CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Course on Computer Communication and Networks

Lecture 5
Chapter 3; Transport Layer, Part B

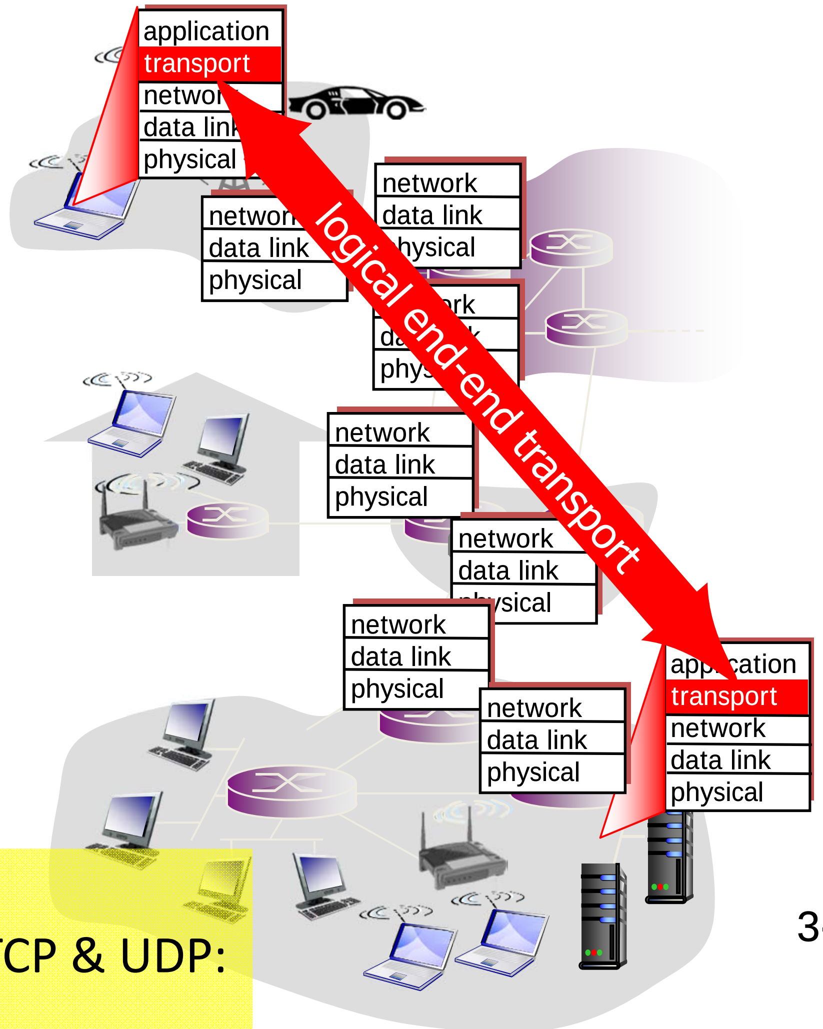
EDA344/DIT 420, CTH/GU

Based on the book Computer Networking: A Top Down Approach, Jim Kurose, Keith Ross, Addison-Wesley.

Internet transport-layer protocols

- reliable, in-order delivery: **TCP**; also provides
  - flow control
  - congestion control
  - connection setup

- unreliable, unordered delivery: **UDP**
  - no-frills extension of "best-effort" IP



Both support addressing (multiplexing)
Transport Layer services **not available** in TCP & UDP:
Delay, bandwidth guarantees

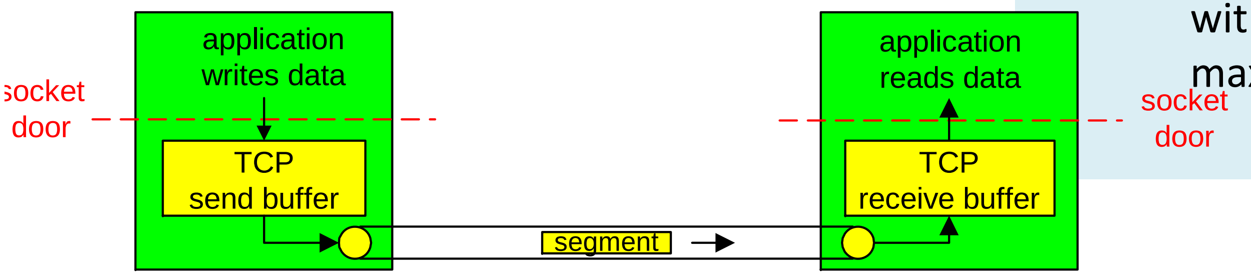
Roadmap Transport Layer

- transport layer services
- multiplexing/demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
  - reliable transfer
    - Acknowledgements
    - Retransmissions
    - Connection management
    - Flow control and buffer space
  - Congestion control
    - Principles
    - TCP congestion control

TCP: Overview RFCs: 793,1122,1323, 2018, 5681

- point-to-point:
  - one sender, one receiver
- reliable, in-order byte steam*:*
- pipelined:
  - TCP congestion and flow control set window size

- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) inits sender & receiver state before data exchange
- flow control:
  - sender will not overwhelm receiver
- congestion control:
  - sender will not flood network with traffic (but still try to maximize throughput)



TCP segment structure

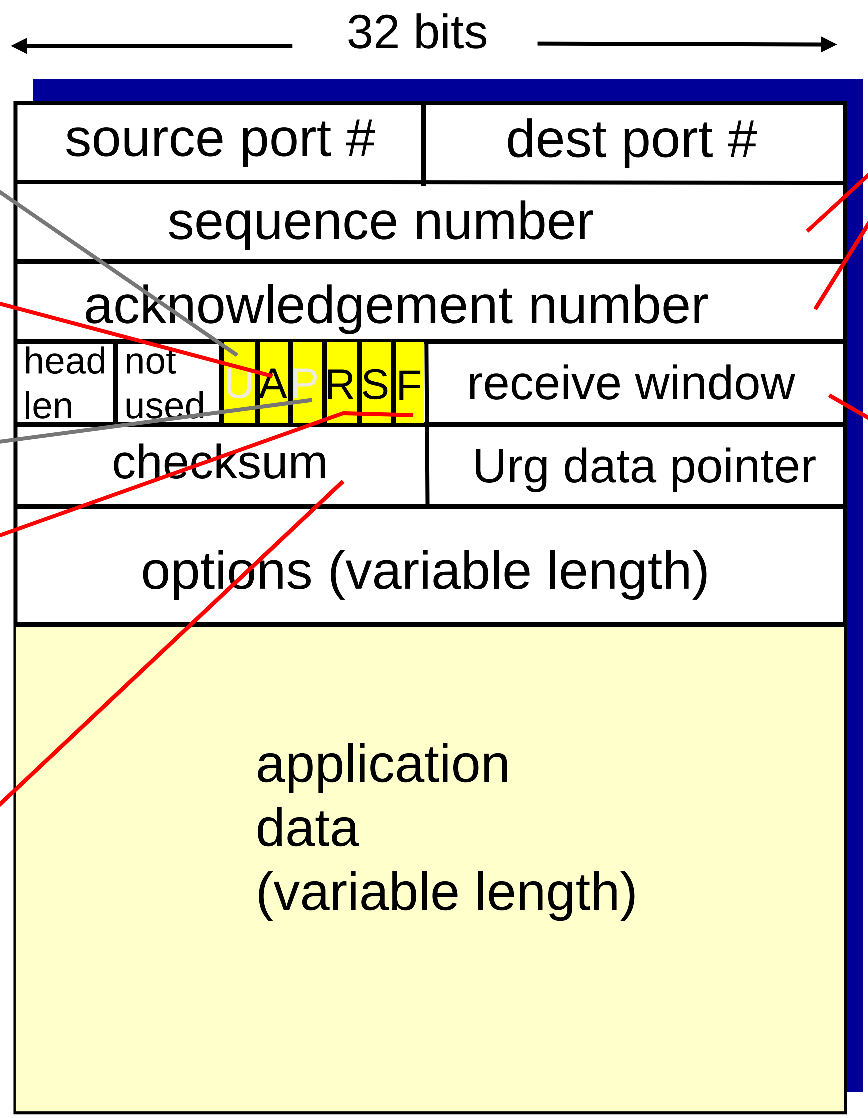
URG: urgent data
(generally not used)

ACK: ACK # valid

PSH: push data now
(generally not used)

RST, SYN, FIN:
connection estab
(setup, teardown
commands)

Internet
checksum
(as in UDP)



counting
by bytes
of data
(not segments!)

bytes
rcvr willing
to accept
(flow control)

Roadmap Transport Layer

- transport layer services
- multiplexing/demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
  - reliable transfer
    - Acknowledgements
    - Retransmissions
    - Connection management
    - Flow control and buffer space
  - Congestion control
    - Principles
    - TCP congestion control

TCP seq. numbers, ACKs

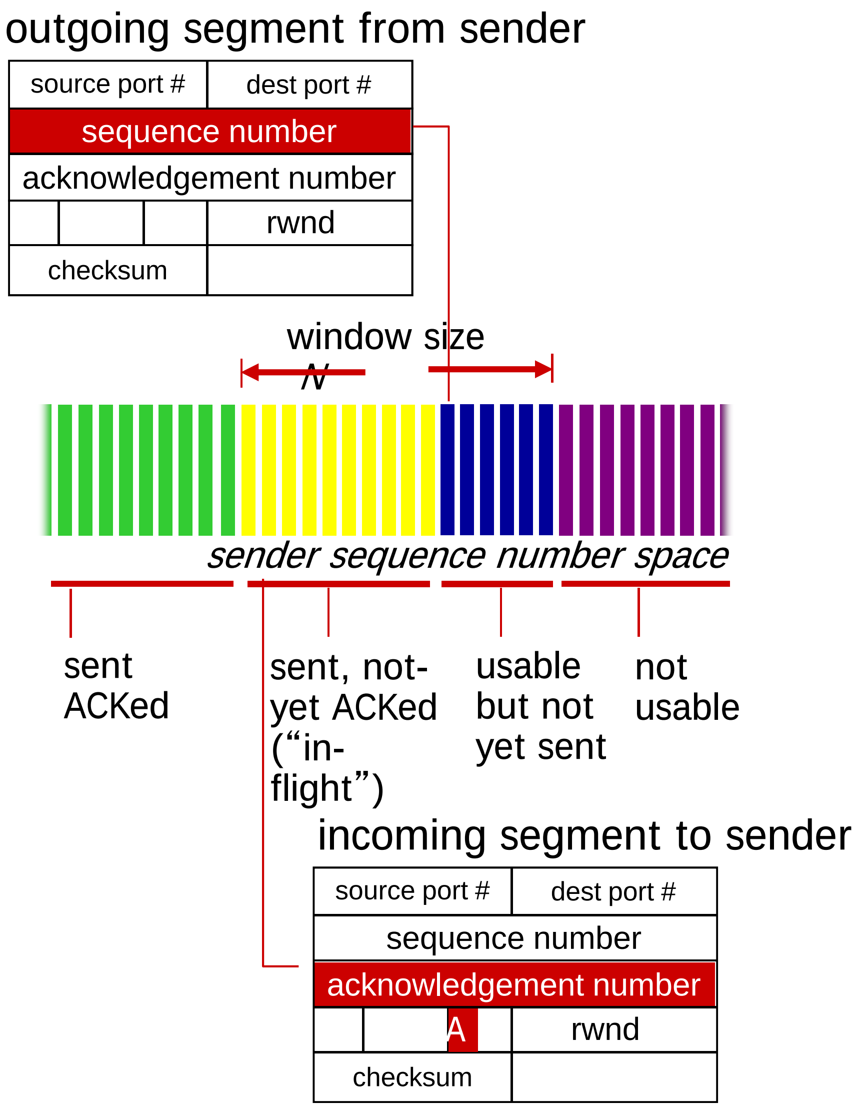
sequence numbers:

– "number" of first byte in segment's data

acknowledgements:

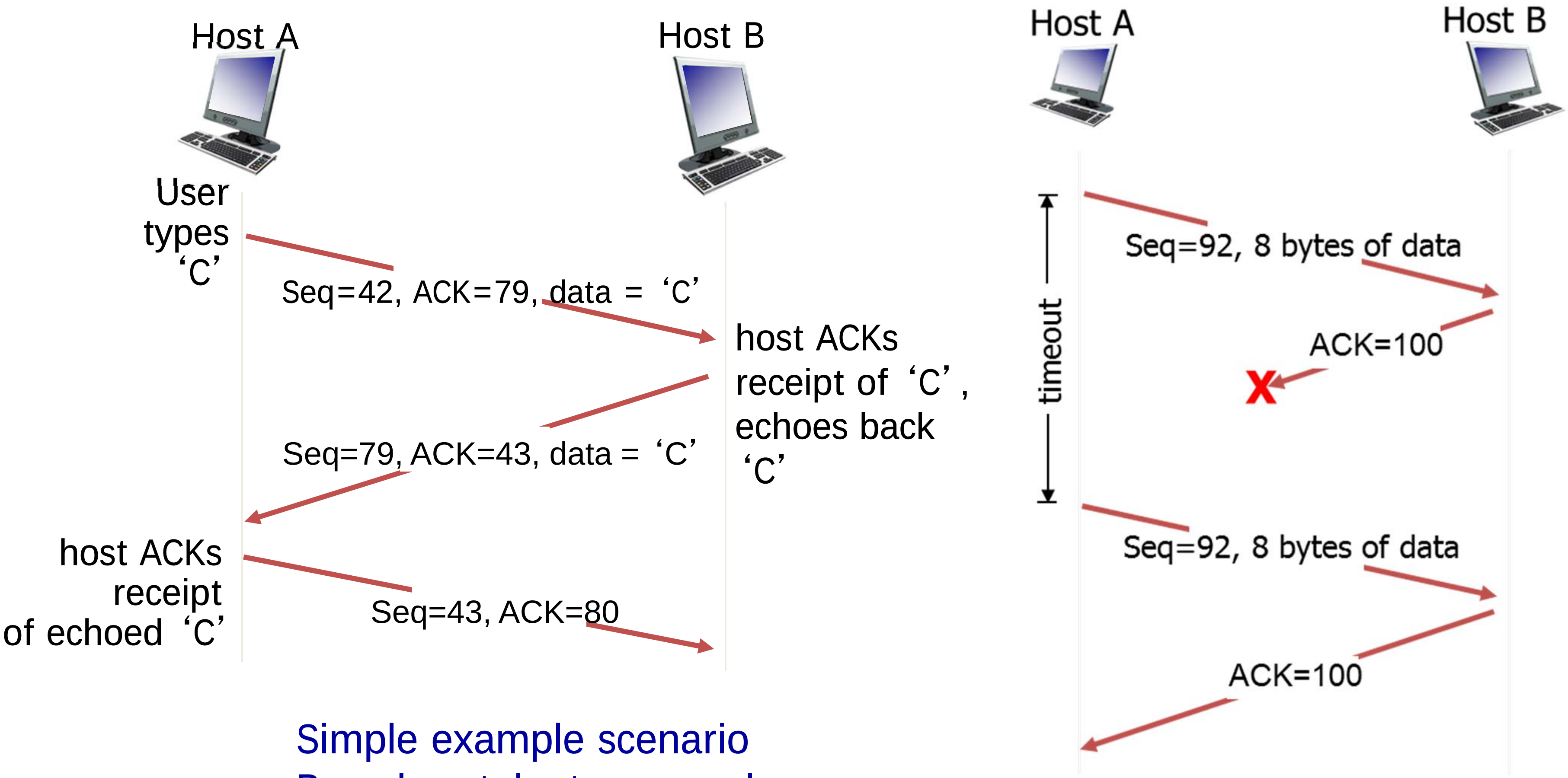
– seq # of next byte expected from other side

– cumulative ACK



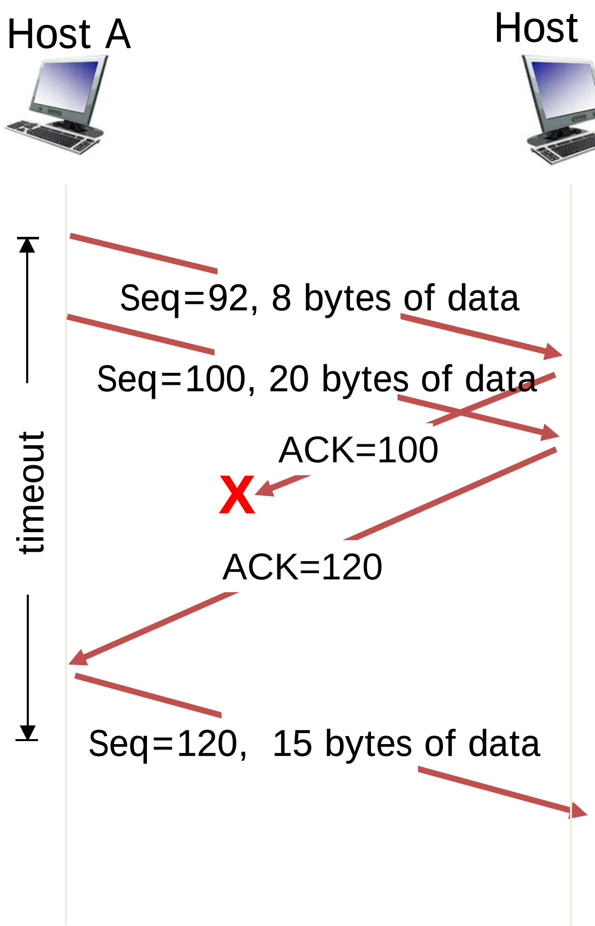
TCP seq. numbers, ACKs

Always ack next in-order expected byte

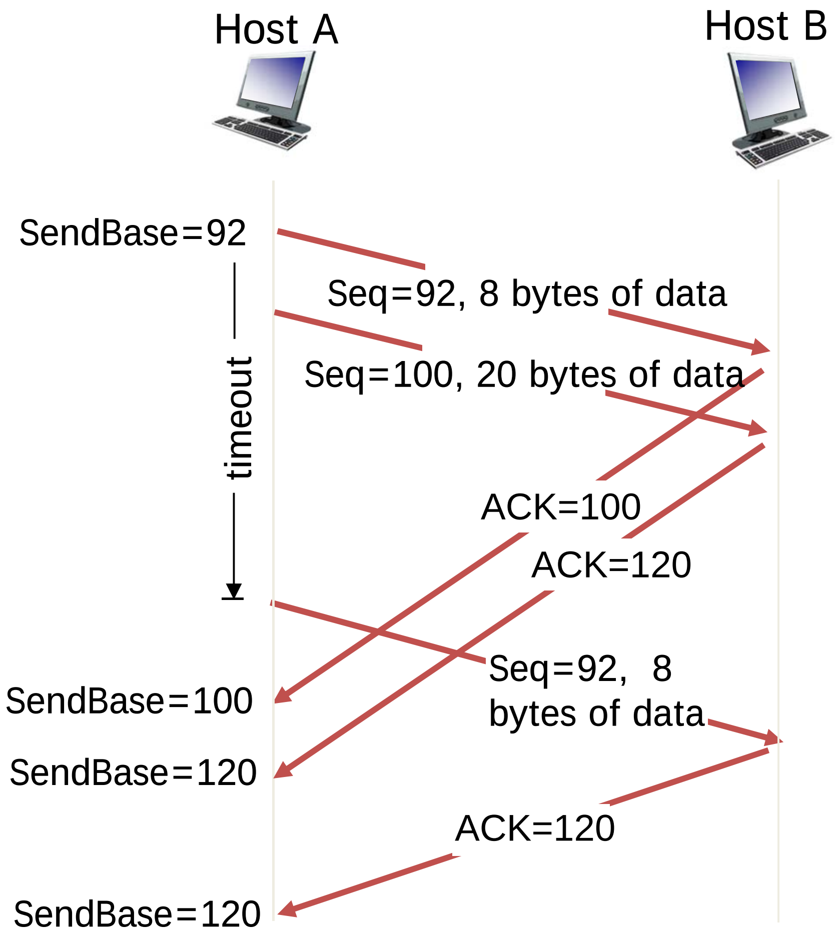


Simple example scenario
Based on telnet msg exchange

TCP:
cumulative Ack - retransmission scenarios



Cumulative ACK



(Premature) timeout

TCP ACK generation [RFC 1122, RFC 5681]

Event	TCP Receiver action
in-order segment arrival, no gaps, everything else already ACKed	**Delayed ACK**. Wait up to 500ms for next segment. If no next segment, send ACK
in-order segment arrival, no gaps, one delayed ACK pending	immediately send single cumulative ACK
out-of-order segment arrival higher-than-expect seq. # gap detected	send (duplicate) ACK, indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	immediate send ACK if segment starts at lower end of gap

From RFC 1122

- TCP SHOULD implement a delayed ACK, but an ACK should not be excessively delayed; in particular, the delay MUST be less than 0.5 seconds, and in a stream of full-sized segments there SHOULD be an ACK for at least every second segment.

- A delayed ACK gives the application an opportunity to update the window and perhaps **to send an immediate response**. In particular, in the case of character-mode remote login, a delayed ACK can reduce the number of segments sent by the server by a factor of 3 (ACK, window update, and echo character all combined in one segment).

- In addition, on some large multi-user hosts, a delayed ACK can substantially reduce protocol processing overhead by reducing the total number of packets to be processed.

- However, excessive delays on ACK's can disturb the round-trip timing and packet "clocking" algorithms.

- We also emphasize that this is a SHOULD, meaning that an implementor should indeed only deviate from this requirement after careful consideration of the implications.

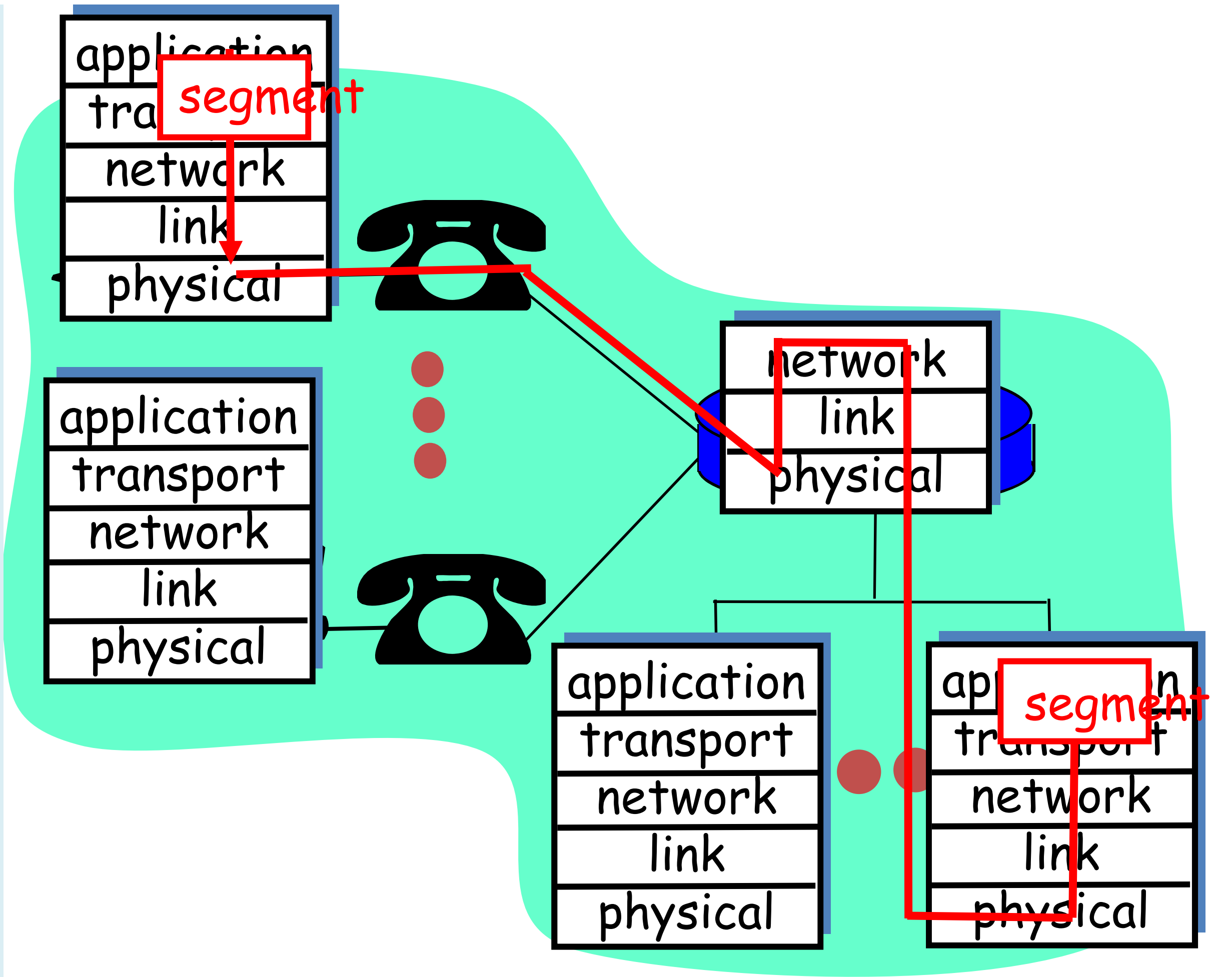
Roadmap Transport Layer

- transport layer services
- multiplexing/demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
  - reliable transfer
    - Acknowledgements
    - Retransmissions
    - Connection management
    - Flow control and buffer space
  - Congestion control
    - Principles
    - TCP congestion control

TCP round trip time, timeout (1)

Q: how to set TCP timeout value?

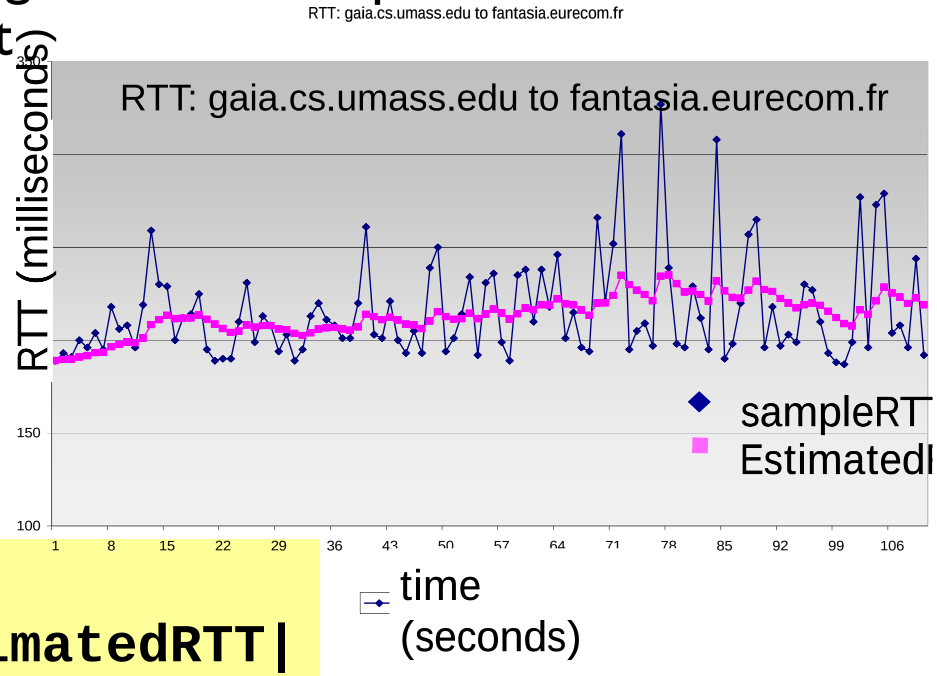
- longer than RTT
  - but RTT varies
- *too short:* premature timeout, unnecessary retransmissions
- *too long:* slow reaction to segment loss



TCP round trip time, timeout (2)

```
EstimatedRTT = (1-α)*EstimatedRTT + α*SampleRTT
```

- exponential weighted moving average: influence of past sample decreases exponentially fast
- typical value: $\alpha = 0.125$




```
DevRTT = (1-β)*DevRTT +
             β*|SampleRTT-EstimatedRTT|
```

(typically, $\beta = 0.25$)

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

estimated RTT   "safety margin"

TCP fast retransmit (RFC 5681)

- time-out period often relatively long:
  - long delay before resending lost packet

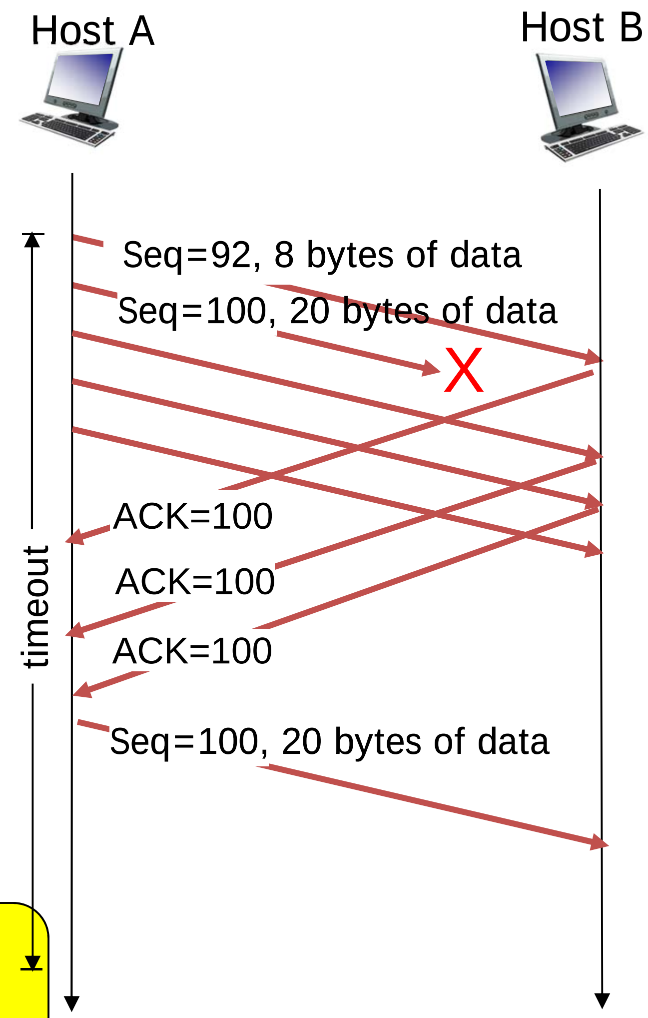
- IMPROVEMENT: detect lost segments via duplicate ACKs

TCP fast retransmit

if sender receives 3 duplicate ACKs for same data

- resend unacked segment with smallest seq #
  - likely that unacked segment lost, so don't wait for timeout

Implicit NAK!
Q: Why need at least 3?



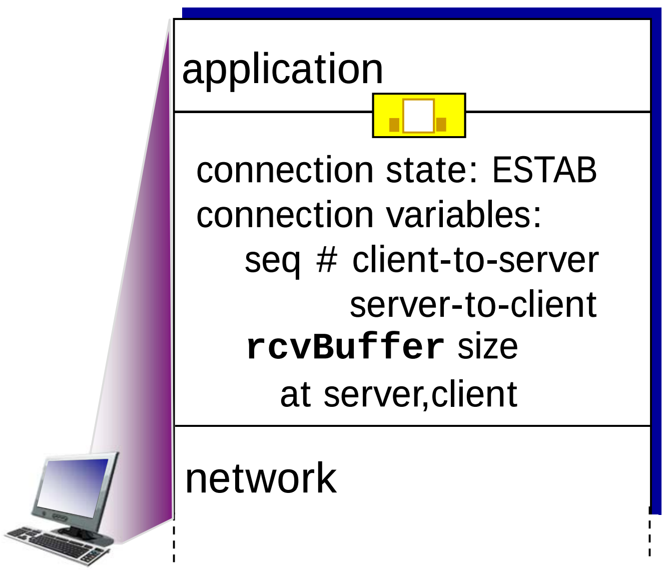
Roadmap Transport Layer

- transport layer services
- multiplexing/demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
  - reliable transfer
    - Acknowledgements
    - Retransmissions
    - Connection management
    - Flow control and buffer space
  - Congestion control
    - Principles
    - TCP congestion control

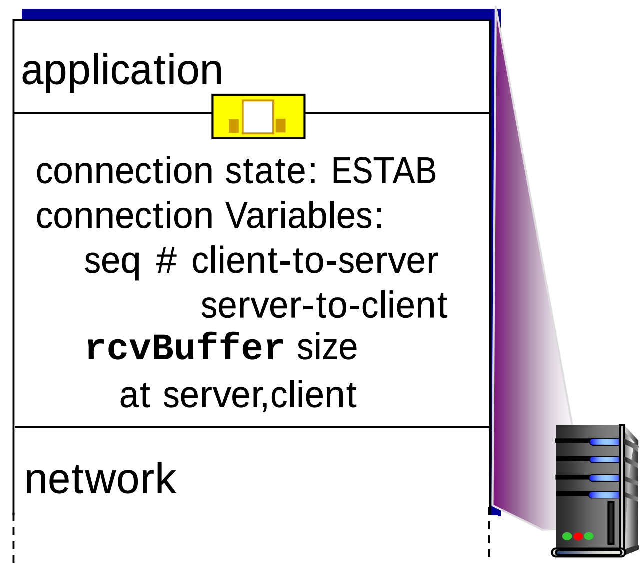
Connection Management

before exchanging data, sender/receiver "handshake":

- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters

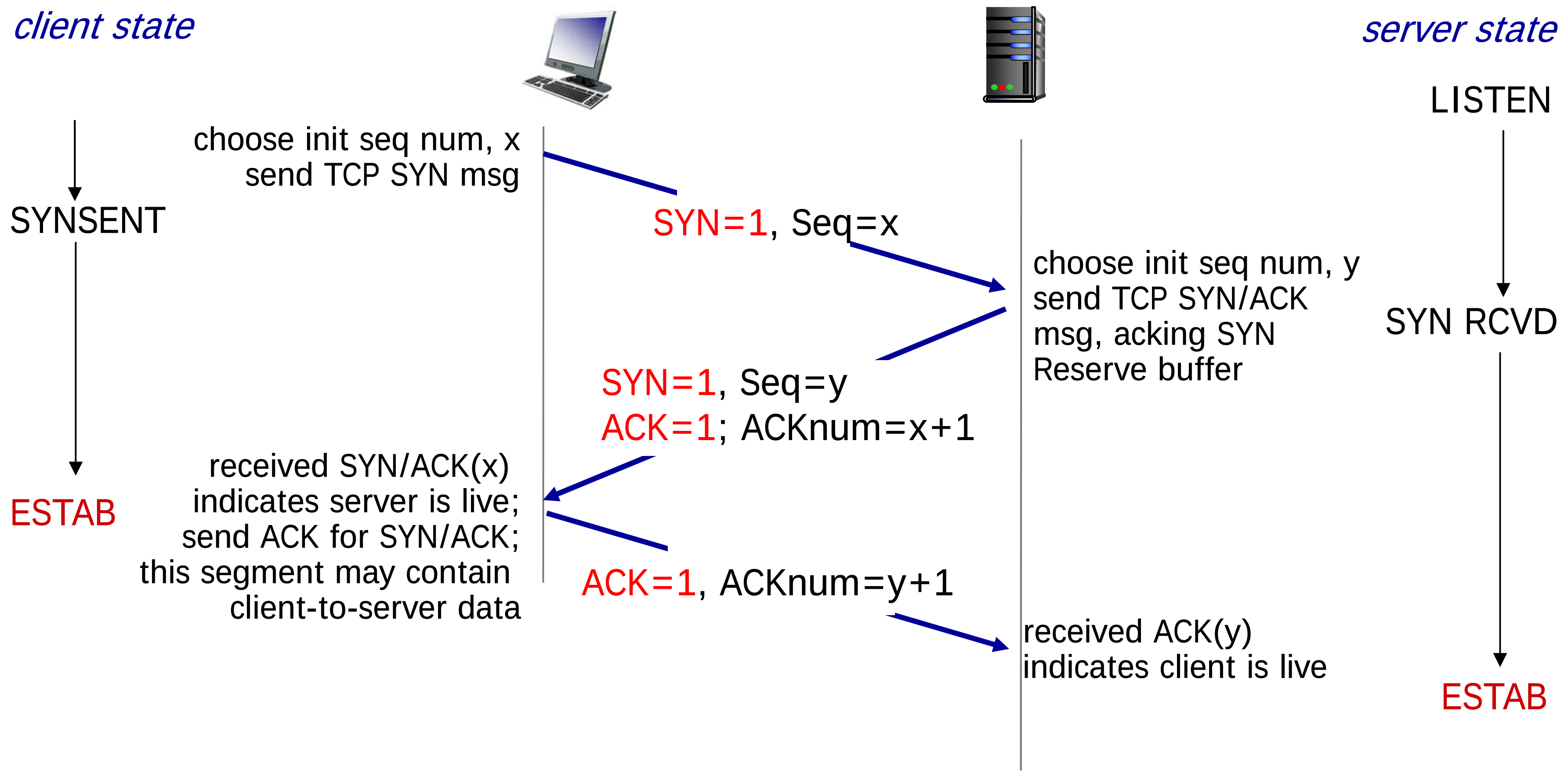



```
Socket clientSocket =
  newSocket("hostname","port
  number");
```

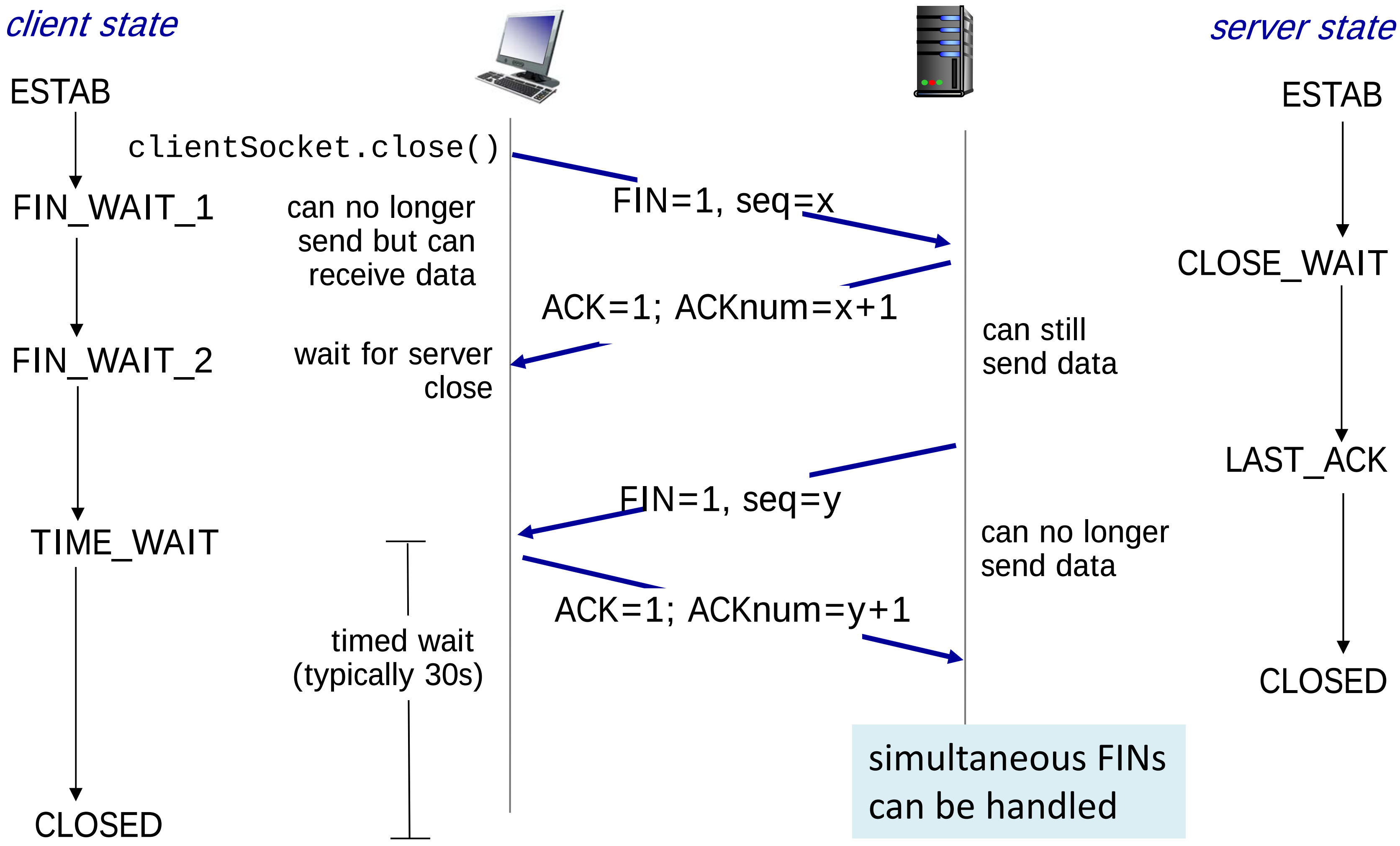



```
Socket connectionSocket =
  welcomeSocket.accept();
```

Setting up a connection: TCP 3-way handshake



TCP: closing a connection



simultaneous FINs can be handled

RST: alternative way to close connection immediately, when **error** occurs

TCP – Closing a connection: Reset

RST

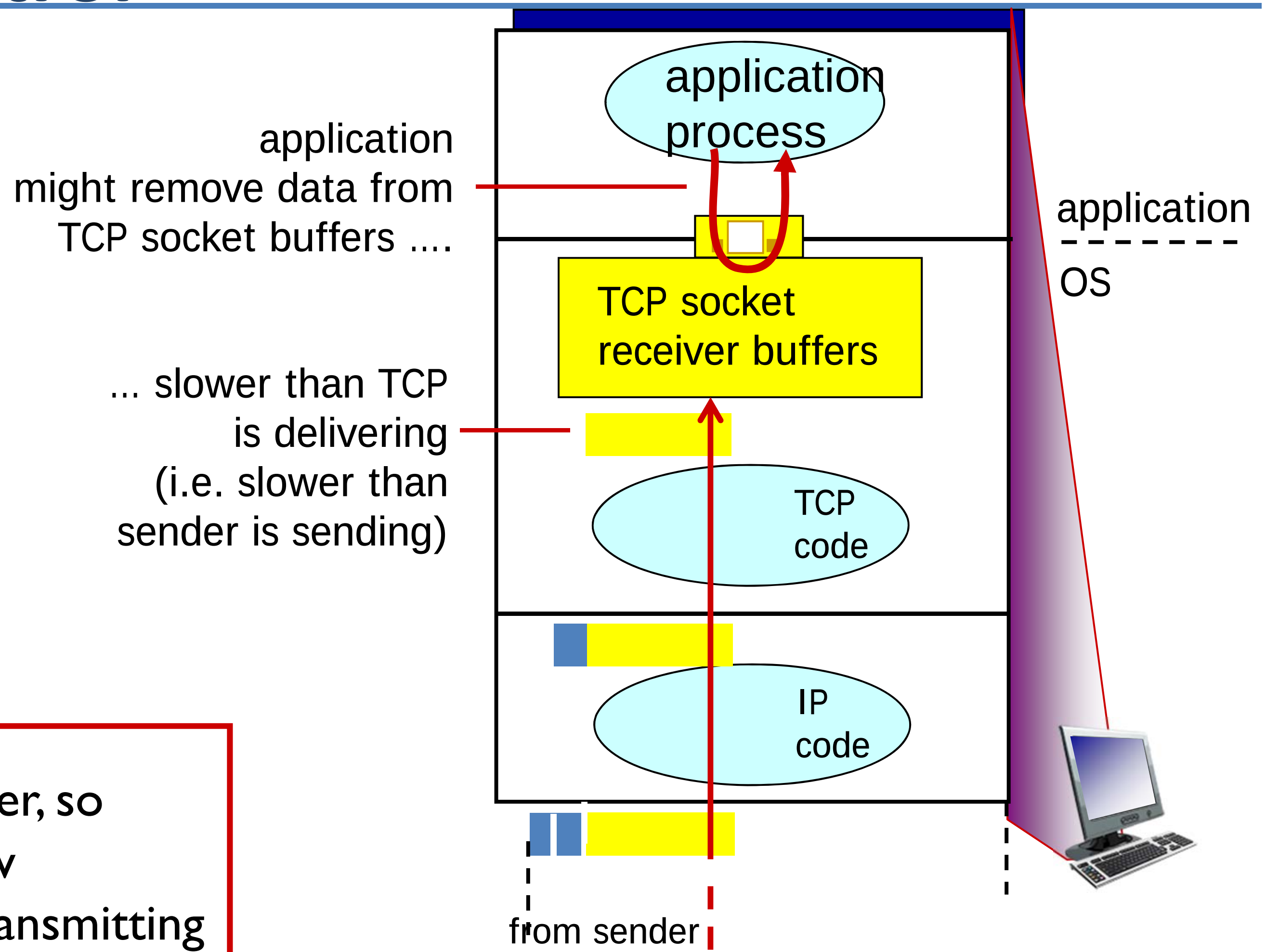
- RST is used to signal an error condition and causes an immediate close of the connection on both sides
- RST packets are not supposed to carry data payload, except for an optional human-readable description of what was the reason for dropping this connection.
- Examples:
  - A TCP data segment when no session exists
  - Arrival of a segment with incorrect sequence number
  - Connection attempt to non-existing port
  - Etc.

Is TCP stateful or stateless?

Roadmap Transport Layer

- transport layer services
- multiplexing/demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
  - reliable transfer
    - Acknowledgements
    - Retransmissions
    - Connection management
    - Flow control and buffer space
  - Congestion control
    - Principles
    - TCP congestion control

TCP flow control



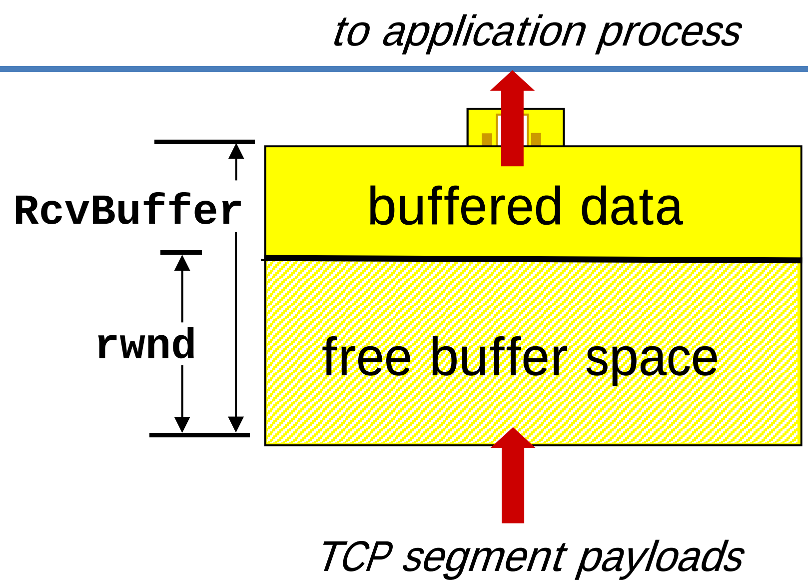
receiver protocol stack

flow control

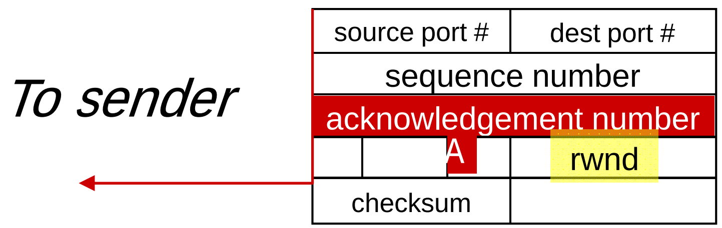
receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

TCP flow control

- receiver "advertises" free buffer space by including **`rwnd`** value in TCP header of receiver-to-sender segments
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **`RcvBuffer`**
- sender limits amount of unacked ("in-flight") data to receiver's **`rwnd`** value
- guarantees receive buffer will not overflow



receiver-side buffering



Roadmap Transport Layer

- transport layer services
- multiplexing/demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
  - reliable transfer
    - Acknowledgements
    - Retransmissions
    - Connection management
    - Flow control and buffer space
  - Congestion control
    - Principles
    - TCP congestion control

Principles of congestion control

congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)

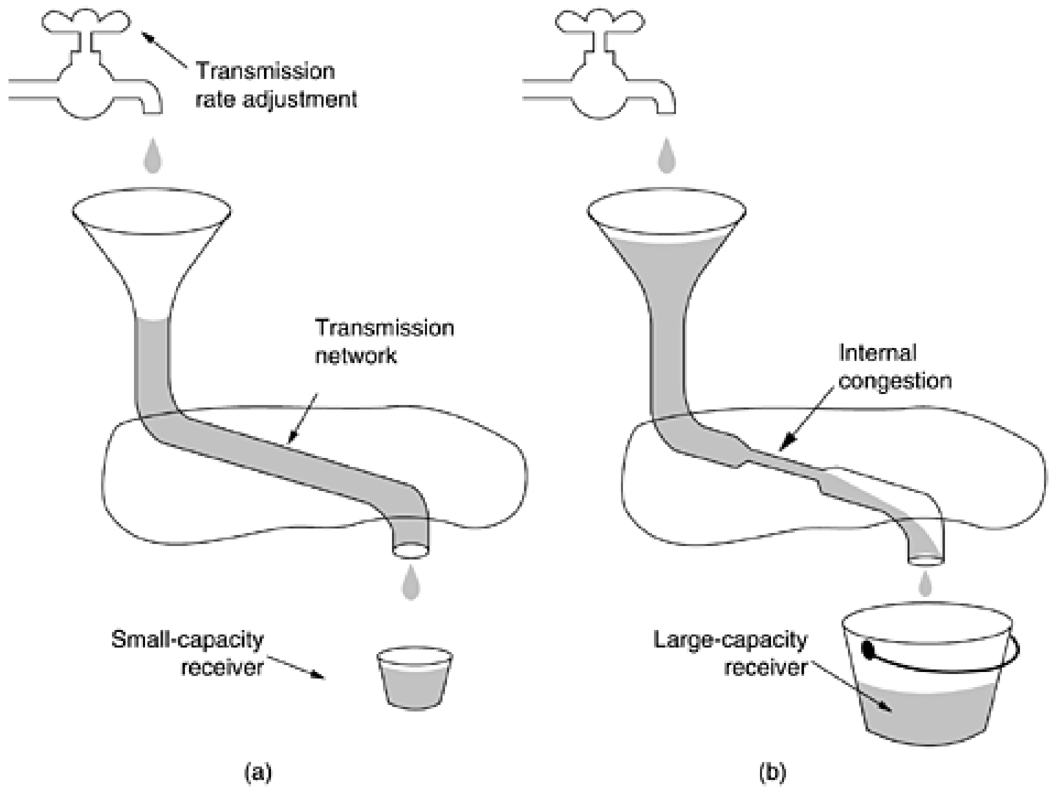


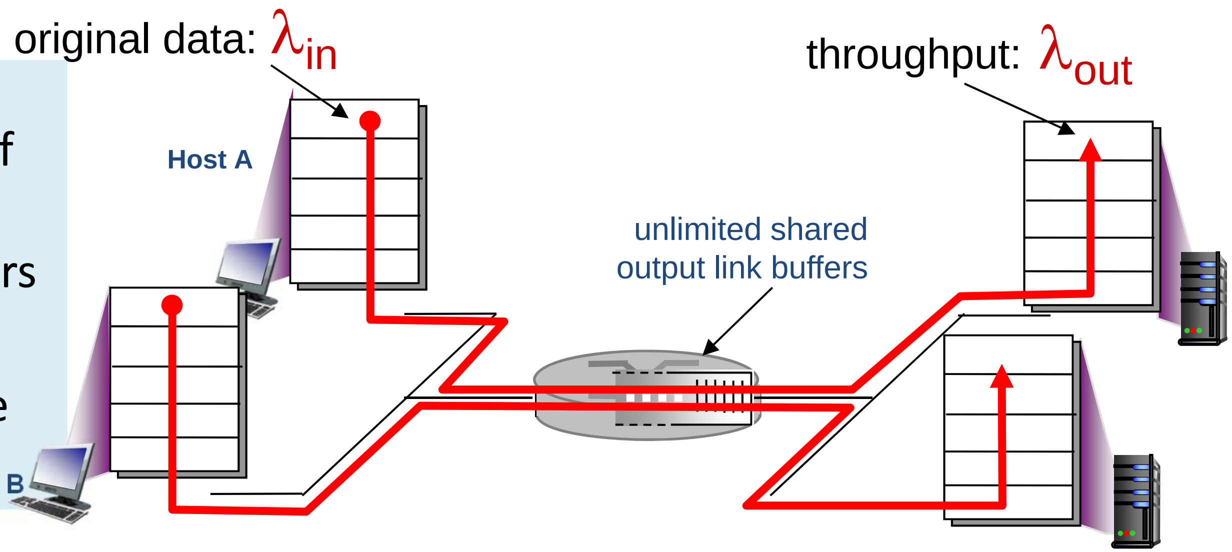
Fig. A. Tanenbaum
Computer Networks

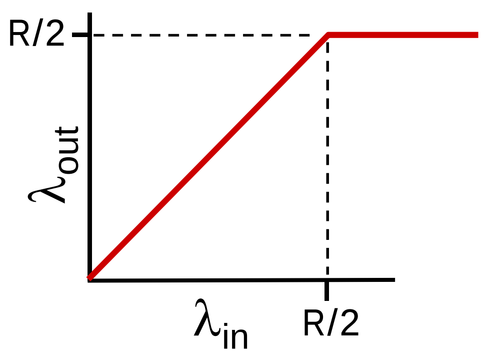
Need for flow control

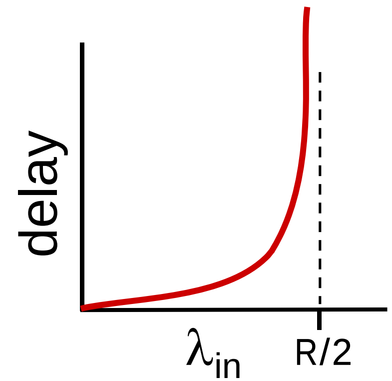
Need for congestion control

Causes/costs of congestion: scenario 1 (unrealistic)

- two senders, two receivers, <u>average</u> rate of data is λ_{in}
- one router, infinite buffers
- output link capacity: R
- (no retransmission in the "picture" yet)





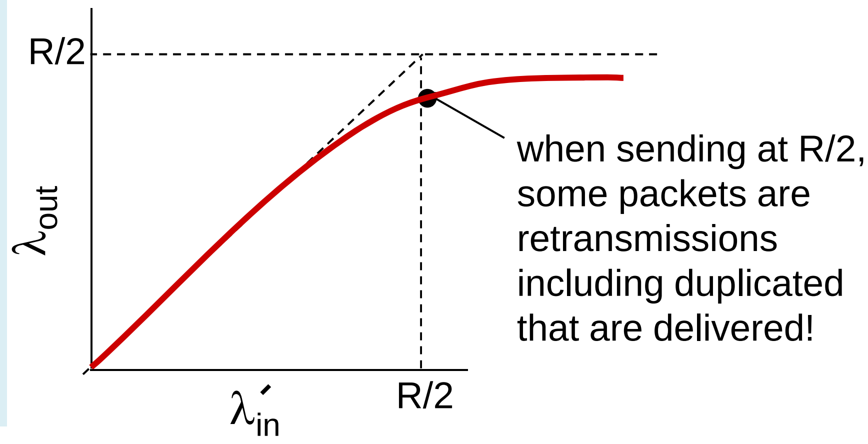


- maximum per-connection throughput: R/2
- large delays as arrival rate, λ_{in}, approaches capacity

Causes/costs of congestion: scenario 2

Realistic buffers bounded =>: duplicates

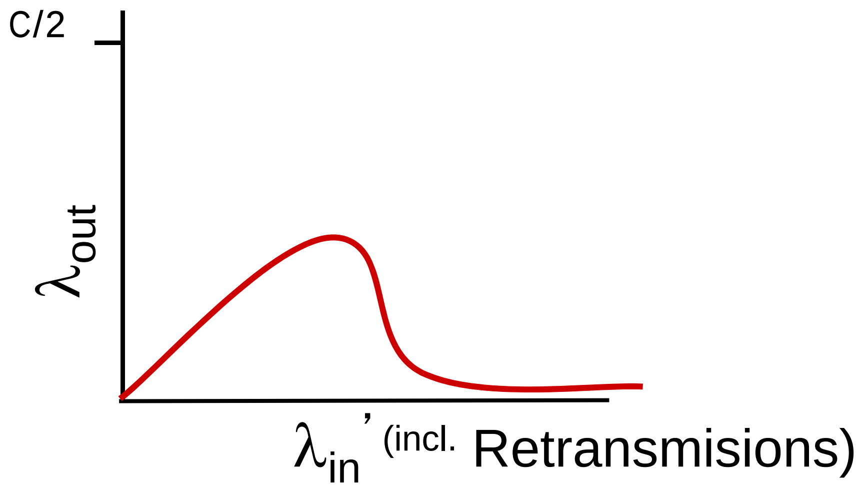
- packets can be lost, dropped at router due to full buffers
- sender times out, sending *two* copies



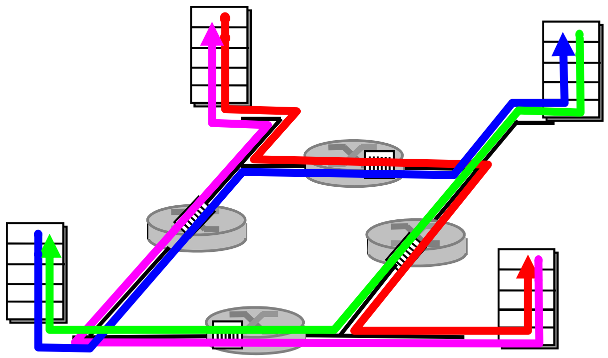
"costs" of congestion:

- more work (retrans) for given "goodput" (application-level throughput)
- unneeded retransmissions: links carry multiple copies of pkt

Causes/costs of congestion: scenario 3



Consider 4 streams



another cost of congestion:

- when packets 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 eg.
  - a single bit indicating congestion
  - explicit rate for sender to send at

Roadmap Transport Layer

- transport layer services
- multiplexing/demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
  - reliable transfer
    - Acknowledgements
    - Retransmissions
    - Connection management
    - Flow control and buffer space
  - Congestion control
    - Principles
    - TCP congestion control

TCP congestion control:
additive increase multiplicative decrease

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

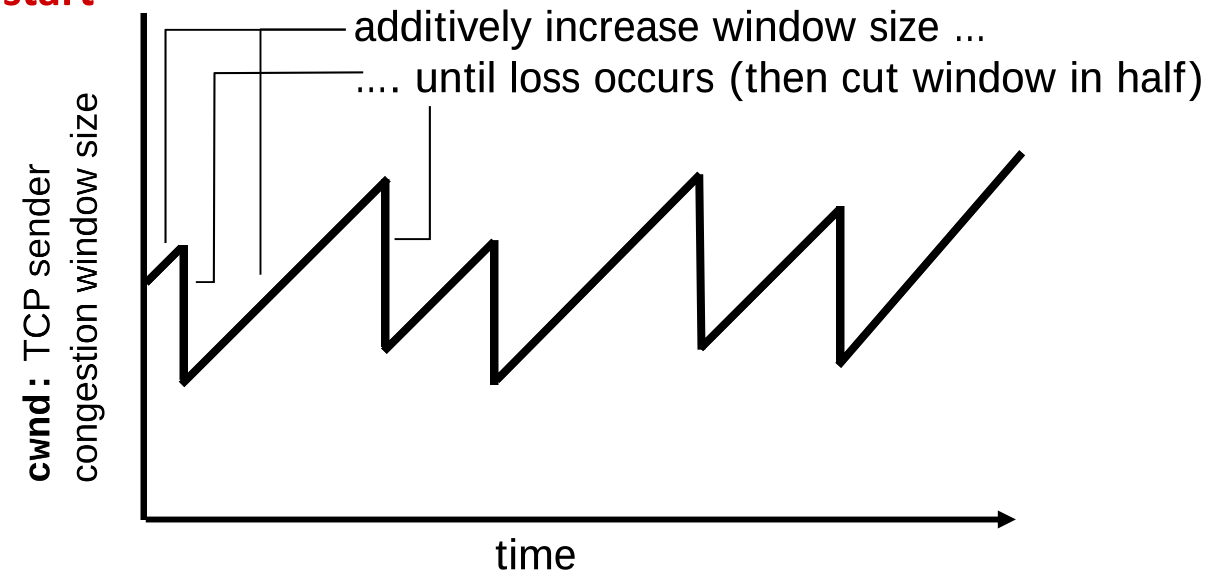
How does sender perceive congestion?

- loss = timeout *or* 3 duplicate acks
- TCP sender reduces rate (**Congestion Window**) then

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

- *Additive Increase:* increase **cwnd** by 1 MSS every RTT until loss detected
- *Multiplicative Decrease*: cut **cwnd** in half after loss
- **To start with: slow start**

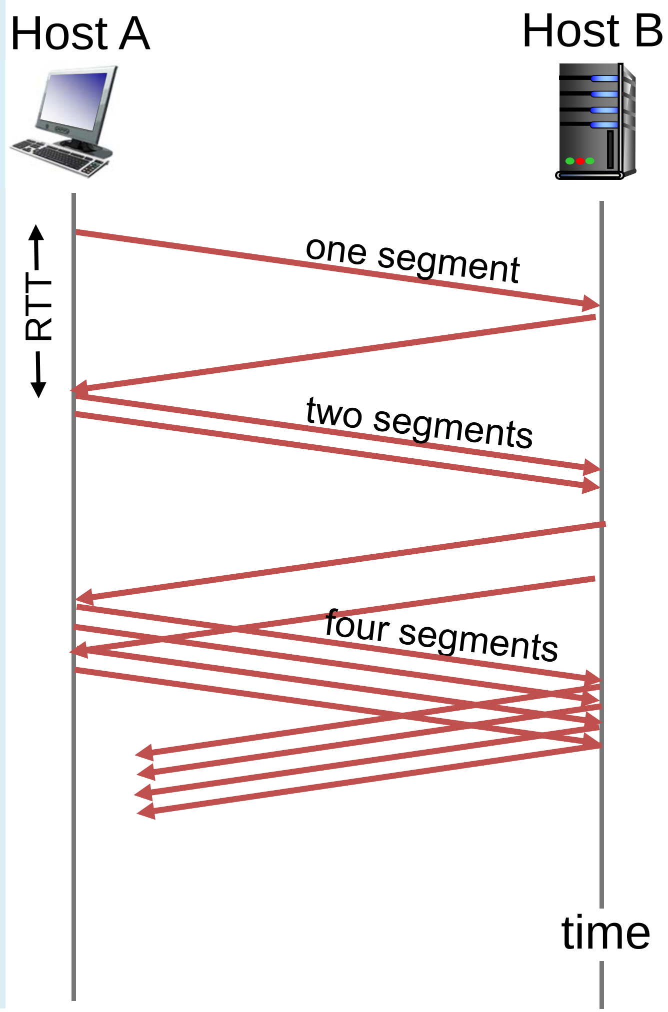
AIMD saw tooth behavior: probing for bandwidth



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

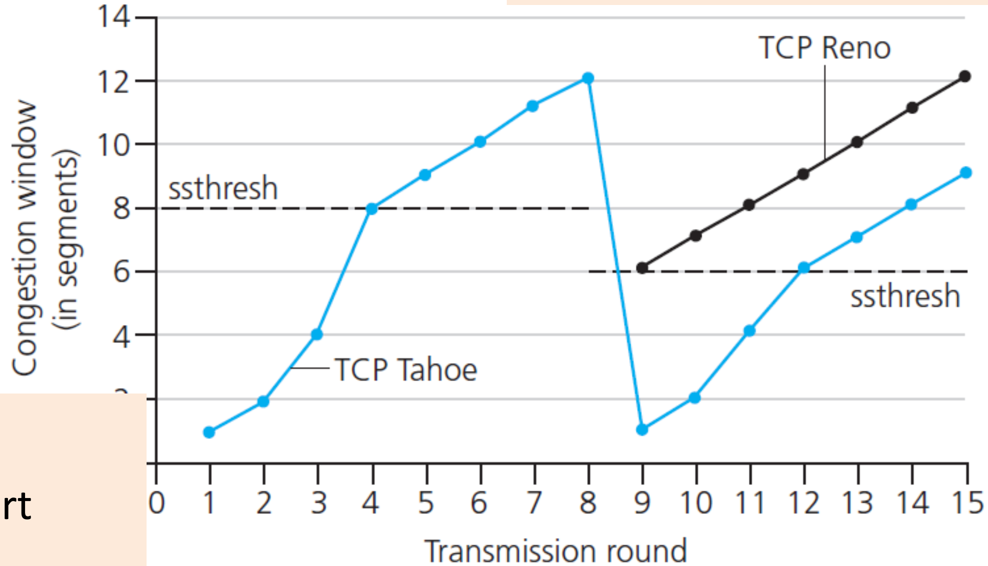


TCP cwnd:
from exp. to linear growth + reacting to loss

Q: when should the exponential increase switch to linear?

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

Reno: loss indicated by timeout or 3 duplicate ACKs: cwnd is cut in half; then grows linearly

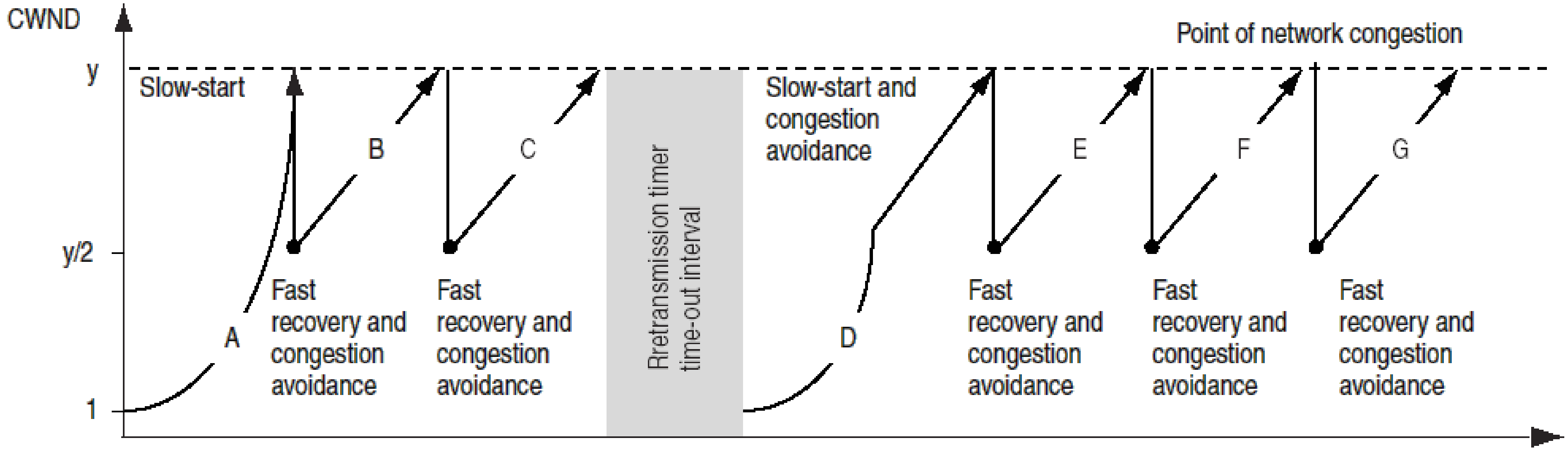


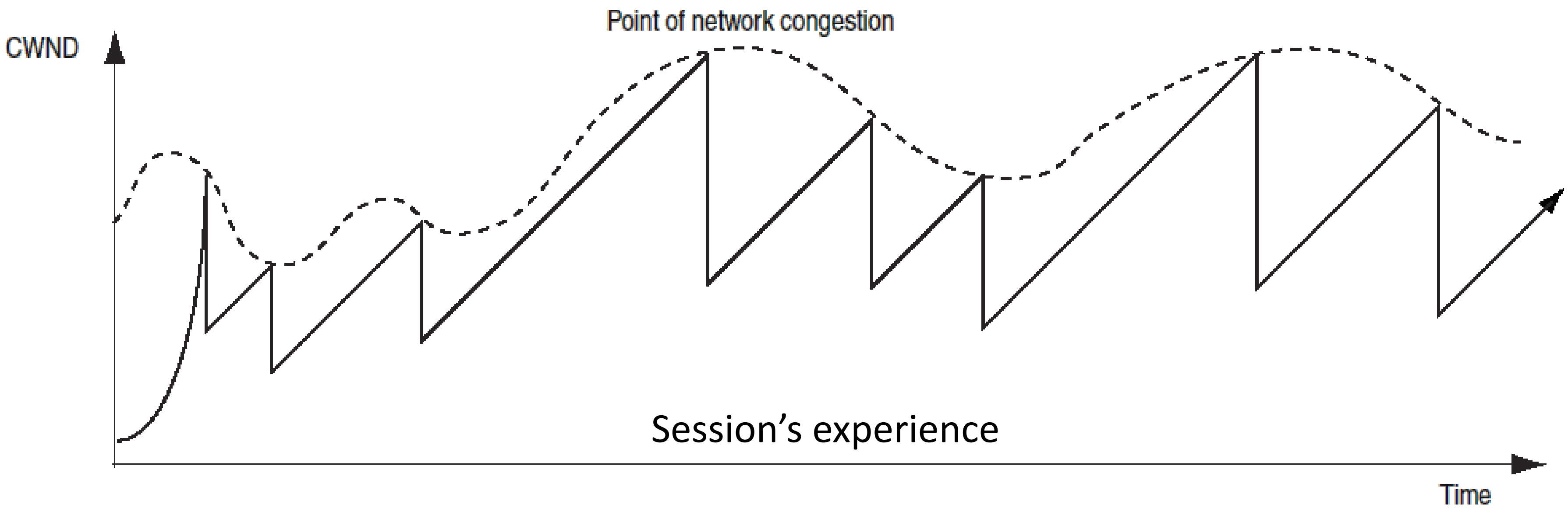
Implementation:

- variable **ssthresh** (slow start threshold)
- on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event

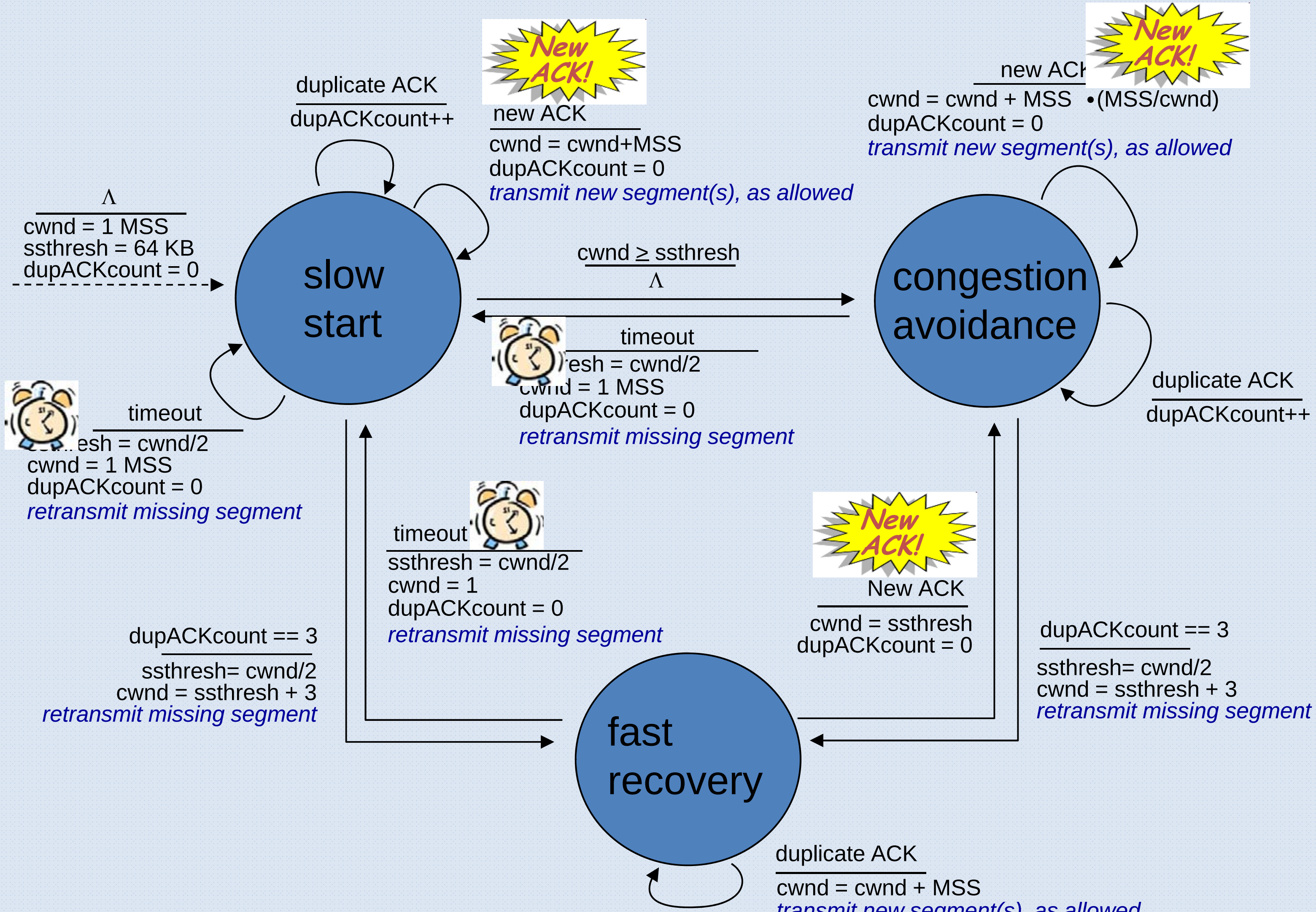
Non-optimized: loss indicated by timeout: cwnd set to 1 MSS; window then grows as in slow start, to threshold, then grows linearly

Fast recovery (Reno)



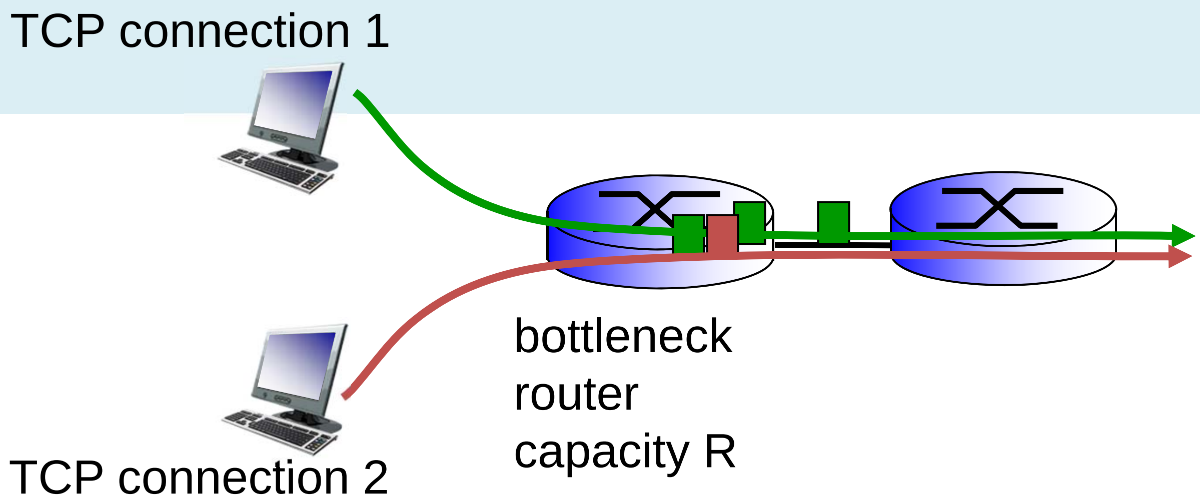


Summary: TCP Congestion Control



TCP Fairness

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



How many windows does a TCP's sender maintain?

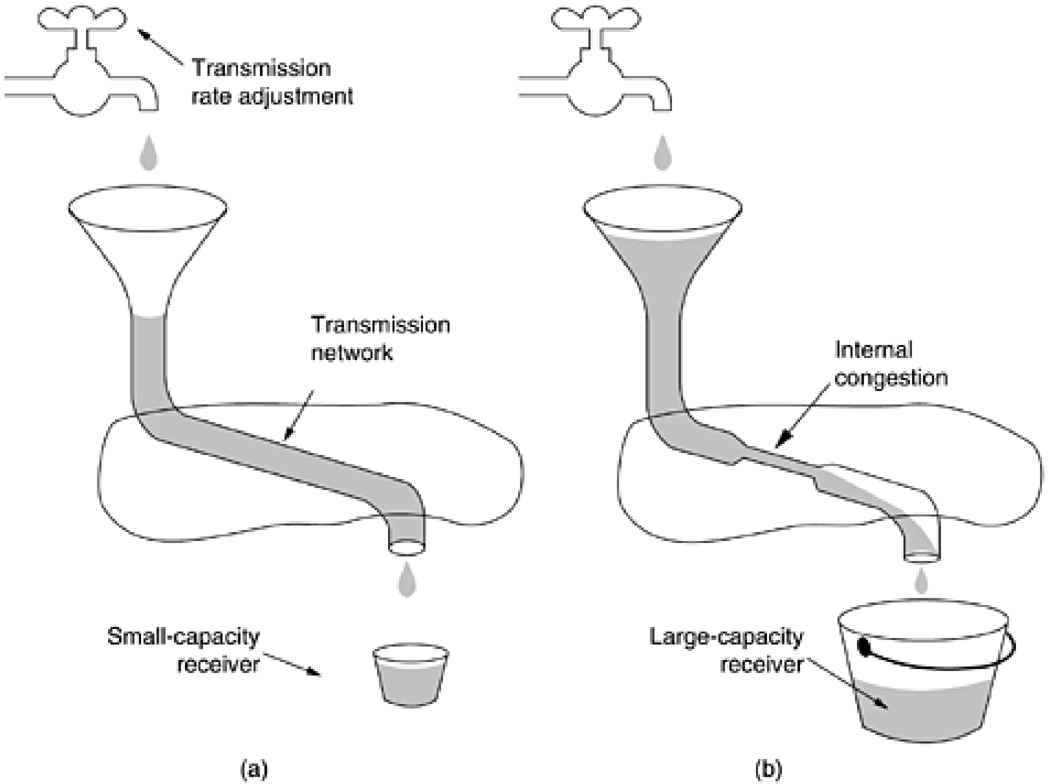
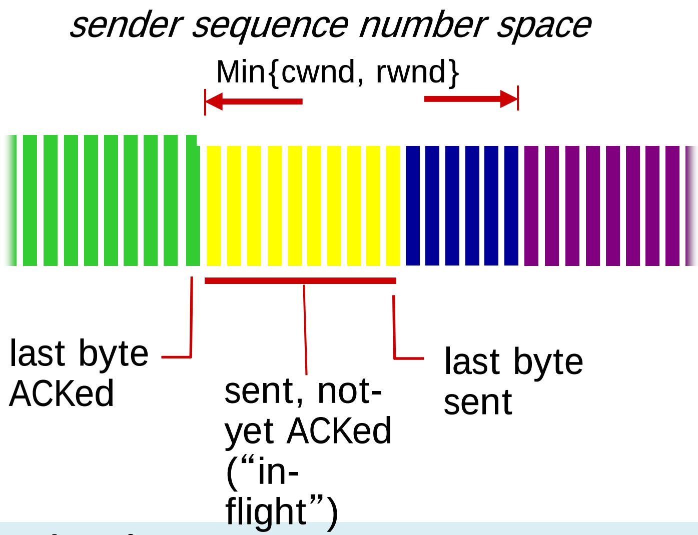


Fig. A. Tanenbaum
Computer Networks

Need for flow control

Need for congestion control

TCP combined flow-ctrl, congestion ctrl windows



TCP sending rate:

- *roughly:* send min {cwnd, rwnd} bytes, wait RTT for ACKS, then send more bytes

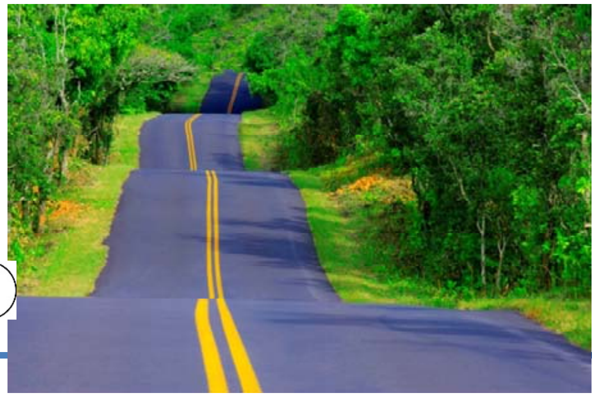
sender limits transmission:

```
LastByteSent-
  LastByteAcked  ≤  Min{cwnd, rwnd}
```

- **cwnd** is dynamic, function of perceived network congestion,
- **rwnd** dymanically limited by receiver's buffer space

Roadmap Transport Layer

- transport layer services
- multiplexing/demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
  - reliable transfer
    - Acknowledgements
    - Retransmissions
    - Connection management
    - Flow control and buffer space
  - Congestion control
    - Principles
    - TCP congestion control



Chapter 3: summary

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

<u>next:</u>

- leaving the network "edge" (application, transport layers)
- into the network "core"

Some review questions on this part

- Describe TCP’s flow control
- Why does TCp do fast retransmit upon a 3rd ack and not a 2nd?
- Describe TCP’s congestion control: principle, method for detection of congestion, reaction.
- Can a TCP’s session sending rate increase indefinitely?
- Why does TCP need connection management?
- Why does TCP use handshaking in the start and the end of connection?
- Can an application have reliable data transfer if it uses UDP? How or why not?

Reading instructions chapter 3

- **KuroseRoss book**

Careful	Quick
3.1, 3.2, 3.4-3.7	3.3

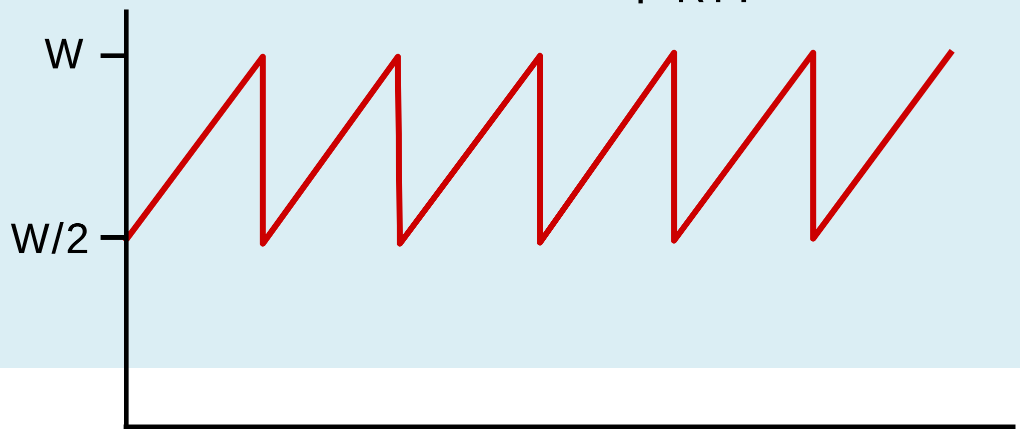
- **Other resources (further study)**
  - Eddie Kohler, Mark Handley, and Sally Floyd. 2006. Designing DCCP: congestion control without reliability. *SIGCOMM Comput. Commun. Rev.* 36, 4 (August 2006), 27-38. DOI=10.1145/1151659.1159918 http://doi.acm.org/10.1145/1151659.1159918
  - http://research.microsoft.com/apps/video/default.aspx?id=104005
  - Exercise/throughput analysis TCP in following slides

Extra slides, for further study

TCP throughput

- avg. TCP throughput as function of window size, RTT?
  - ignore slow start, assume always data to send
- W: window size (measured in bytes) where loss occurs
  - avg. window size (# in-flight bytes) is ¾ W
  - avg. trhoughput is 3/4W per RTT

$$\text{avg TCP trhoughput} = \frac{3}{4}\frac{W}{RTT} \text{ bytes/sec}$$



TCP Futures: TCP over "long, fat pipes"

- example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- requires W = 83,333 in-flight segments
- throughput in terms of segment loss probability, L [Mathis 1997]:

$$\text{TCP throughput} = \frac{1.22 \cdot MSS}{RTT \sqrt{L}}$$

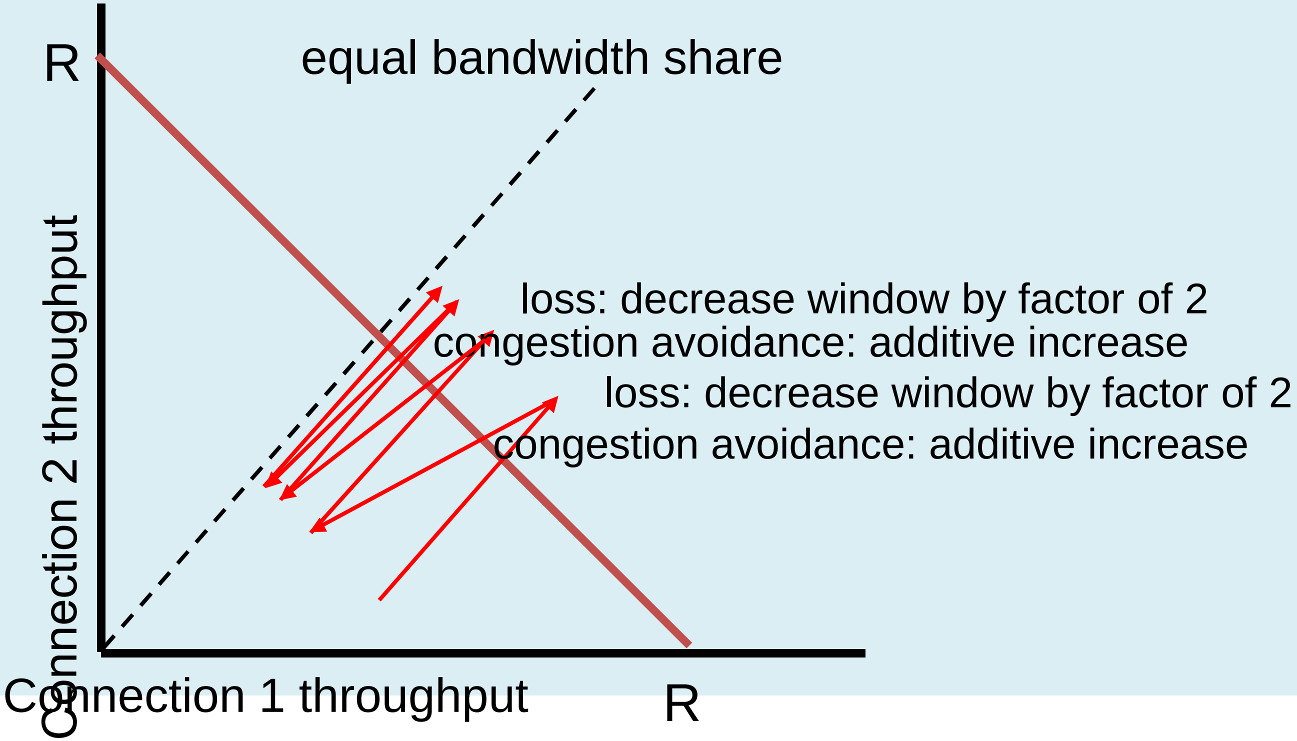
➜ to achieve 10 Gbps throughput, need a loss rate of L = $2 \cdot 10^{-10}$ *– a very small loss rate!*

- new versions of TCP for high-speed

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:
  - send audio/video at constant rate, tolerate packet loss

Fairness, parallel TCP connections

- application can open multiple parallel connections between two hosts
- web browsers do this
- e.g., link of rate R with 9 existing connections:
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2

TCP delay modeling (slow start – related)

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

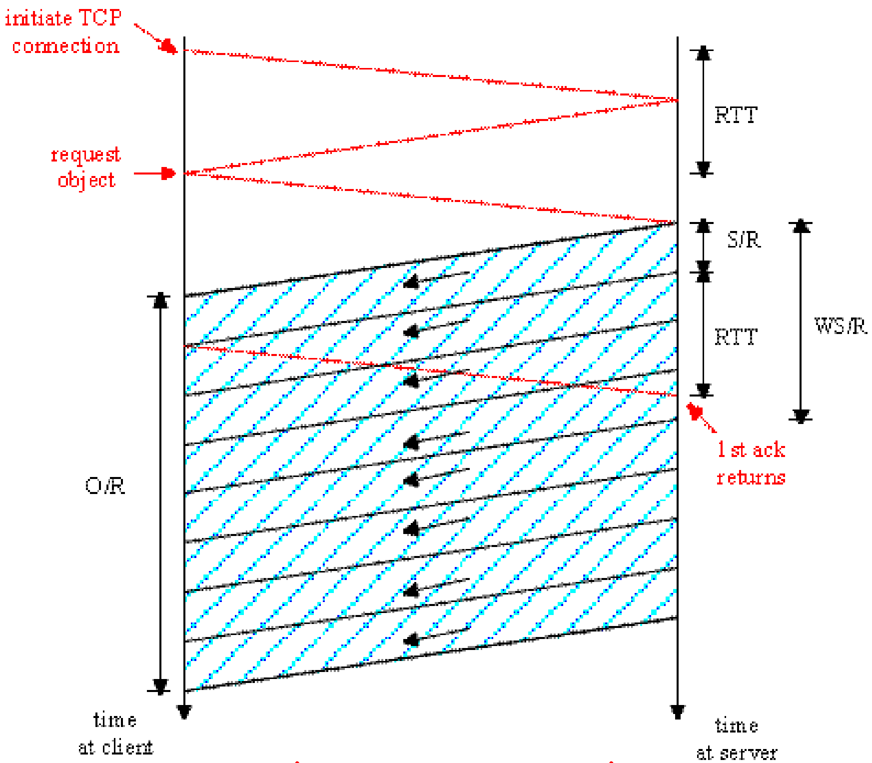
- TCP connection establishment
- data transfer delay

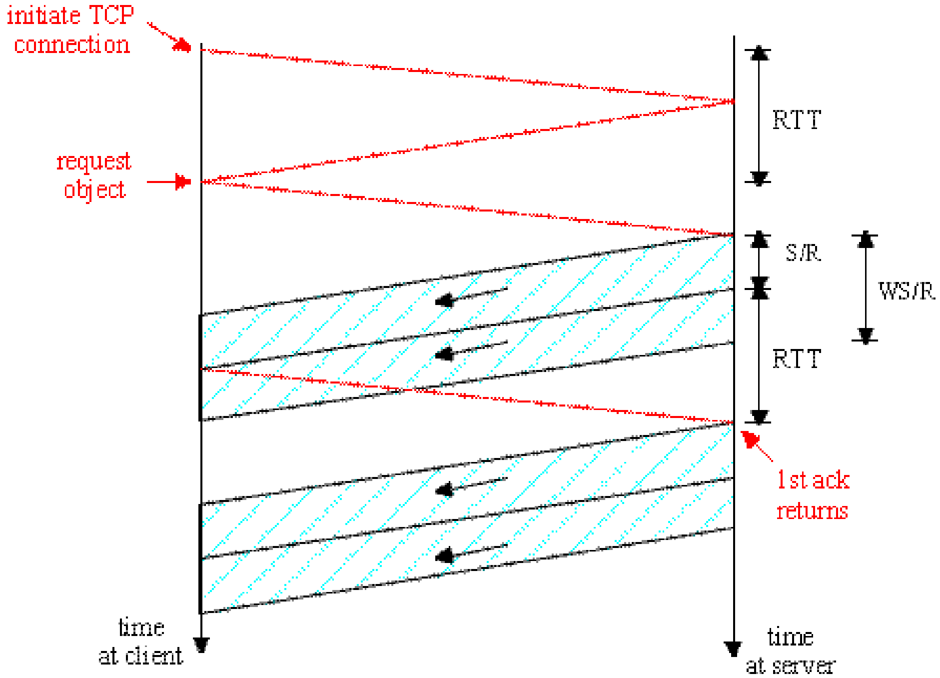
Notation, assumptions:

- Assume one link between client and server of rate R
- Assume: fixed congestion window, W segments
- S: MSS (bits)
- O: object size (bits)
- no retransmissions (no loss, no corruption)
- Receiver has unbounded buffer

TCP delay Modeling: simplified, fixed window

K:= O/WS





Case 1: WS/R > RTT + S/R:
ACK for first segment in window returns before window's worth of data nsent
delay = 2RTT + O/R

Case 2: 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]

$$\text{delay} = \frac{O}{R} + 2RTT + \sum^{P} idleTime_p$$

TCP Delay Modeling: Slow Start

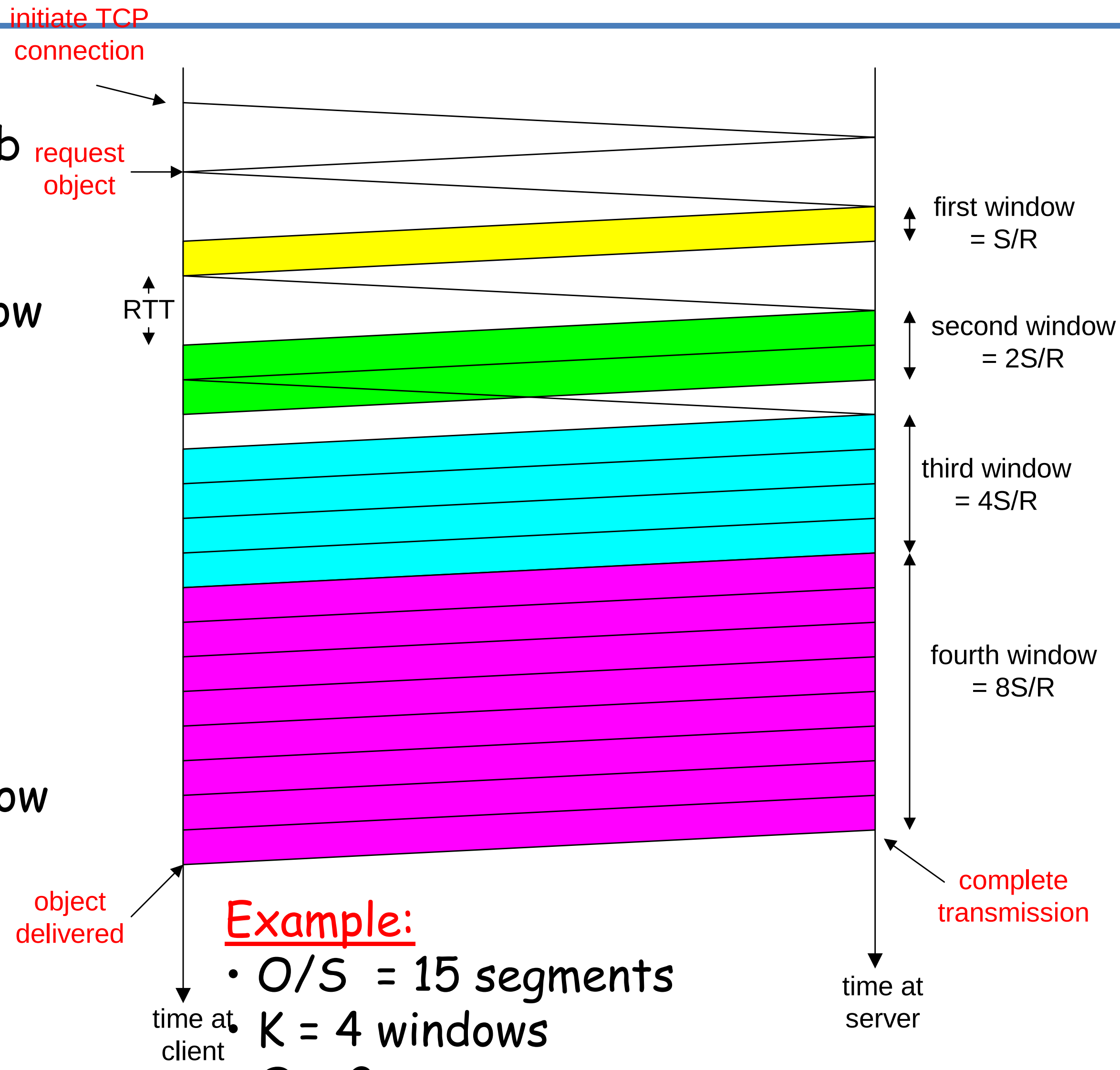
Delay components:

- 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

where
- Q = #times server stalls until cong. window is larger than a "full-utilization" window (if the object were of unbounded size).

- K = #(incremental-sized) congestion-windows that "cover" the object.



Example:

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

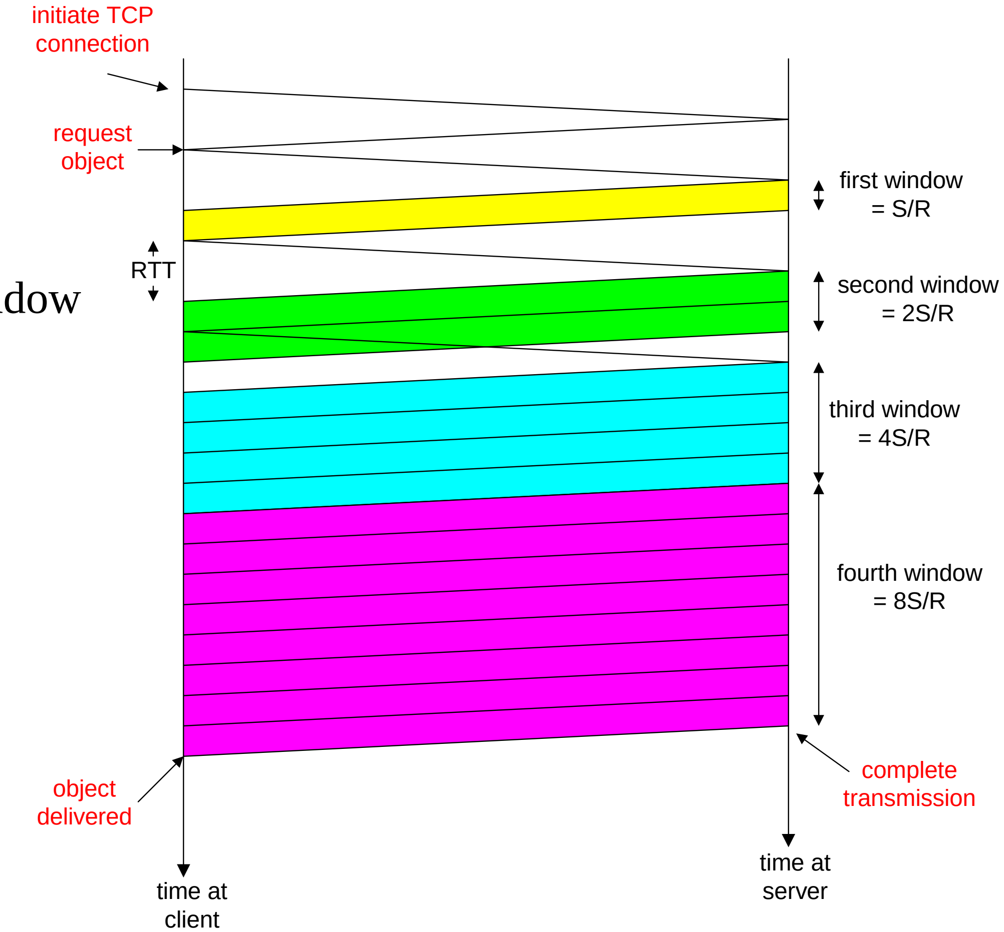
TCP Delay Modeling (slow start - cont)

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

until server receives acknowledgement

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

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



$$\text{delay} = \frac{O}{R} + 2RTT + \sum_{p=1}^{P} idleTime_p$$

$$= \frac{O}{R} + 2RTT + \sum_{k=1}^{P}\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right]$$

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

TCP Delay Modeling

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.