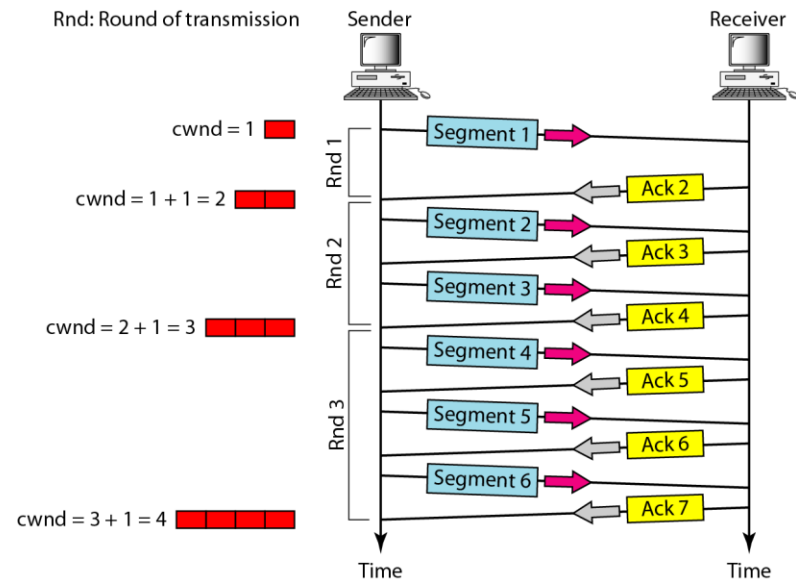
Q: When should the exponential increase switch to linear?

A: When **CongWin** gets to 1/2 of its value before timeout.



Implementation:

- Variable Threshold
- At loss event, Threshold is set to 1/2 of CongWin just before loss event



Summary: TCP Congestion Control

- When **CongWin** is below **Threshold**, sender in **slow-start** phase, window grows exponentially.
- When **CongWin** is above **Threshold**, sender is in **congestion-avoidance** phase, window grows linearly.
- When a **triple duplicate ACK** occurs, **Threshold** set to **CongWin/2** and **CongWin** set to **Threshold**.
- When **timeout** occurs, **Threshold** set to **CongWin/2** and **CongWin** is set to 1 MSS.

TCP sender congestion control

Event	State	TCP Sender Action	Commentary
ACK receipt for previously unacked data	Slow Start (SS)	$\text{CongWin} = \text{CongWin} + \text{MSS}$, If ($\text{CongWin} > \text{Threshold}$) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
ACK receipt for previously unacked data	Congestion Avoidance (CA)	$\text{CongWin} = \text{CongWin} + \text{MSS} * (\text{MSS} / \text{CongWin})$	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
Loss event detected by triple duplicate ACK	SS or CA	$\text{Threshold} = \text{CongWin} / 2$, $\text{CongWin} = \text{Threshold}$, Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
Timeout	SS or CA	$\text{Threshold} = \text{CongWin} / 2$, $\text{CongWin} = 1 \text{ MSS}$, Set state to "Slow Start"	Enter slow start
Duplicate ACK	SS or CA	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

