BM 402 Bilgisayar Ağları (Computer Networks)

M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Not: Bu dersin sunumları, ders kitabının yazarları James F. Kurose ve Keith W. Ross tarafından sağlanan sunumlar üzerinde değişiklik yapılarak hazırlanmıştır.

Ders konuları

- Congestion Control
- TCP Congestion Control

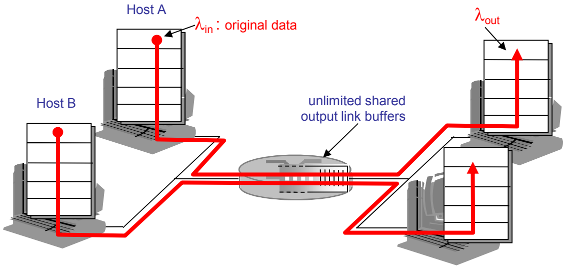
Principles of Congestion Control

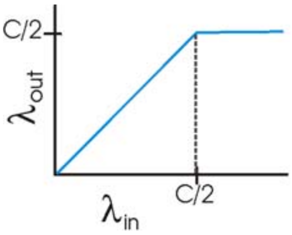
Congestion:

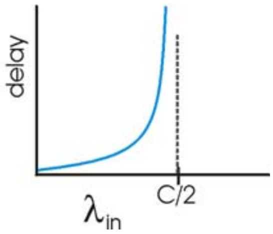
- informally: “too many sources sending too much data too fast for *network* to handle”
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!

Causes/costs of congestion: scenario 1

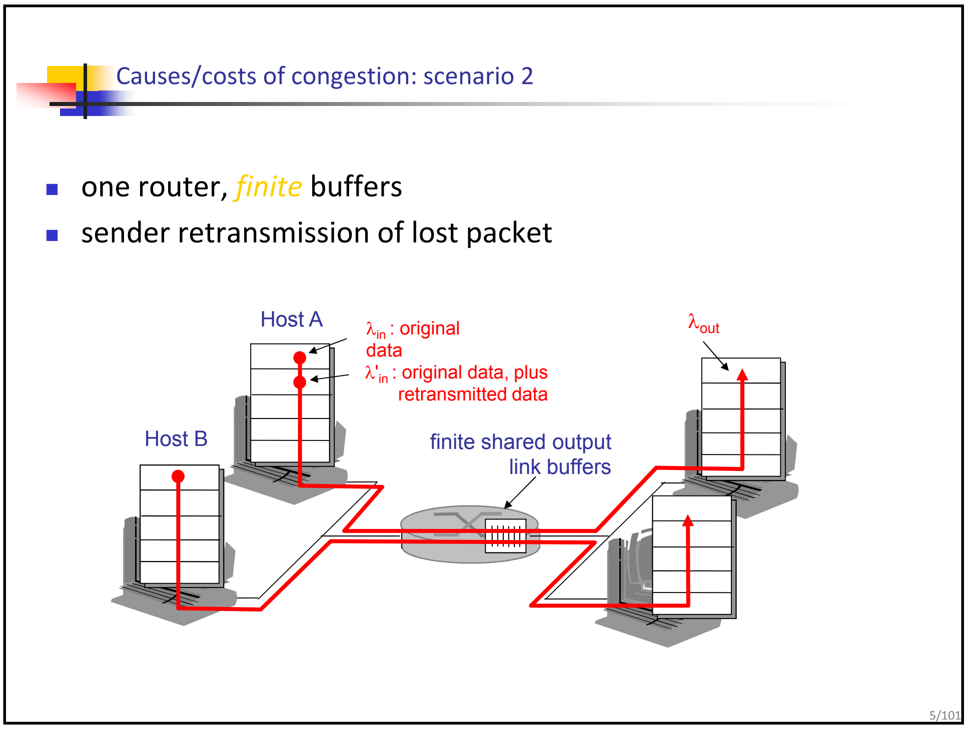
- two senders, two receivers
- one router, infinite buffers
- no retransmission


Host A
λ_{in} : original data
λ_{out}
Host B
unlimited shared output link buffers





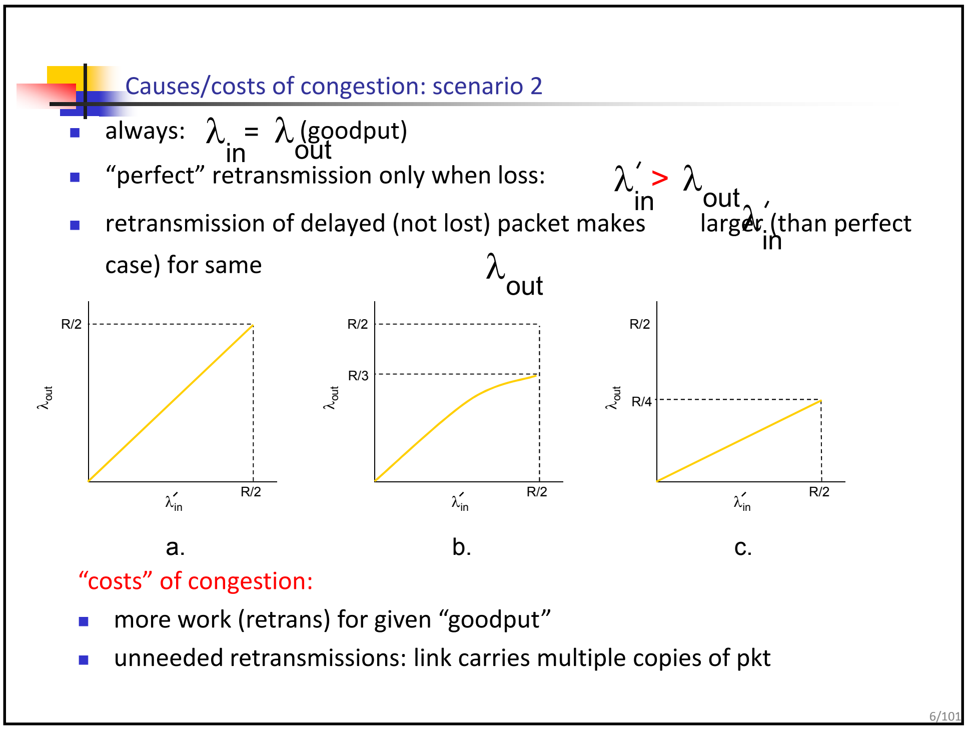
- large delays when congested
- maximum achievable throughput



Causes/costs of congestion: scenario 2

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

Host A
λ_{in} : original data
λ'_{in} : original data, plus retransmitted data
λ_{out}
Host B
finite shared output link buffers



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}

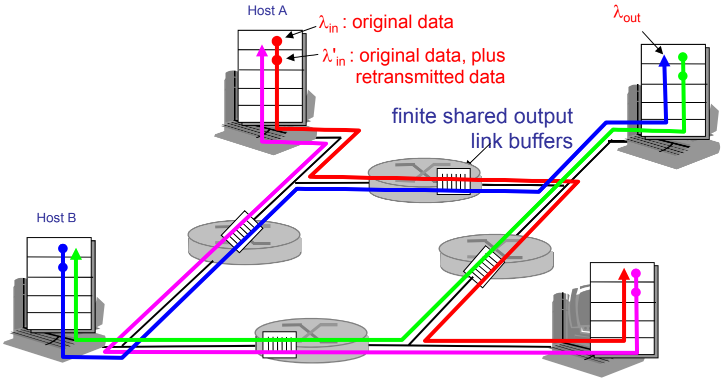
"costs" of congestion:

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

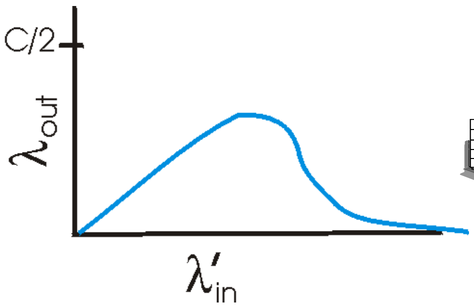
Causes/costs of congestion: scenario 3

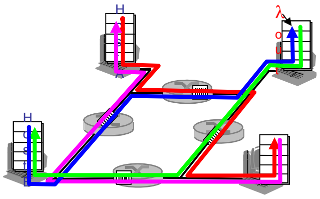
- four senders
- multihop paths
- timeout/retransmit

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


Host A
λ_{in} : original data
λ'_{in} : original data, plus retransmitted data
λ_{out}
finite shared output link buffers
Host B

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

Case study: ATM ABR congestion control

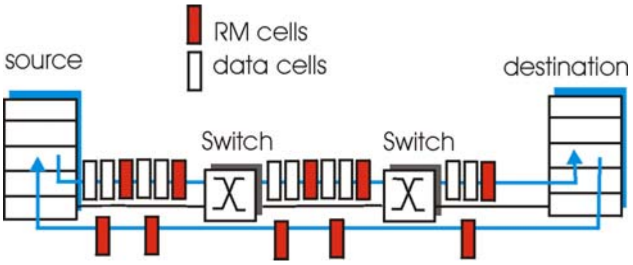
ABR: available bit rate:

- "elastic service"
- if sender's path "underloaded":
  - sender should use available bandwidth
- if sender's path congested:
  - sender throttled to minimum guaranteed rate

RM (resource management) cells:

- sent by sender, interspersed with data cells
- bits in RM cell set by switches ("*network-assisted"*)
  - NI bit: no increase in rate (mild congestion)
  - CI bit: congestion indication
- RM cells returned to sender by receiver, with bits intact

Case study: ATM ABR congestion control



- two-byte ER (explicit rate) field in RM cell
  - congested switch may lower ER value in cell
  - sender’ send rate thus minimum supportable rate on path
- EFCI bit in data cells: set to 1 in congested switch
  - if data cell preceding RM cell has EFCI set, sender sets CI bit in returned RM cell

Ders konuları

- Congestion Control
- TCP Congestion Control

TCP Congestion Control

- end-end control (no network assistance)
- sender limits transmission:

  LastByteSent-LastByteAcked
  ≤ CongWin

- Roughly,

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

- **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

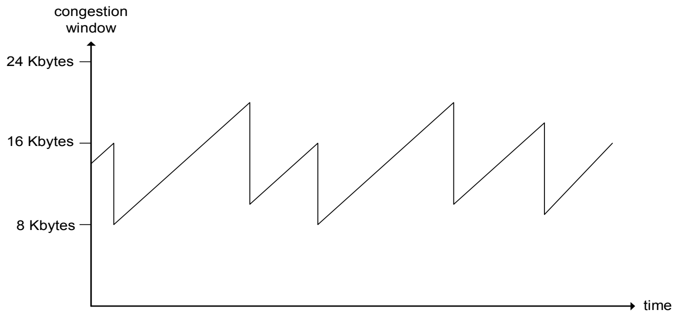
three mechanisms:

- 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

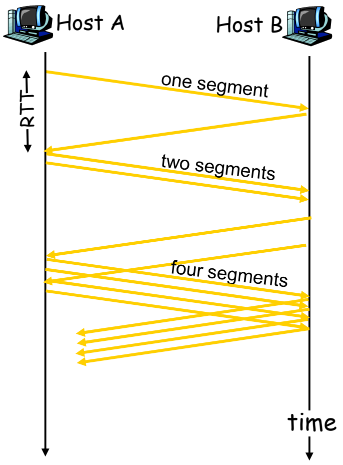
TCP Slow Start

- When connection begins, **`CongWin`** = 1 MSS
  - Example: MSS = 500 bytes & RTT = 200 msec
  - initial rate = 20 kbps
- available bandwidth may be >> 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

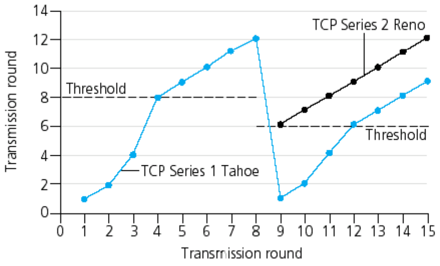
Philosophy:

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

Refinement (more)

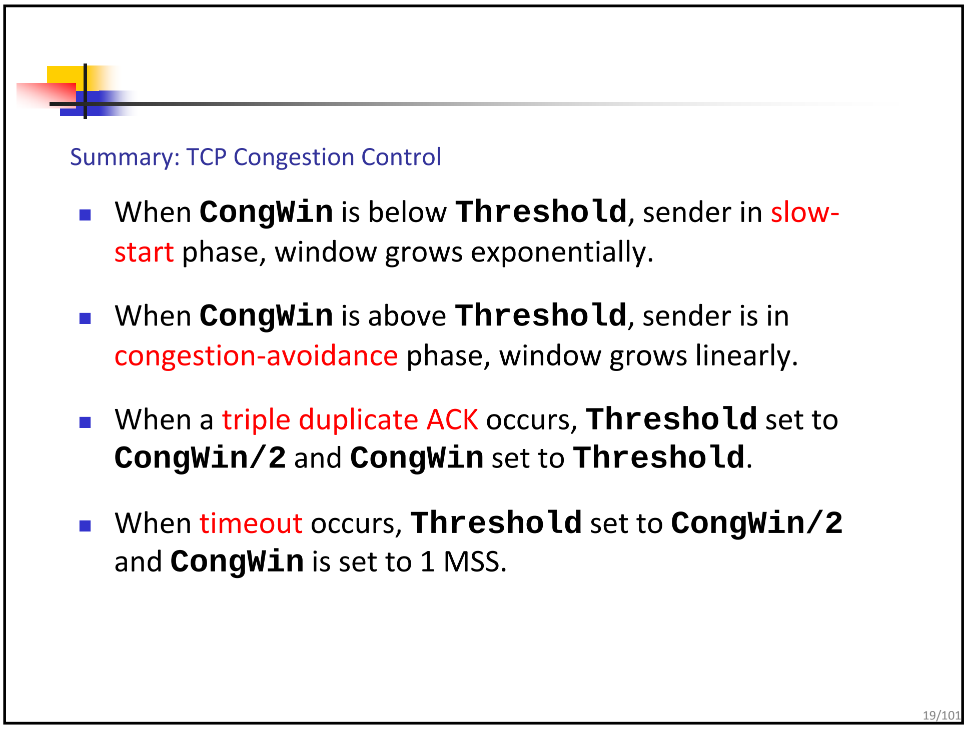
Q: When should the exponential increase switch to linear?

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



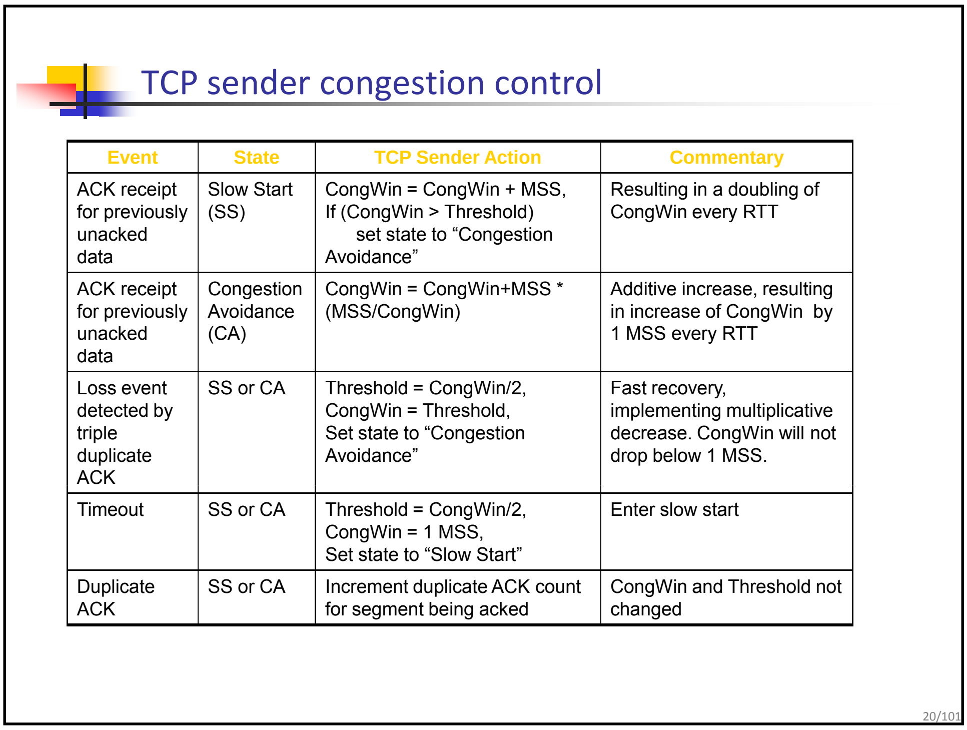
Implementation:

- 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.



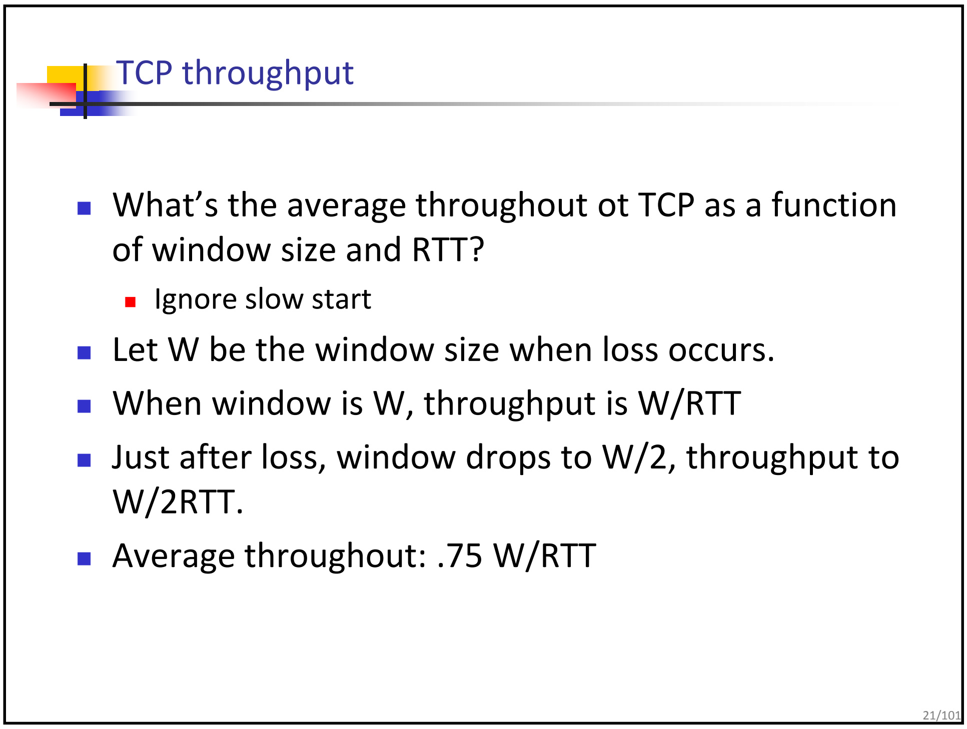
TCP sender congestion control

Event	State	TCP Sender Action	Commentary
ACK receipt for previously unacked data	Slow Start (SS)	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
ACK receipt for previously unacked data	Congestion Avoidance (CA)	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT

Loss event detected by triple duplicate ACK	SS or CA	Threshold = CongWin/2,
		CongWin = Threshold,
		Set state to "Congestion Avoidance"
			Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.

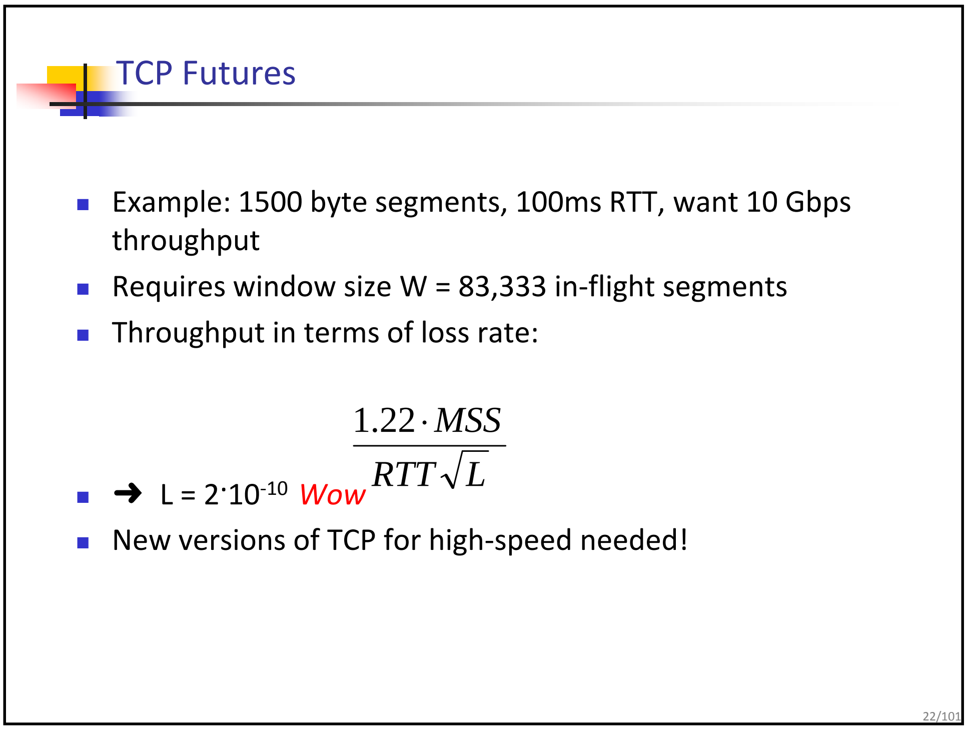
Timeout	SS or CA	Threshold = CongWin/2,
		CongWin = 1 MSS,
		Set state to "Slow Start"
			Enter slow start

Duplicate ACK	SS or CA	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed



TCP throughput

- What’s the average throughout ot TCP as a function of window size and RTT?
  - Ignore slow start
- Let W be the window size when loss occurs.
- When window is W, throughput is W/RTT
- Just after loss, window drops to W/2, throughput to W/2RTT.
- Average throughout: .75 W/RTT

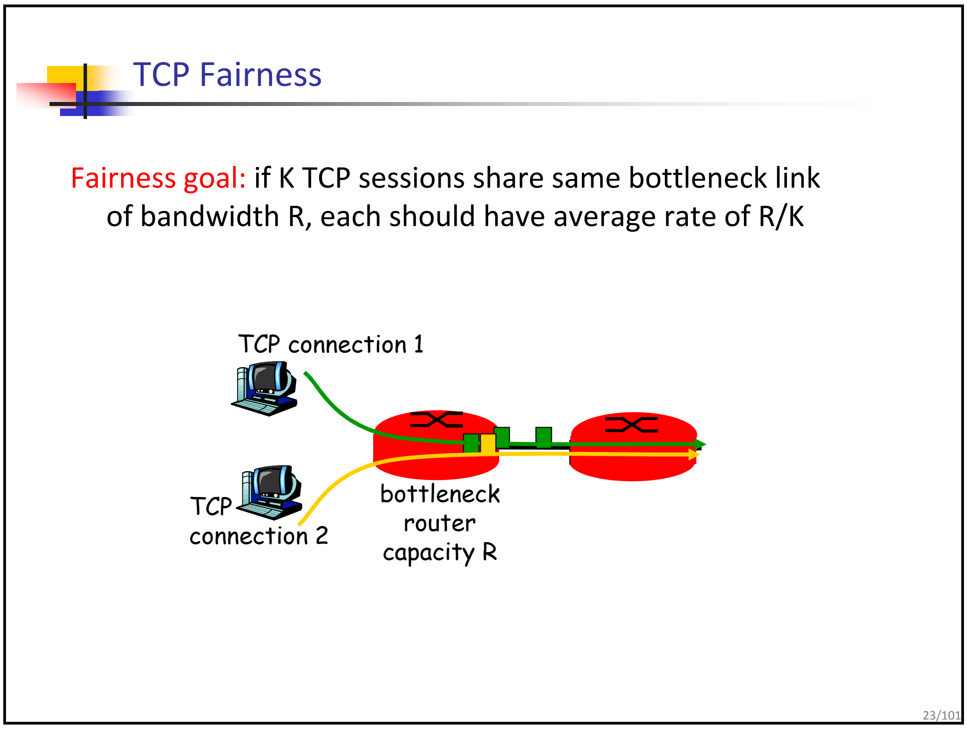


TCP Futures

- Example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- Requires window size W = 83,333 in-flight segments
- Throughput in terms of loss rate:

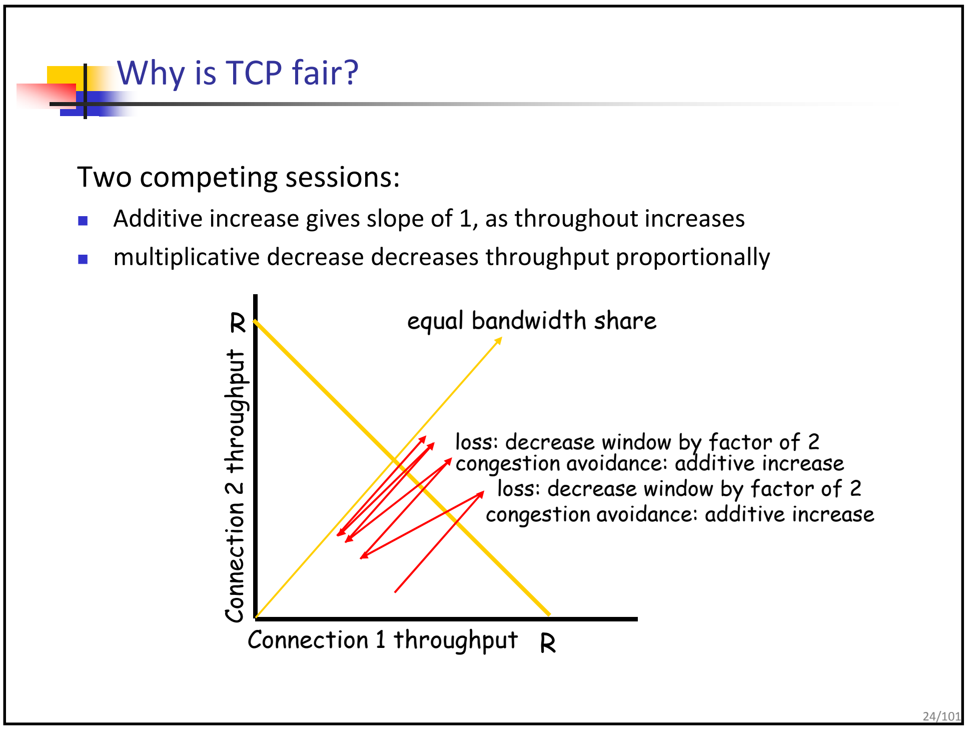
$$\frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

- ➜ L = $2 \cdot 10^{-10}$ *Wow*
- New versions of TCP for high-speed needed!



TCP Fairness

Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Why is TCP fair?

Two competing sessions:

- Additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally

Fairness (more)

Fairness and UDP

- Multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- Instead use UDP:
  - pump audio/video at constant rate, tolerate packet loss
- Research area: TCP friendly

Fairness and parallel TCP connections

- nothing prevents app from opening parallel cnctions between 2 hosts.
- Web browsers do this
- Example: link of rate R supporting 9 cnctions;
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2 !

Delay modeling

Q: How long does it take to receive an object from a Web server after sending a request?

Ignoring congestion, delay is influenced by:

- TCP connection establishment
- data transmission delay
- slow start

Notation, assumptions:

- Assume one link between client and server of rate R
- S: MSS (bits)
- O: object size (bits)
- no retransmissions (no loss, no corruption)

Window size:

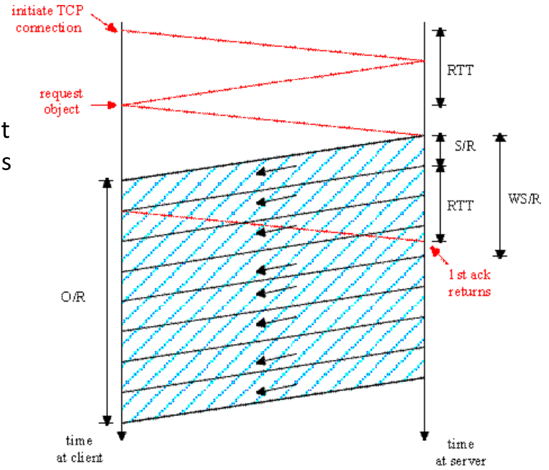
- First assume: fixed congestion window, W segments
- Then dynamic window, modeling slow start

Fixed congestion window (1)

First case:

WS/R > RTT + S/R: ACK for first segment in window returns before window's worth of data sent

delay = 2RTT + O/R

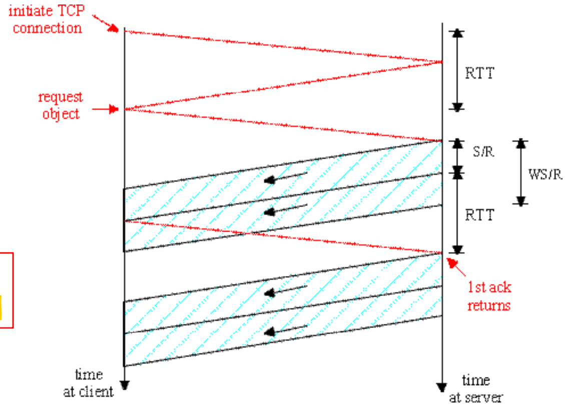


Fixed congestion window (2)

Second case:

- WS/R < RTT + S/R: wait for ACK after sending window's worth of data sent

delay = 2RTT + O/R + (K-1)[S/R + RTT - WS/R]



TCP Delay Modeling: Slow Start (1)

Now suppose window grows according to slow start

Will show that the delay for one object is:

$$\boxed{Latency = 2RTT + \frac{O}{R} + P\left[RTT + \frac{S}{R}\right] - (2^P - 1)\frac{S}{R}}$$

where P is the number of times TCP idles at server:

$$P = \min\{Q, K-1\}$$

- where Q is the number of times the server idles if the object were of infinite size.

- and K is the number of windows that cover the object.

TCP Delay Modeling: Slow Start (2)

Delay components:

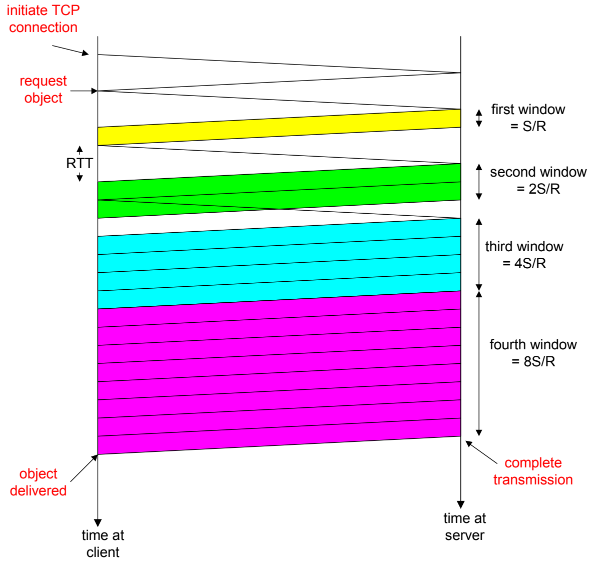
- 2 RTT for connection estab and request
- O/R to transmit object
- time server idles due to slow start

Server idles:
P = min{K-1,Q} times

Example:

- O/S = 15 segments
- K = 4 windows
- Q = 2
- P = min{K-1,Q} = 2

Server idles P=2 times



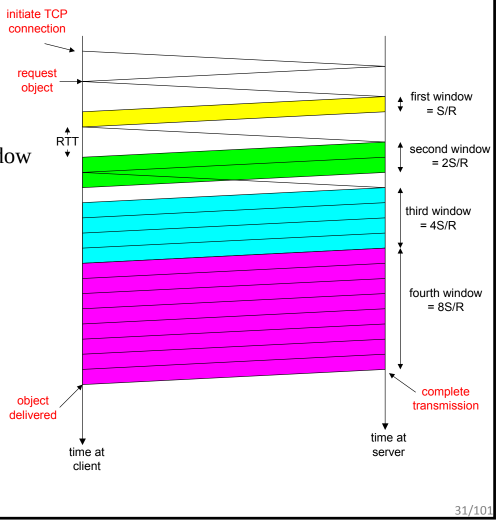
TCP Delay Modeling (3)

$\frac{S}{R} + RTT$ = time from when server starts to send segment until server receives acknowledgement

$2^{k-1}\frac{S}{R}$ = time to transmit the kth window

$\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right]^{+}$ = idle time after the kth window

$$\begin{aligned}\text{delay} &= \frac{O}{R} + 2RTT + \sum_{p=1}^{P} idleTime_p \\ &= \frac{O}{R} + 2RTT + \sum_{k=1}^{P}\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right] \\ &= \frac{O}{R} + 2RTT + P\left[RTT + \frac{S}{R}\right] - (2^{P}-1)\frac{S}{R}\end{aligned}$$



TCP Delay Modeling (4)

Recall K = number of windows that cover object

How do we calculate K ?

$$\begin{aligned}K &= \min\{k : 2^0 S + 2^1 S + \cdots + 2^{k-1} S \geq O\} \\ &= \min\{k : 2^0 + 2^1 + \cdots + 2^{k-1} \geq O/S\} \\ &= \min\{k : 2^k - 1 \geq \frac{O}{S}\} \\ &= \min\{k : k \geq \log_2(\frac{O}{S} + 1)\} \\ &= \left\lceil \log_2(\frac{O}{S} + 1) \right\rceil\end{aligned}$$

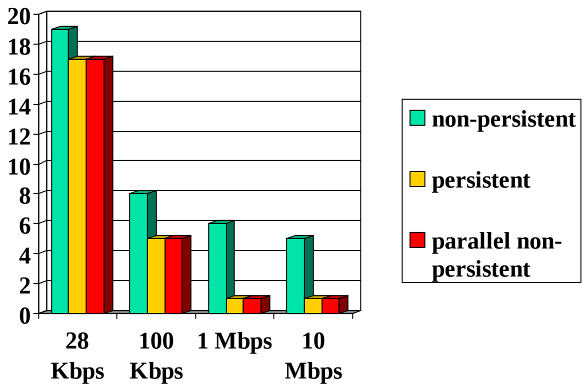
Calculation of Q, number of idles for infinite-size object, is similar (see HW).

HTTP Modeling

- Assume Web page consists of:
  - *1* base HTML page (of size *O* bits)
  - *M* images (each of size *O* bits)
- Non-persistent HTTP:
  - *M+1* TCP connections in series
  - *Response time = (M+1)O/R + (M+1)2RTT + sum of idle times*
- Persistent HTTP:
  - *2 RTT* to request and receive base HTML file
  - *1 RTT* to request and receive M images
  - *Response time = (M+1)O/R + 3RTT + sum of idle times*
- Non-persistent HTTP with X parallel connections
  - Suppose M/X integer.
  - 1 TCP connection for base file
  - M/X sets of parallel connections for images.
  - *Response time = (M+1)O/R + (M/X + 1)2RTT + sum of idle times*

HTTP Response time (in seconds)

RTT = 100 msec, O = 5 Kbytes, M=10 and X=5

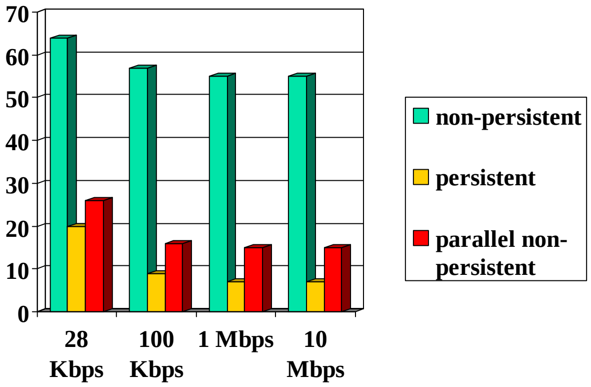


For low bandwidth, connection & response time dominated by transmission time.

Persistent connections only give minor improvement over parallel connections.

HTTP Response time (in seconds)

RTT =1 sec, O = 5 Kbytes, M=10 and X=5



For larger RTT, response time dominated by TCP establishment & slow start delays. Persistent connections now give important improvement: particularly in high delay • bandwidth networks.