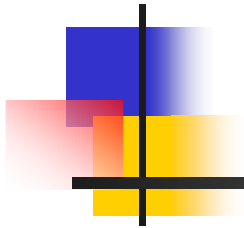


Congestion Control



Dept. of Computer Science, University of Rochester



What is good about UDP?

- TCP features may not be needed by some applications
- less overhead:
 - no connection establishment (which can add delay)
 - no connection state at sender, receiver (which add memory space requirement)
 - small segment header
 - no congestion control: UDP can blast away as fast as desired (may not be good for others, but not bad for my own connection 😊)



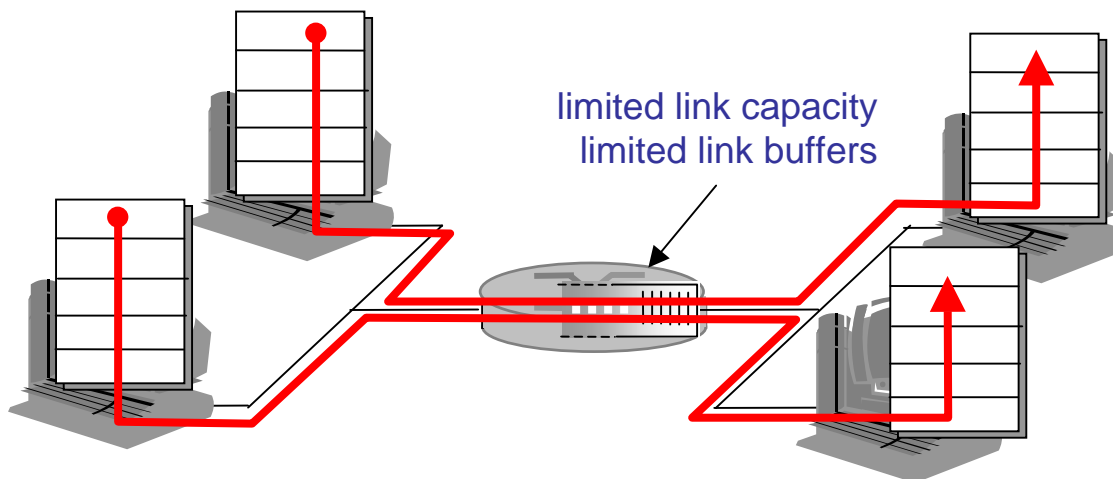
UDP: More

- Often used for streaming multimedia apps
 - loss tolerant
 - rate sensitive
- In general, UDP is used when TCP features are not important
- What if you want features not in UDP while the full TCP is overkill?
 - implemented at application-level
 - flow control and loss recovery in some multimedia apps

Principles of Congestion Control

Congestion:

- informally: "too many sources sending too much data too fast for the network to handle"
- results of congestion:
 - long delays (e.g. queueing in router buffers)
 - lost packets (e.g. buffer overflow at routers)





Congestion Control

Congestion threatens the growth of the Internet

- tragedy of commons

Two broad approaches towards congestion control:

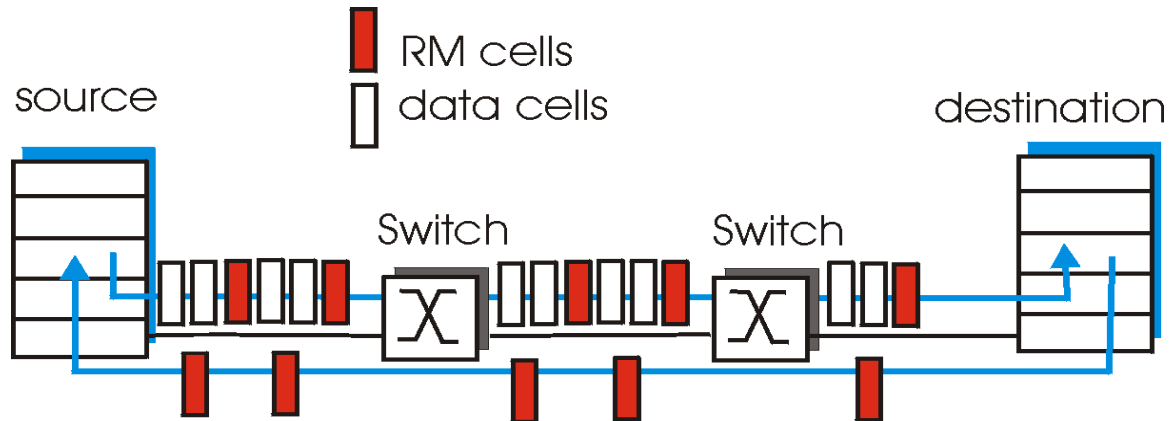
Network-assisted congestion control:

- routers provide feedback to end systems
 - mark bit indicating congestion
 - explicit rate sender should send at

End-end congestion control:

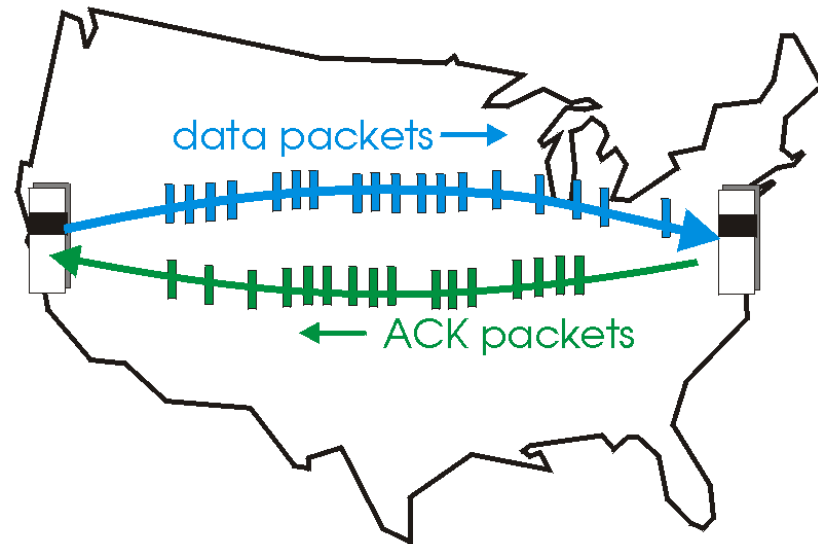
- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- approach taken by TCP

Network-assisted Congestion Control: **ATM**



- **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
 - **ER (explicit rate) field:** congested switch may lower ER value in cell
- RM cells returned to sender by receiver, with bits intact
 - sender control its rate based on information received in RM cells

TCP Congestion Control



- **Mechanism:** sender controls the sending rate by adjusting the number of packets allowed in flight simultaneously (size of the sliding window)
- **Objective:** figure out the highest rate that does not cause network congestion

TCP Congestion Control

- **CongWin** is maximum allowed in-flight or un-ACKed bytes
- Roughly:
 - $\text{throughput} = \text{CongWin} / \text{RTT}$
- dynamically control **CongWin**, based on perceived network congestion

How does sender perceive congestion?

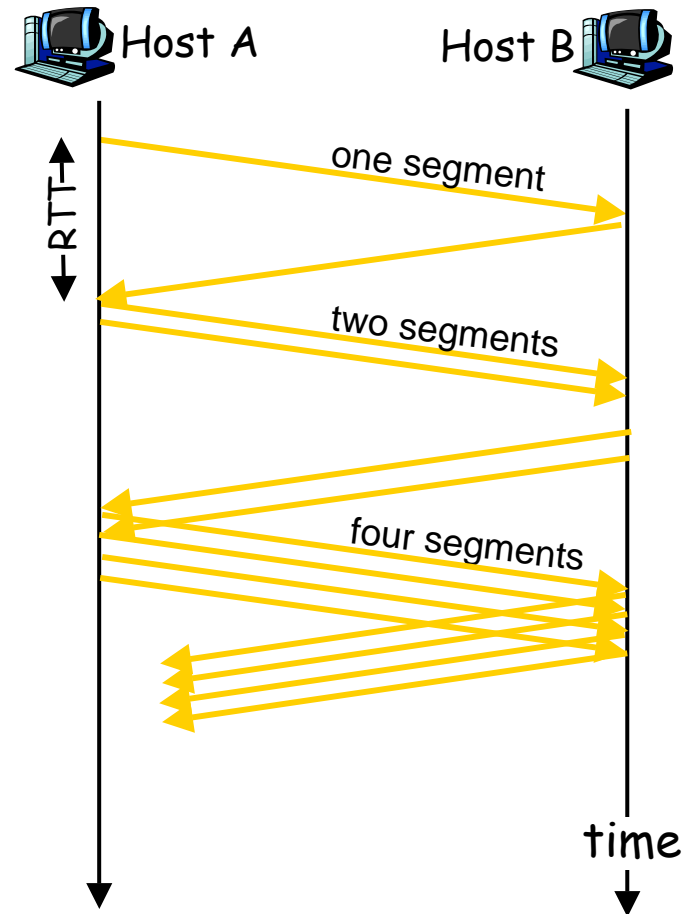
- loss event = timeout *or* triple duplicate ACKs
- timeout before 3 dup ACKs is **more alarming**
 - 3 dup ACKs indicates network capable of delivering some segments

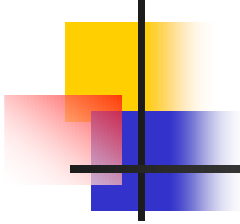
two phases for TCP congestion control:

- slow start phase
- steady state, or congestion avoidance phase

Slow Start Phase

- At the beginning, $\text{CongWin} = 1 \text{ MSS}$
- Available bandwidth may be $\gg \text{MSS}/\text{RTT}$
 - Example: $\text{MSS} = 500 \text{ bytes}$ & $\text{RTT} = 200 \text{ msec}$
 - \Rightarrow initial rate = 20 kbps
 - desirable to quickly ramp up to respectable rate
- **Exponential increase:** double CongWin after every RTT in the absence of loss events
- In this phase, the CongWin starts at a slow rate, but it grows fast
- When should we stop exponential growth?





When does Slow Start become Steady State?

- Answer:
 - when there is a loss event
 - or when CongWin reaches a Threshold.
- **Threshold** divides between two states
 - when you are far away from causing congestion, so you do not need to worry about congestion
 - when you are very close to causing congestion, do need to be very careful
- **Threshold** is maintained dynamically:
 - initially set to a large value so it has no effect;
 - set to half of CongWin at a loss event

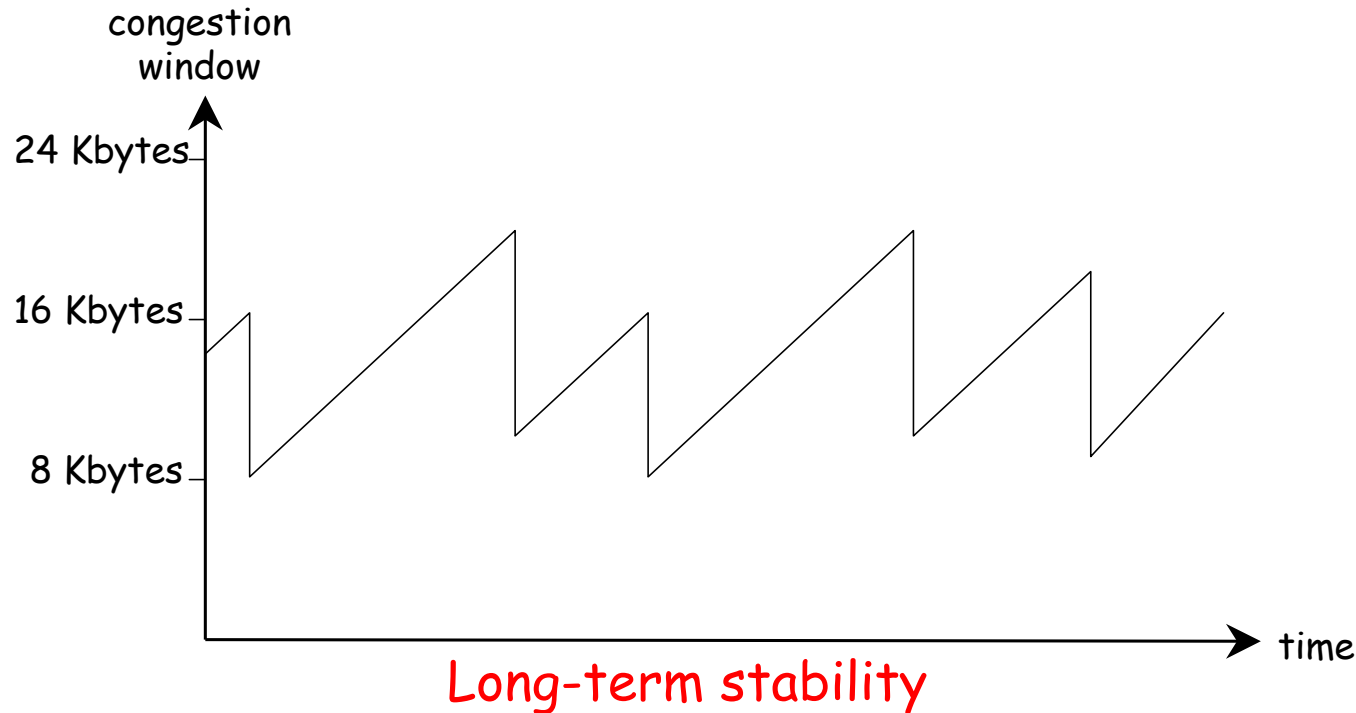
Steady State, Congestion Avoidance Phase: AIMD

Additive increase:

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

Multiplicative decrease:

cut CongWin in half after 3 duplicate ACKs





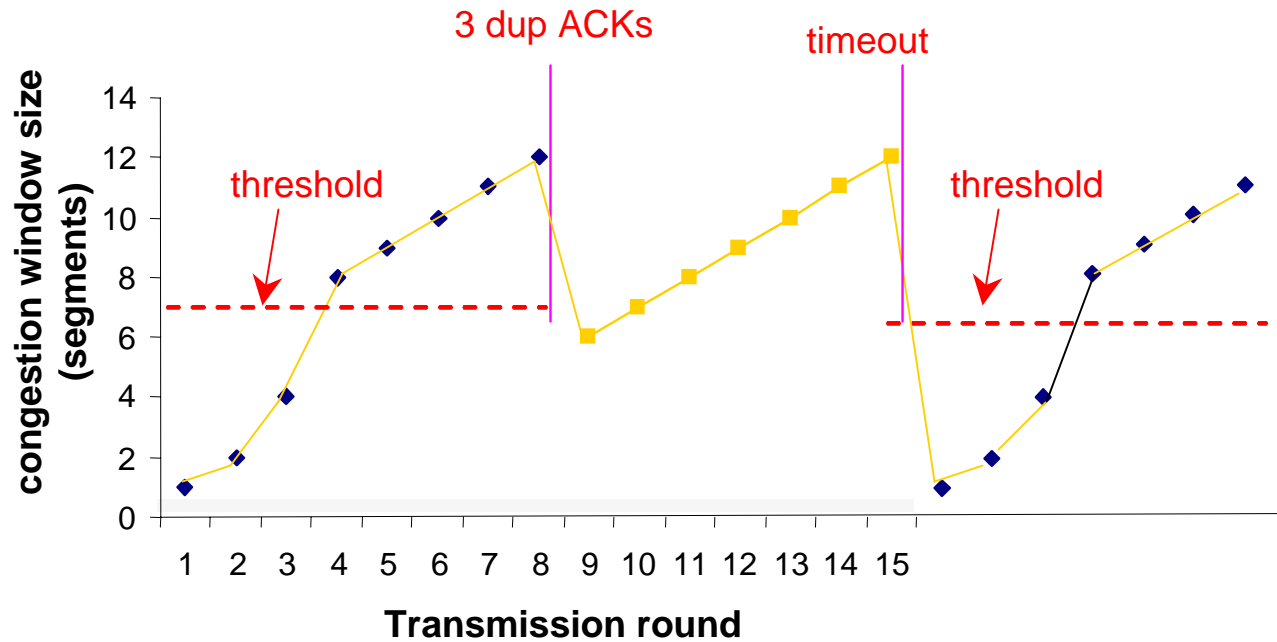
Handling Loss Events

- After 3 dup ACKs:
 - CongWin is cut in half, it then grows linearly
- But after timeout event:
 - CongWin instead set to 1 MSS, fall back to slow start
- the very beginning of the connection

Philosophy:

timeout before 3 dup ACKs is **more alarming** because 3 dup ACKs indicates network capable of delivering some segments

TCP in Action





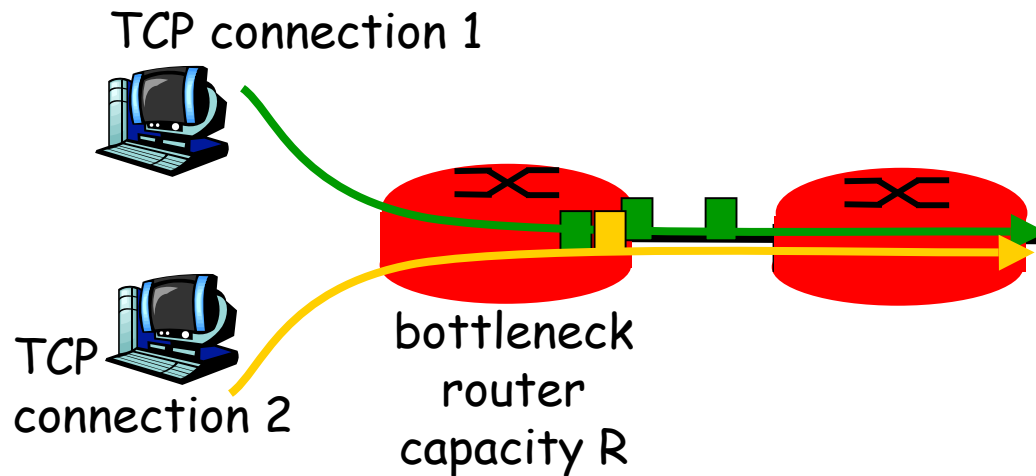
Outline

- principles of congestion control
- congestion control in TCP
- impact of congestion control on fairness
- impact of congestion control on TCP efficiency

TCP Fairness

Fairness goal:

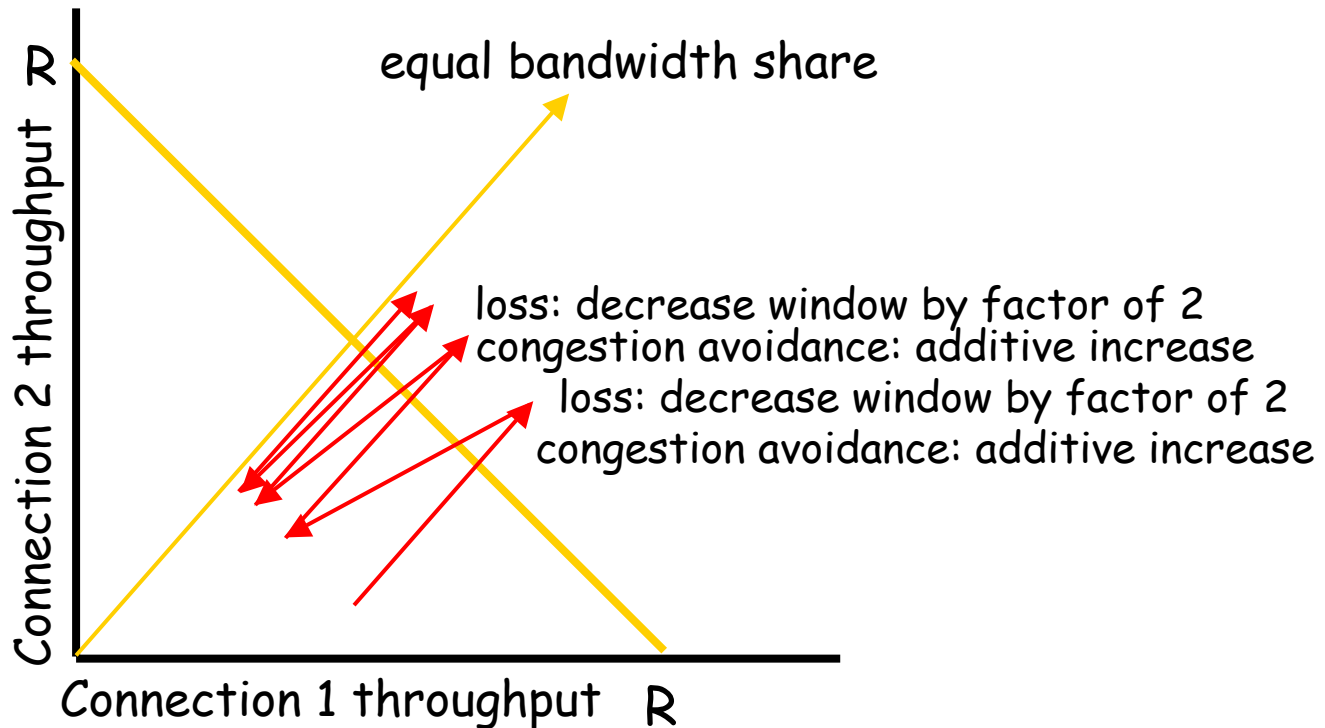
if K TCP sessions share same bottleneck link of bandwidth R , each should have average rate of R/K



AIMD is Fair

Two competing sessions:

- Additive increase gives slope of 45° , as throughputs of both sessions increase at the same speed
- multiplicative decrease decreases throughput proportionally





More on Fairness

Fairness with UDP

- Multimedia apps often do not use TCP
 - do not want rate controlled by congestion control
- Instead use UDP:
 - pump audio/video at constant rate, tolerate packet loss
- TCP is network friendly while UDP is not

Fairness with parallel TCP connections

- Nothing prevents app from opening parallel connections between 2 hosts.
- Web browsers do this
- Example: link of rate R currently supporting 9 connections:
 - new app uses 1 TCP connection, gets rate $R/10$
 - new app asks for 9 TCP connections, gets $R/2$!



TCP Efficiency

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

- TCP connection establishment
- data transmission delay
- congestion control (slow start)

Case study:

- first assume fixed congestion window: W segments
- more complicated with full TCP congestion control

Fixed Congestion Window (1)

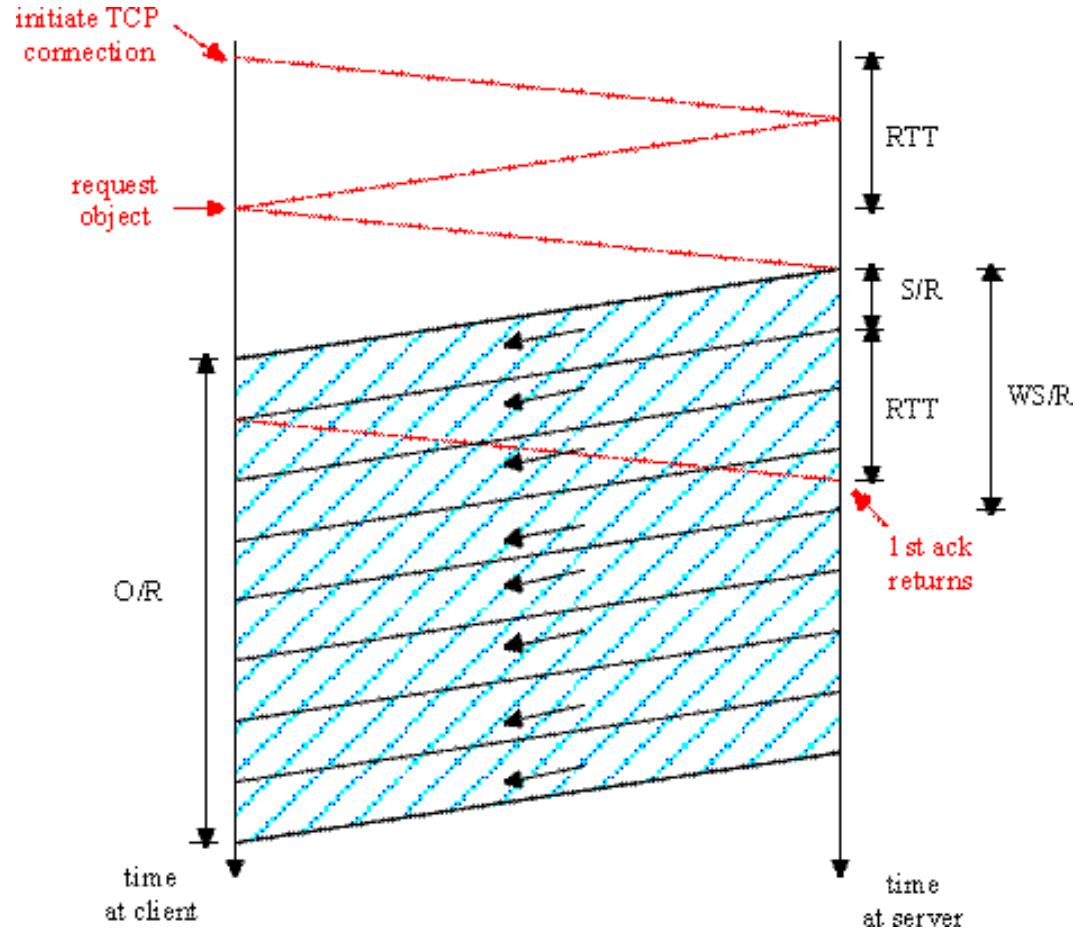
Notation, assumptions:

- Link rate: R
- Object size: O
- Fixed window size: WS
- No loss, no corruption

Case 1: congestion window is large

- ACK for first segment in window returns before window's worth of data sent

$$\text{delay} = 2RTT + O/R$$



Fixed Congestion Window (2)

Notation, assumptions:

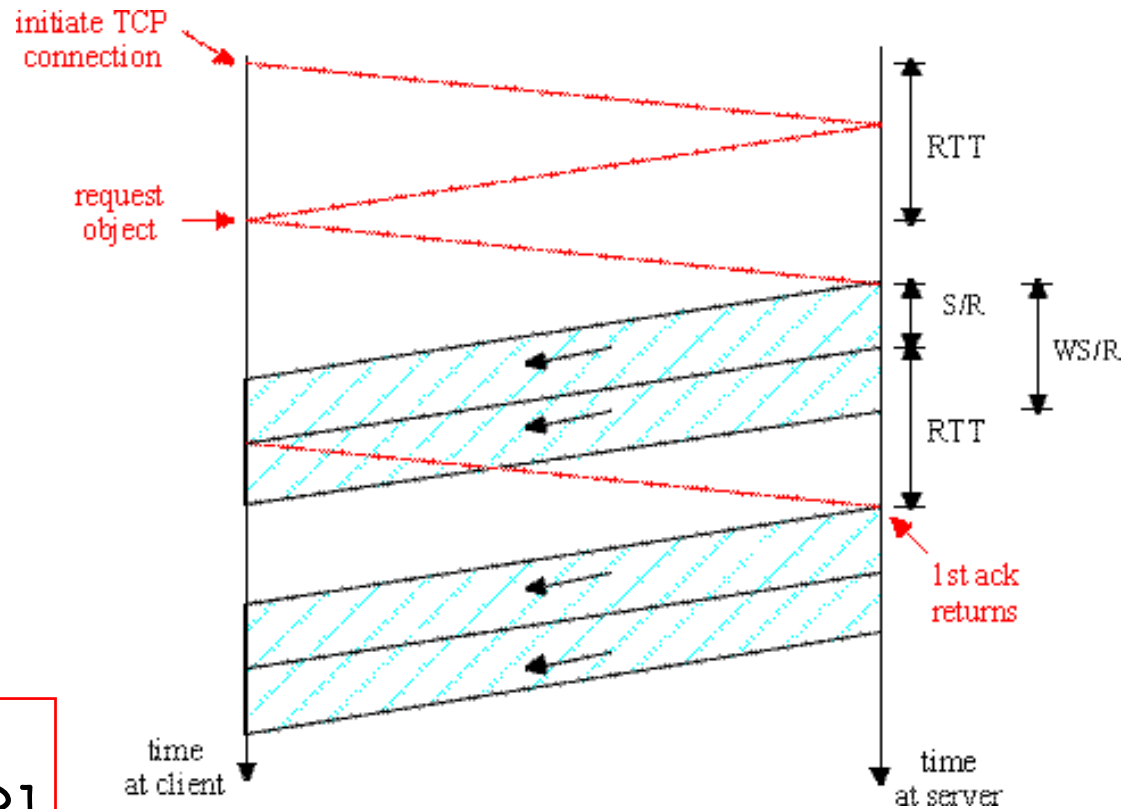
- Link rate: R
- Object size: O
- Fixed window size: WS
- No loss, no corruption

Case 2: congestion window is not large

- wait for ACK after sending window's worth of data sent

$$\text{delay} = 2RTT + O/R + (K-1)[S/R + RTT - WS/R]$$

$$\text{where } K = O/WS$$





TCP Efficiency with Slow Start

- small window size in the slow start process \Rightarrow inefficiency
- inefficiency is amortized over long-running connections
 - persistent connection in HTTP/1.1



Summary: TCP Congestion Control

- At **slow-start** phase, window starts at 1 MSS and grows exponentially.
- When CongWin is above Threshold, sender is in **congestion-avoidance** phase, window grows linearly.
- When a **triple duplicate ACK** occurs, CongWin is cut in half.
- When **timeout** occurs, Threshold set to CongWin/2 and CongWin is set to 1 MSS (**slow start** again).

This is the **Reno** version of TCP congestion control, an upgrade from an earlier version called **Tahoe**.

A newer version called **Vegas** includes some optimizations, but not yet widely used.



Disclaimer

- Parts of the lecture slides contain original work of James Kurose, Larry Peterson, and Keith Ross. The slides are intended for the sole purpose of instruction of computer networks at the University of Rochester. All copyrighted materials belong to their original owner(s).