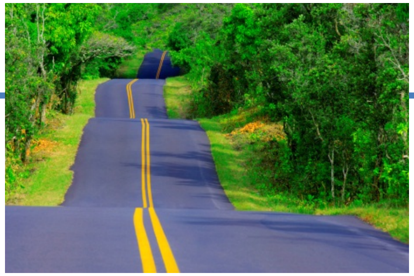
Course on Computer Communication and Networks

Lecture 5
Chapter 3; Transport Layer, Part B

EDA344/DIT 423, CTH/GU

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

Roadmap



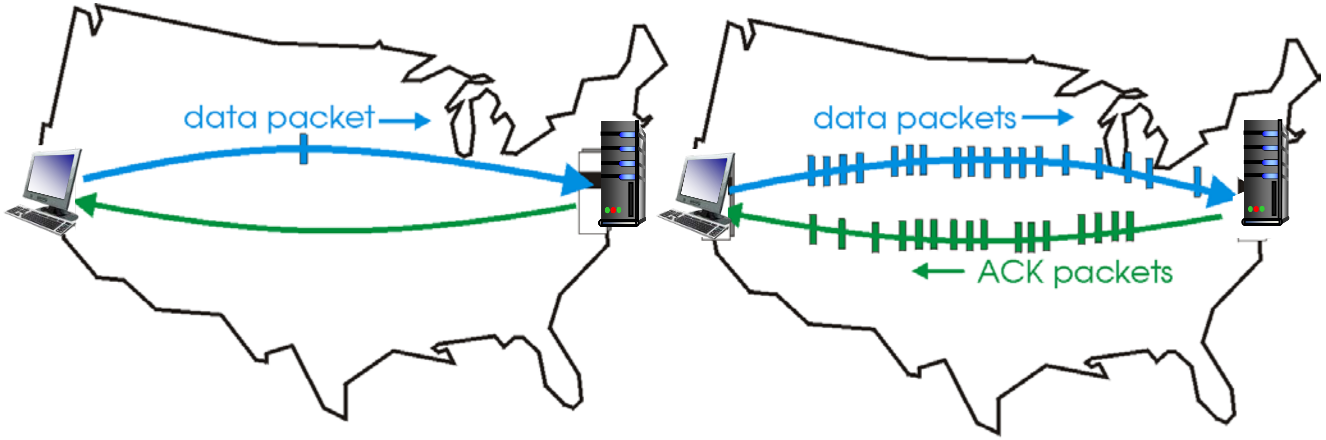
- Transport layer services in Internet
- Addressing, multiplexing/demultiplexing
- Connectionless, unreliable transport: UDP
- principles of reliable data transfer
  - Efficiency perspective

- *Next lecture: connection-oriented transport: TCP*
  - *reliable transfer*
  - *flow control*
  - *connection management*
  - *TCP congestion control*

Pipelined ack-based error-control protocols

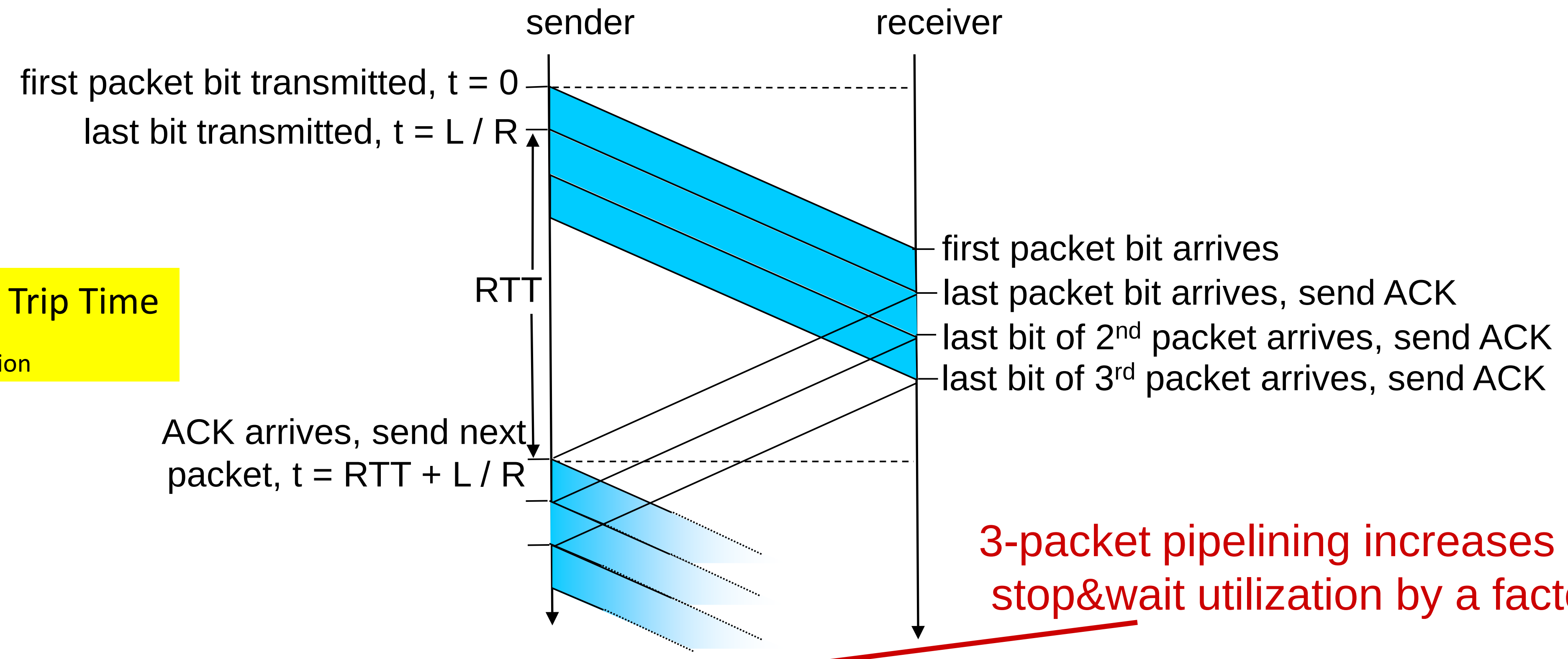
pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged pkts

- range of sequence numbers must be increased
- buffering at sender and/or receiver



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

Pipelining: increased utilization



RTT = Round Trip Time
= 2 * $T_{propagation}$

3-packet pipelining increases stop&wait utilization by a factor of 3!

$$U_{sender} = \frac{3L / R}{RTT + L / R} = \frac{.0024}{30.008} = 0.00081$$

Pipelined protocols: ack-based error control

two generic forms (i.e. ack + book-keeping policies) to deal with lost data:

- *go-Back-n*
- *selective repeat*

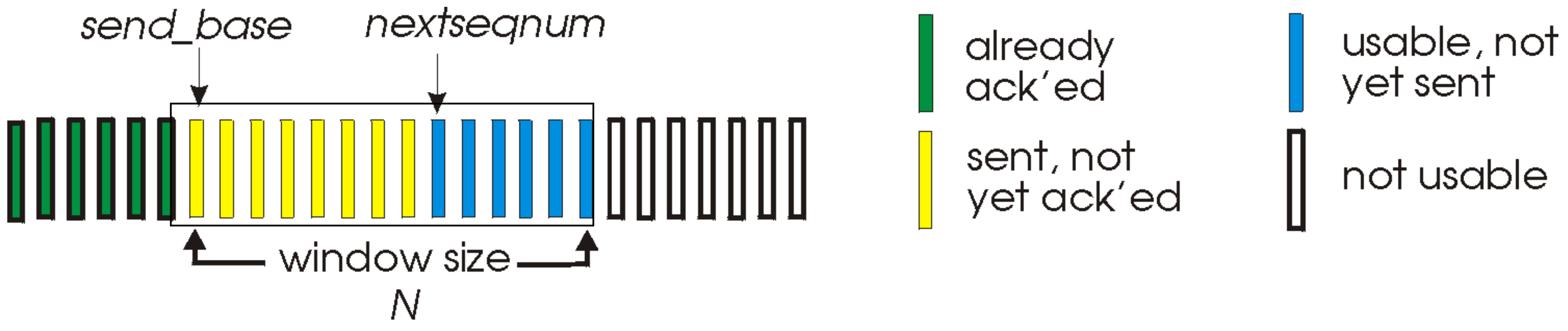
Go-Back-n

sender

- "window" of up to N, consecutive unack'ed pkts allowed

receiver

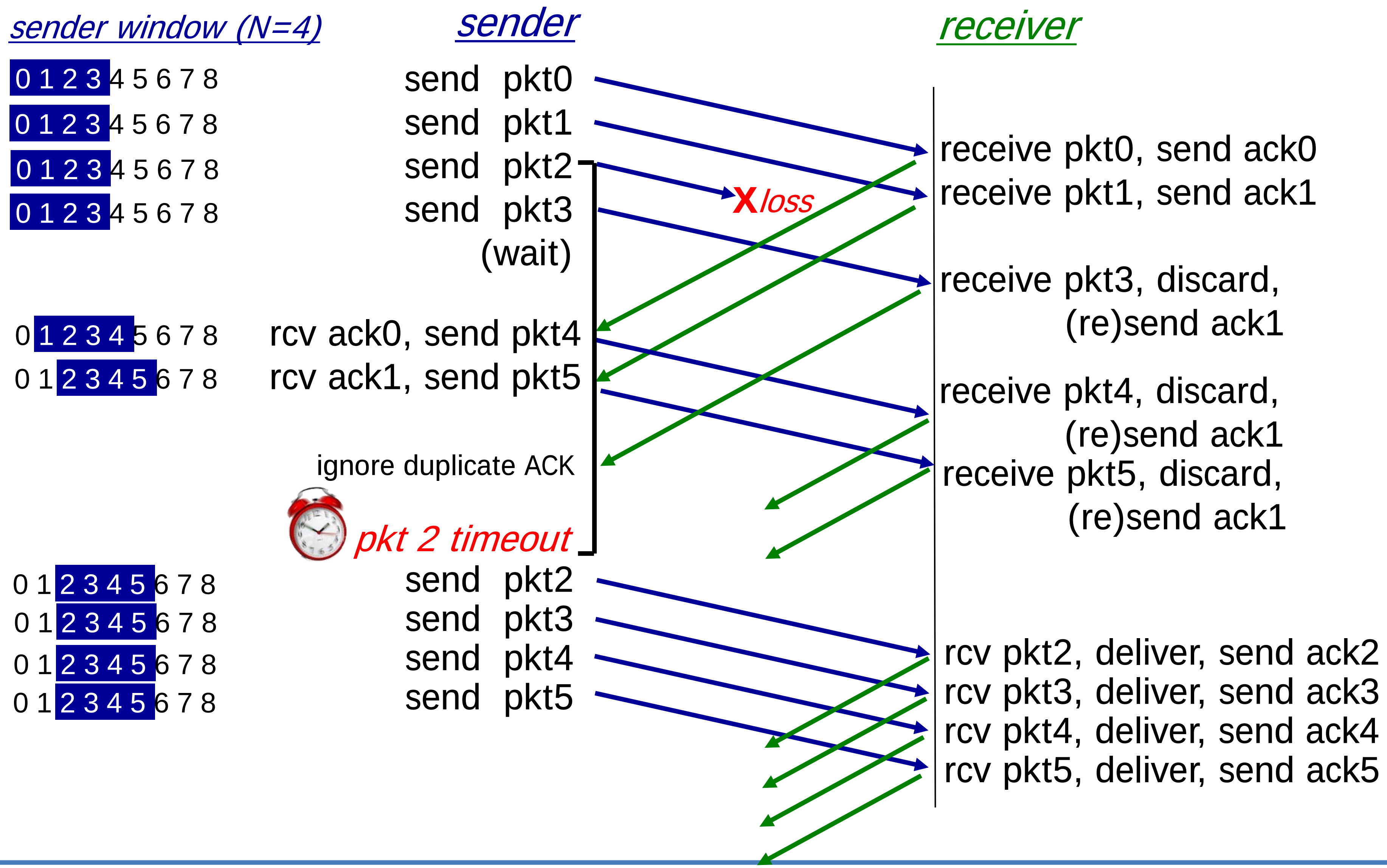
- Ack last correctly received pkt



- ACK(n): ACKs all pkts up to, including seq # n - *"cumulative ACK"*
  - may receive duplicate ACKs
- timer for oldest in-flight pkt
- *timeout(n):* retransmit packet n and all higher seq # pkts in window

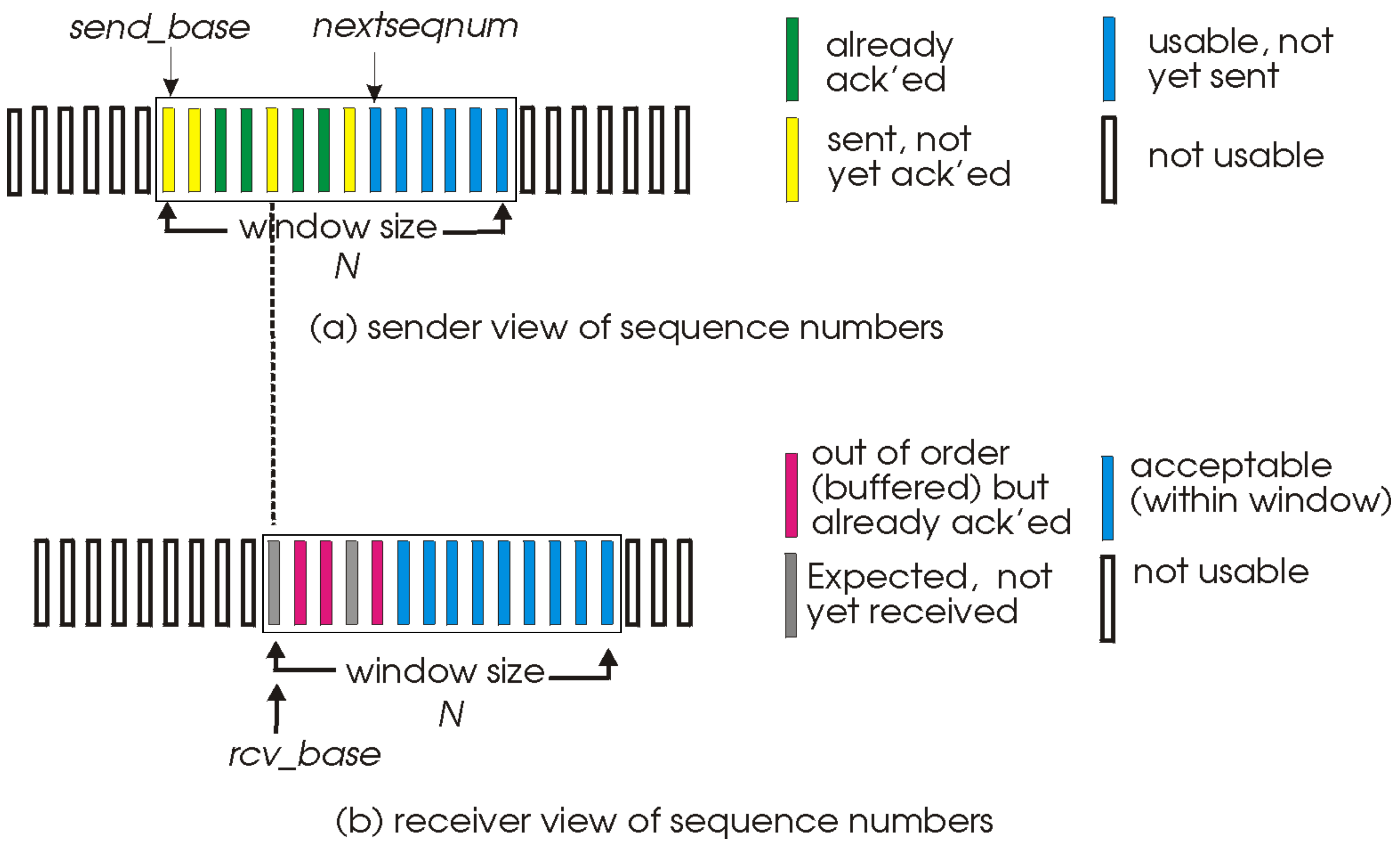
GBn in action

https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/go-back-n-protocol/index.html



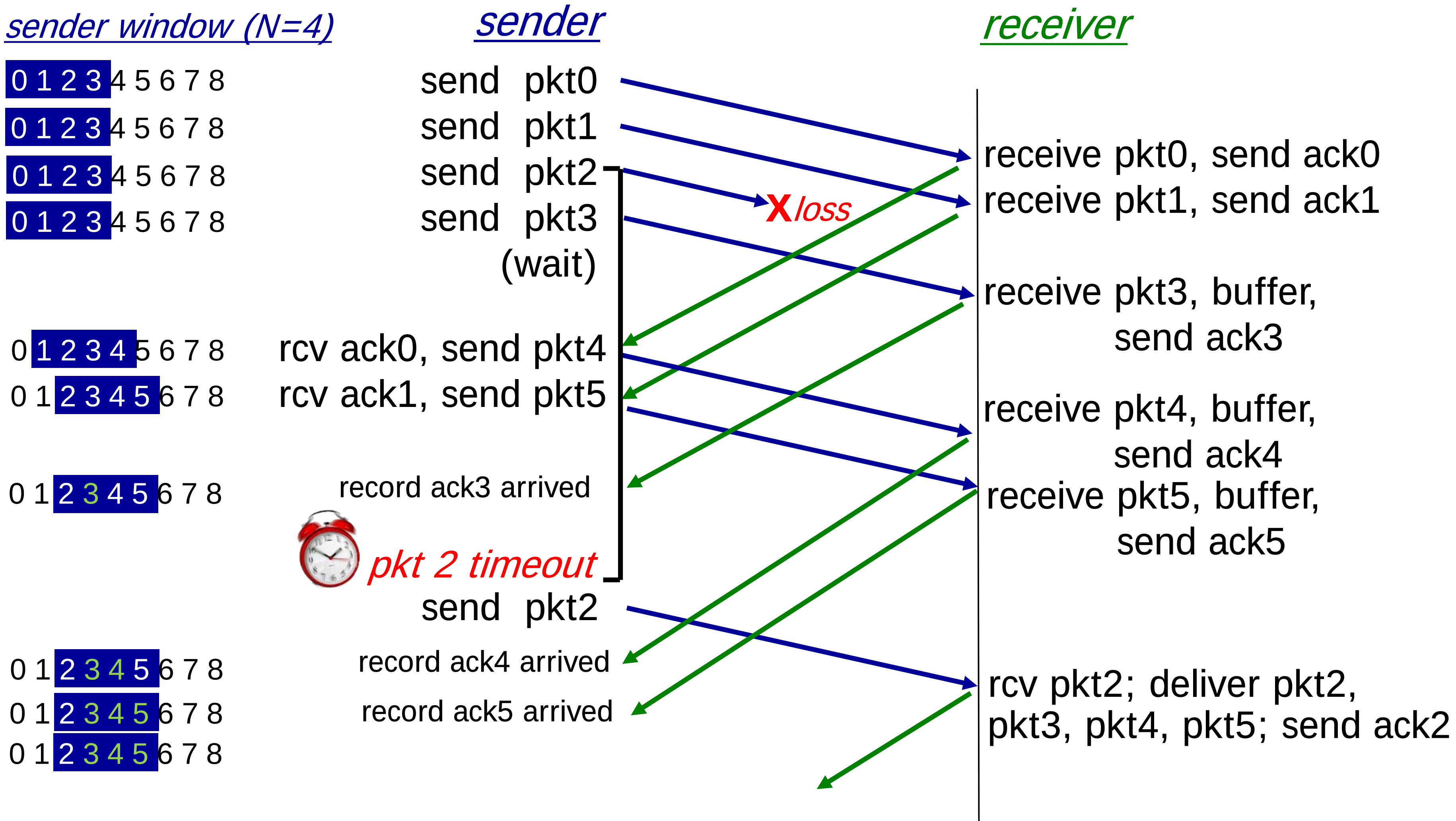
Selective repeat: sender, receiver windows

- receiver *individually* acknowledges received pkts
  - buffers pkts for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
  - Requires timer for each unACKed pkt



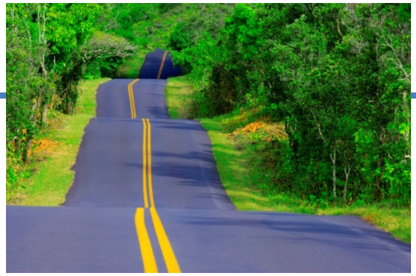
Selective repeat in action

https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/selective-repeat-protocol/index.html



Q: what happens when ack2 arrives?

Roadmap



- Transport layer services in Internet
- Addressing, multiplexing/demultiplexing
- Connectionless, unreliable transport: UDP
- principles of reliable data transfer
  - Efficiency perspective: pipelined protocols & error control through go-back-n, selective-repeat
    - Sequence numbers

- *Next: connection-oriented transport: TCP*
  - *reliable transfer*
  - *flow control*
  - *connection management*
  - *TCP congestion control*

Selective repeat: Sequence numbers

example:

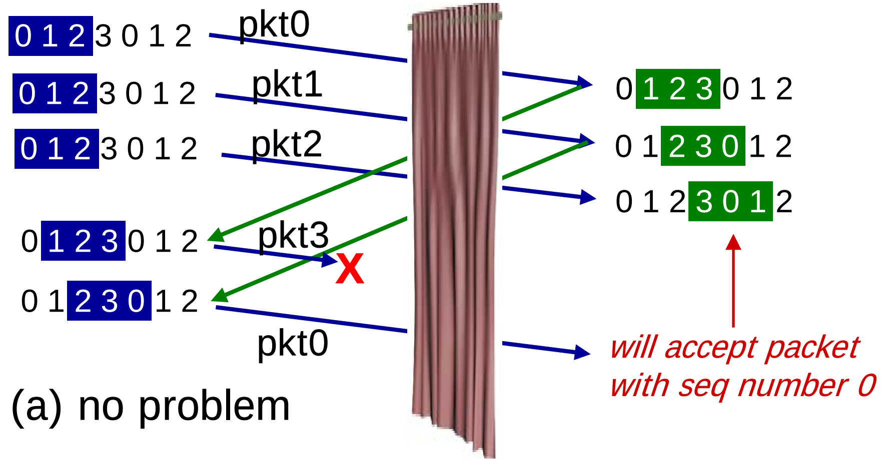
- seq #' s: 0, 1, 2, 3
- window size=3

❖ duplicate data accepted as new in (b)

Q: what relationship between seq # size and window size to avoid problem in (b)?

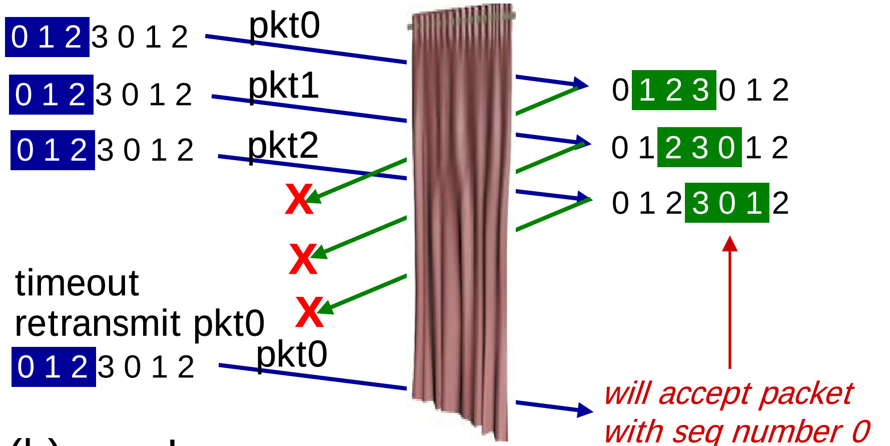
sender window (after receipt)

receiver window (after receipt)



(a) no problem

receiver can't see sender side.
receiver behavior identical in both cases!
something's (very) wrong!



(b) oops!

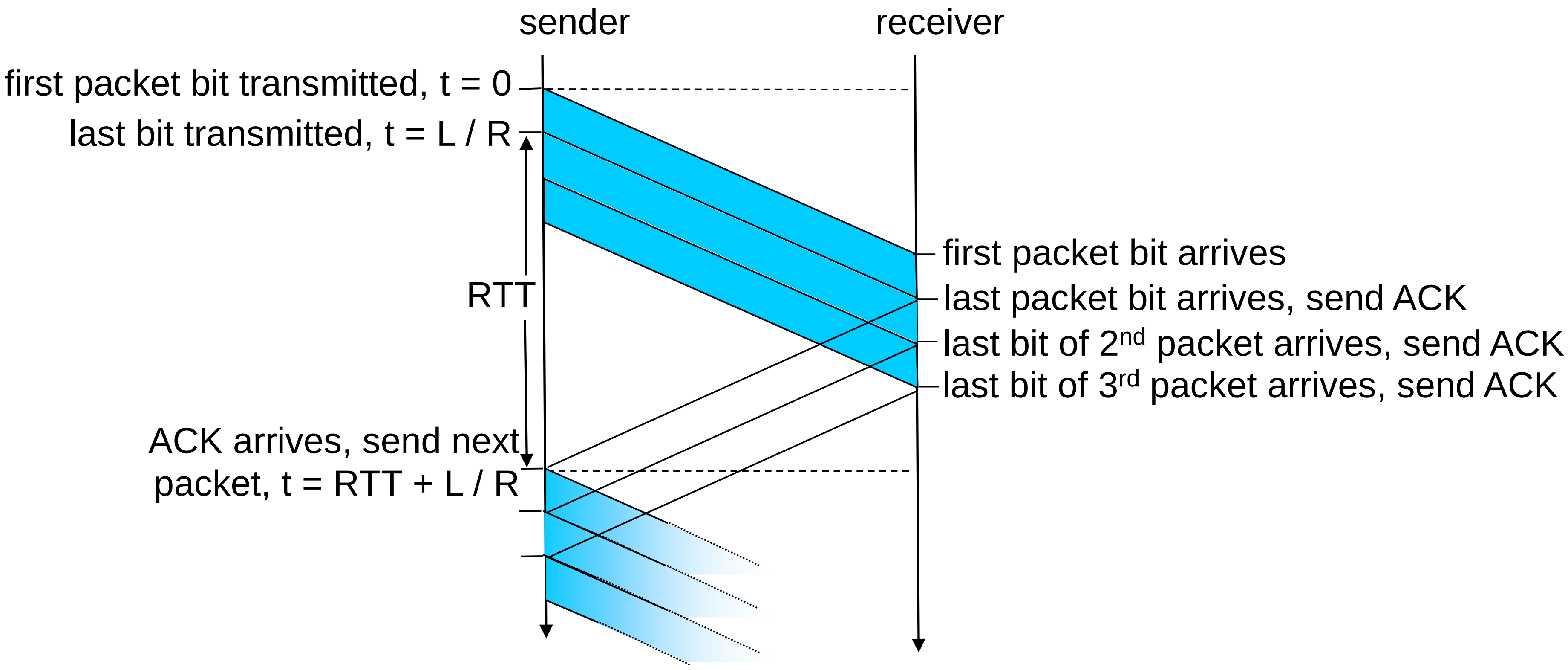
Question

What do Ack's achieve besides reliability?

Flow control: receiver can ack its receiving capacity i.e. **avoid swamping the receiver**

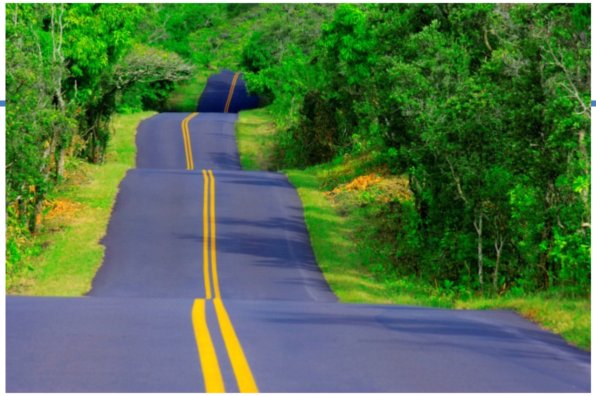
Flow control: Sender/receiver (ie network edge) issue: S cares to not overwhelm R

Ack-based pipelining => error-control & flow control at the same time!!!



Flow control: Sender/receiver problem; S cares to not overwhelm R

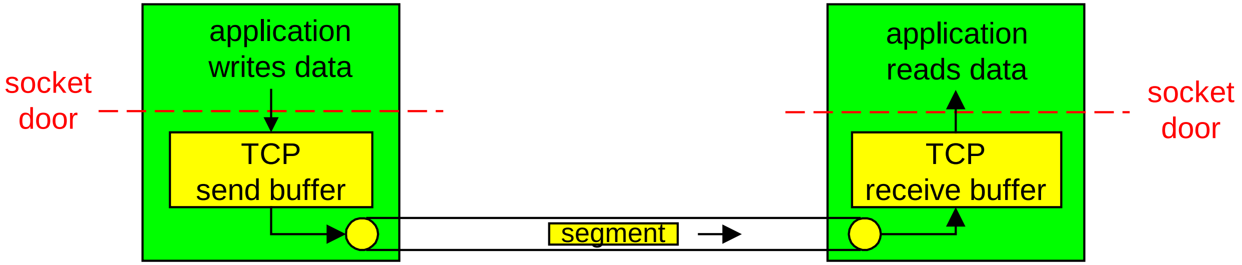
Roadmap Transport Layer



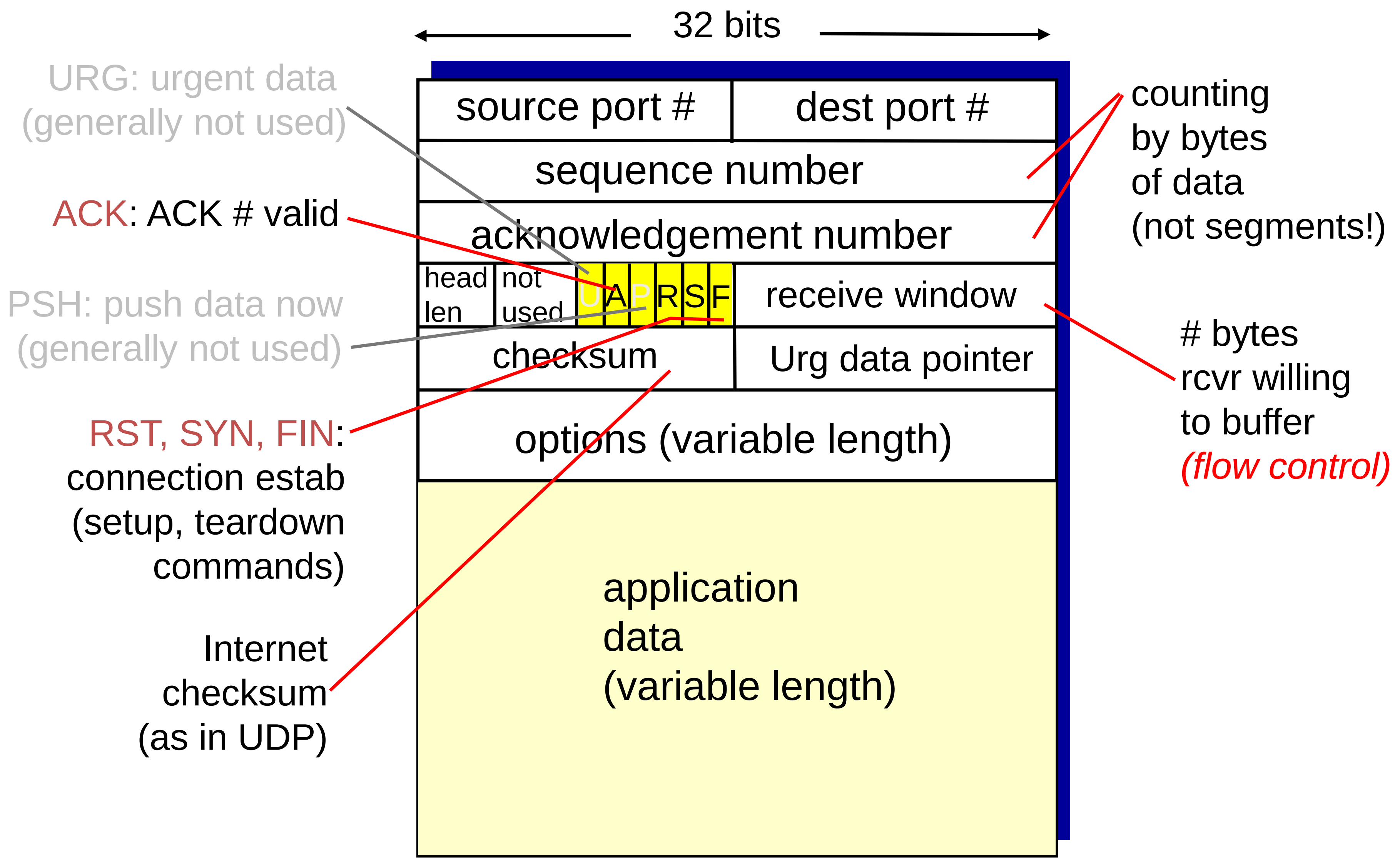
- transport layer services
- multiplexing/demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
  - reliable transfer
    - Acknowledgements
    - Retransmissions
    - Connection management
    - Flow control and buffer space
  - Congestion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

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

- point-to-point & full duplex data:
  - one sender, one receiver
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented, reliable, in-order byte steam:
  - Needs handshaking (exchange of control msgs); inits sender & receiver state before data exchange
- Flow&error control: ack-based, pipelined:
- (+ extra) congestion control:
  - sender will not flood network

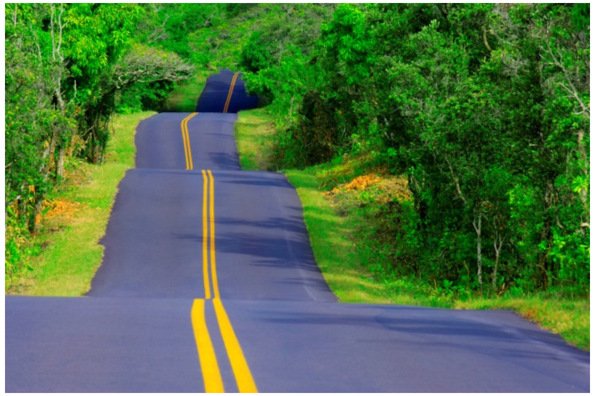


TCP segment structure



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

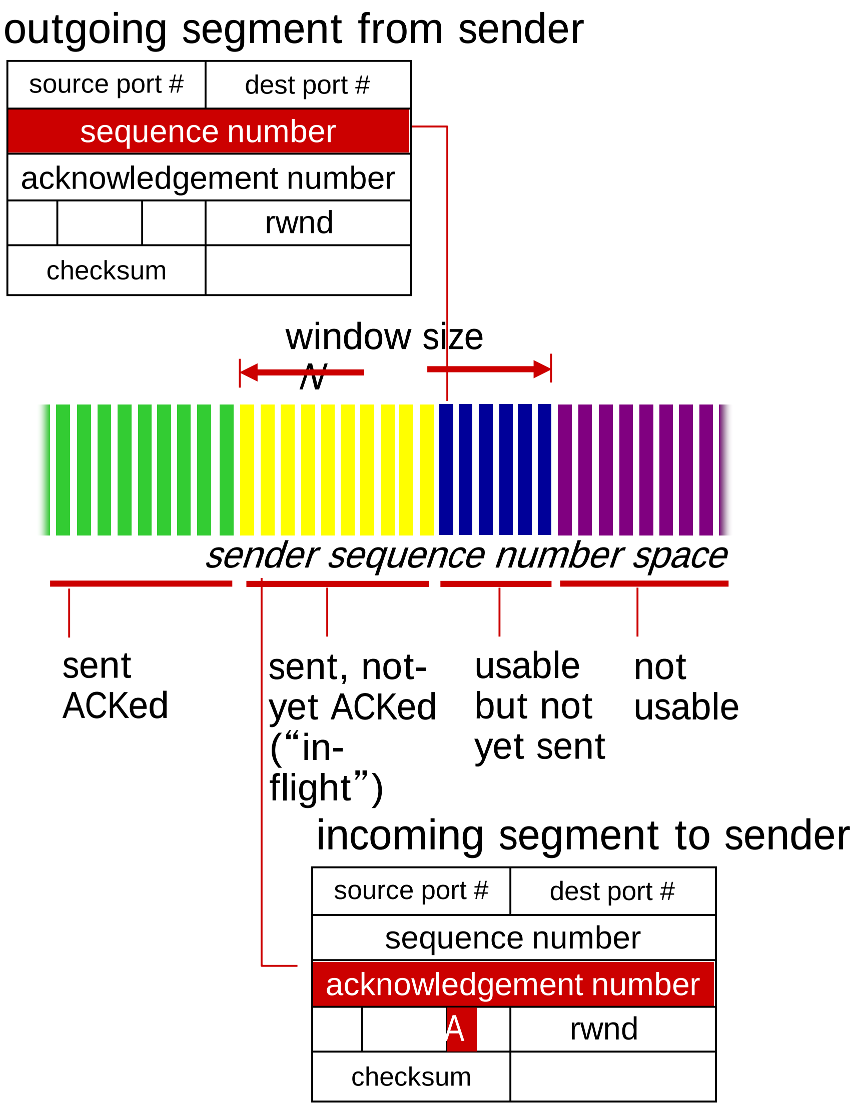
<u>sequence numbers:</u>

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

<u>acknowledgements:</u>

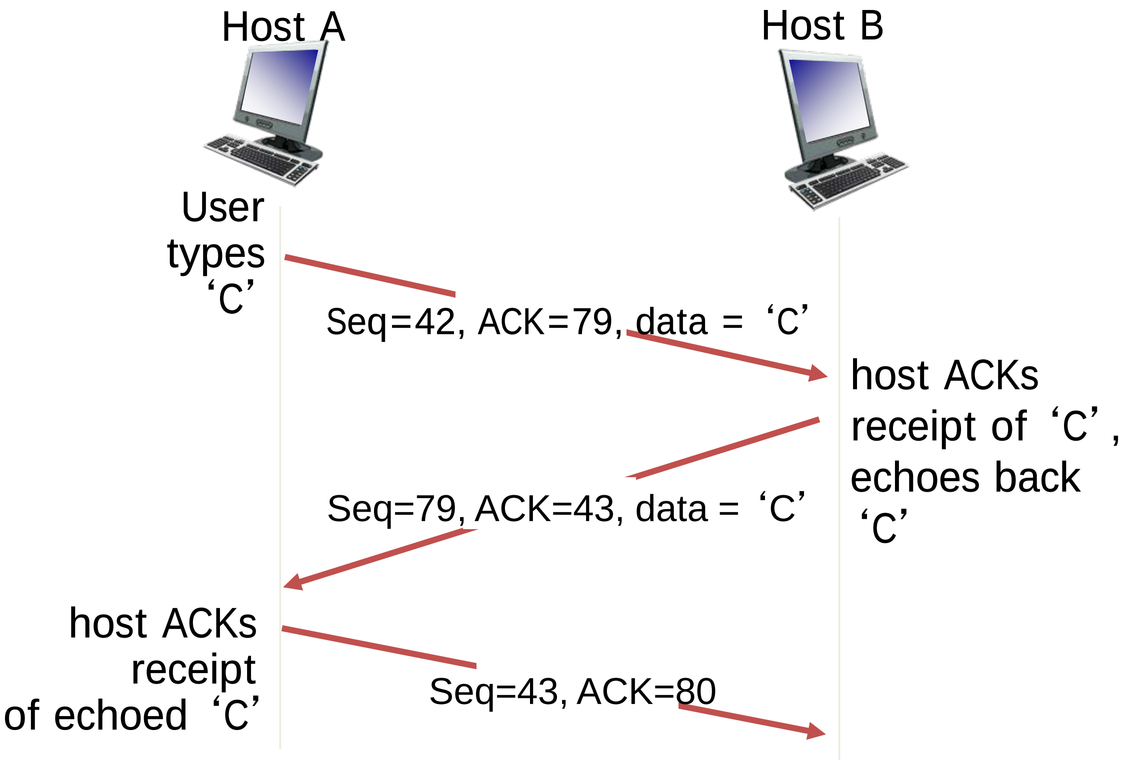
– seq # of next byte expected from other side

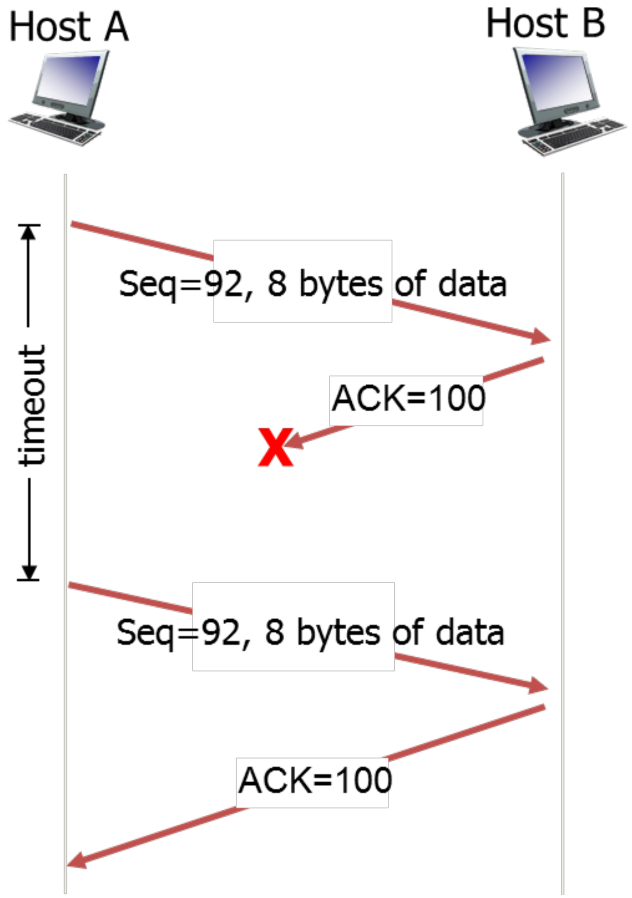
– **cumulative ACK**



TCP seq. numbers, ACKs

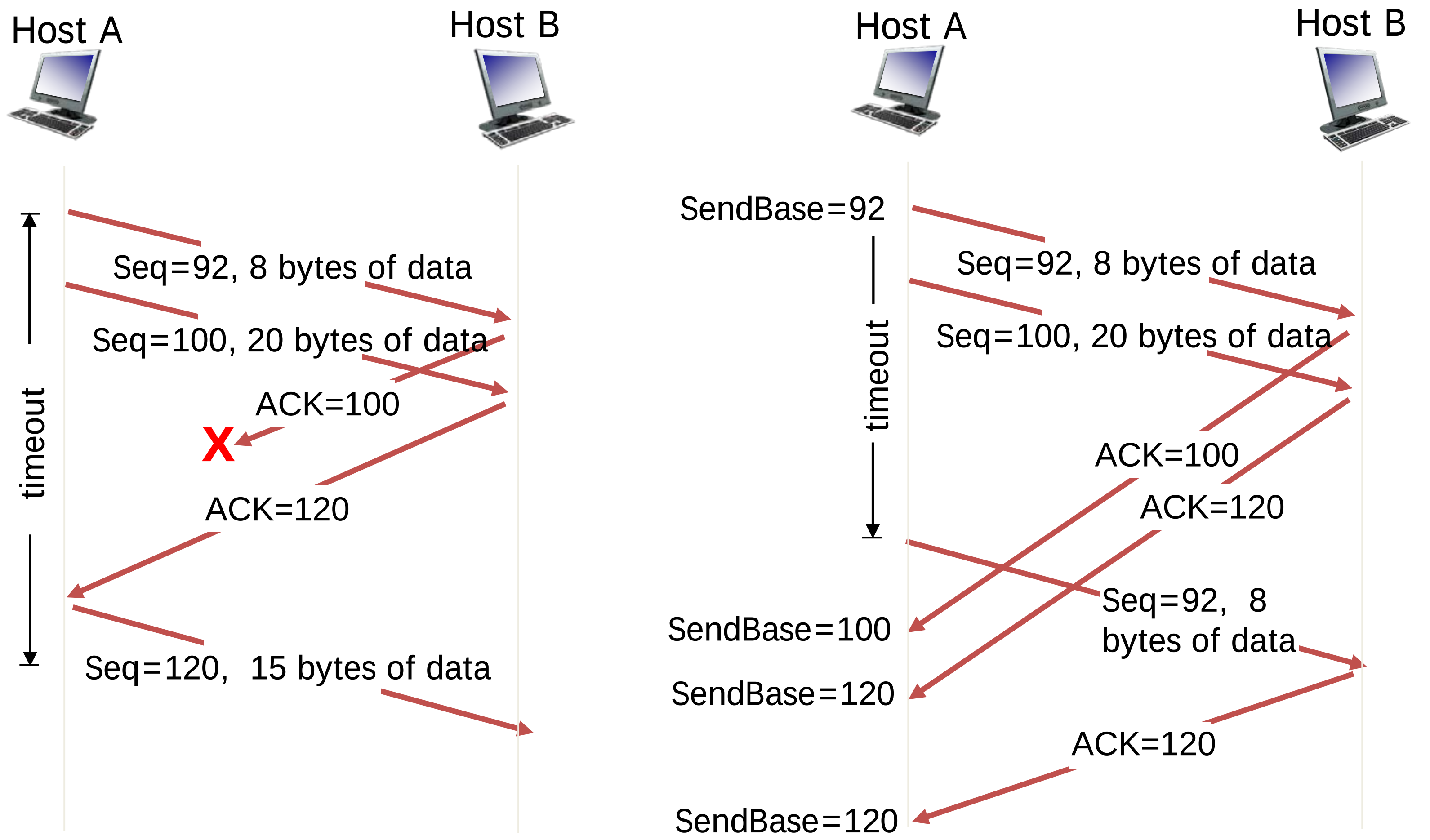
Always ack next in-order expected byte





Simple example scenario
Based on telnet msg exchange

TCP: cumulative Ack - retransmission scenarios



Cumulative ACK

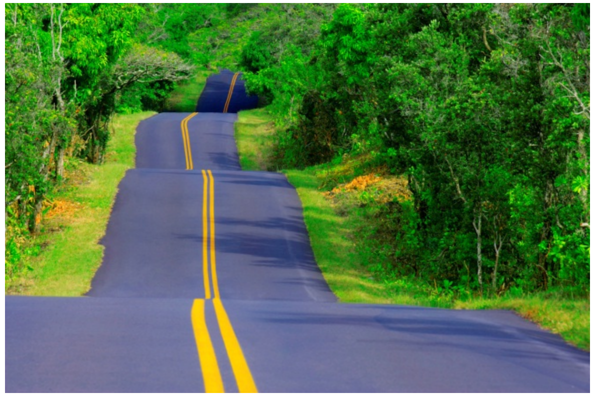
(Premature) timeout

TCP ACK generation [RFC 1122, RFC 5681]

Event	TCP Receiver action
in-order segment arrival, no gaps, everything else already ACKed	**Delayed ACK**. Wait max 500ms for next segment then 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

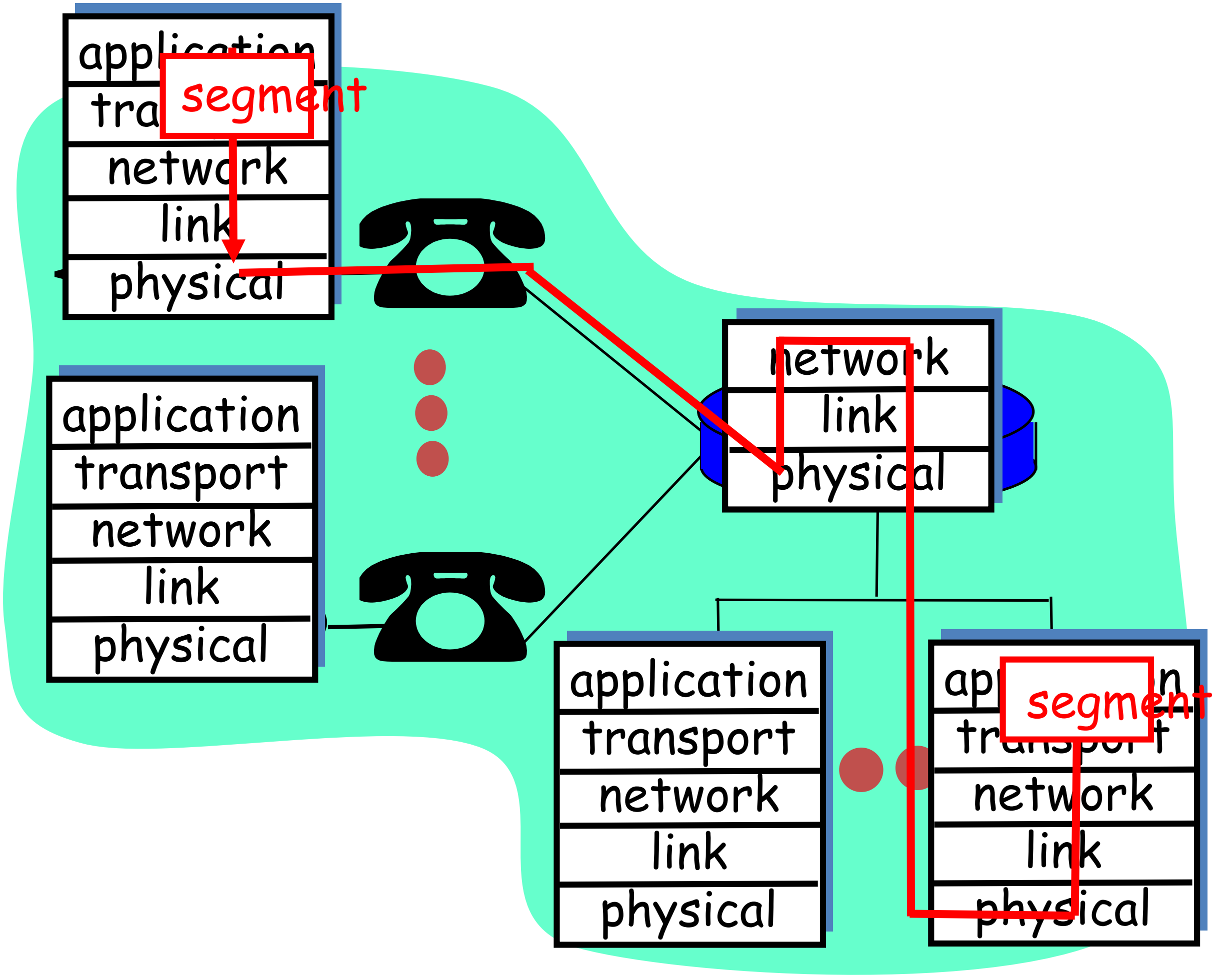
Roadmap Transport Layer

- 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



Q: how to set TCP timeout value?

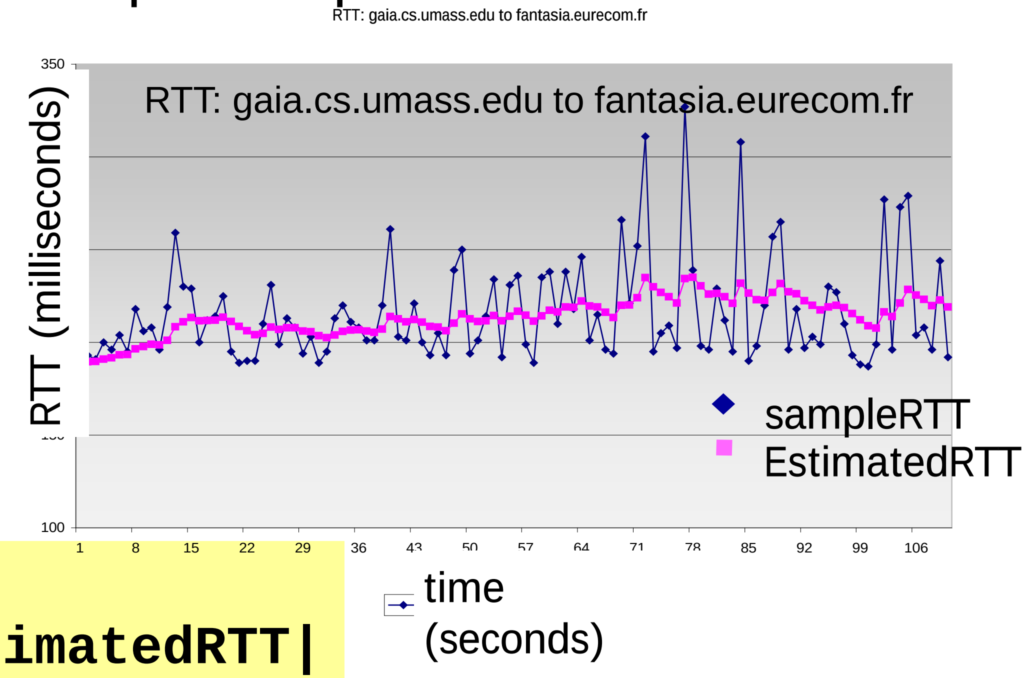
- longer than end-to-end RTT
  - but that varies!!!
- *too short* timeout*:*
  - premature, unnecessary retransmissions
- *too long:*
  - slow reaction to loss



TCP round trip time, timeout estimation

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

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




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

(typically, β = 0.25)

`TimeoutInterval = EstimatedRTT + 4*DevRTT`

"safety margin"

TCP fast retransmit (RFC 5681)

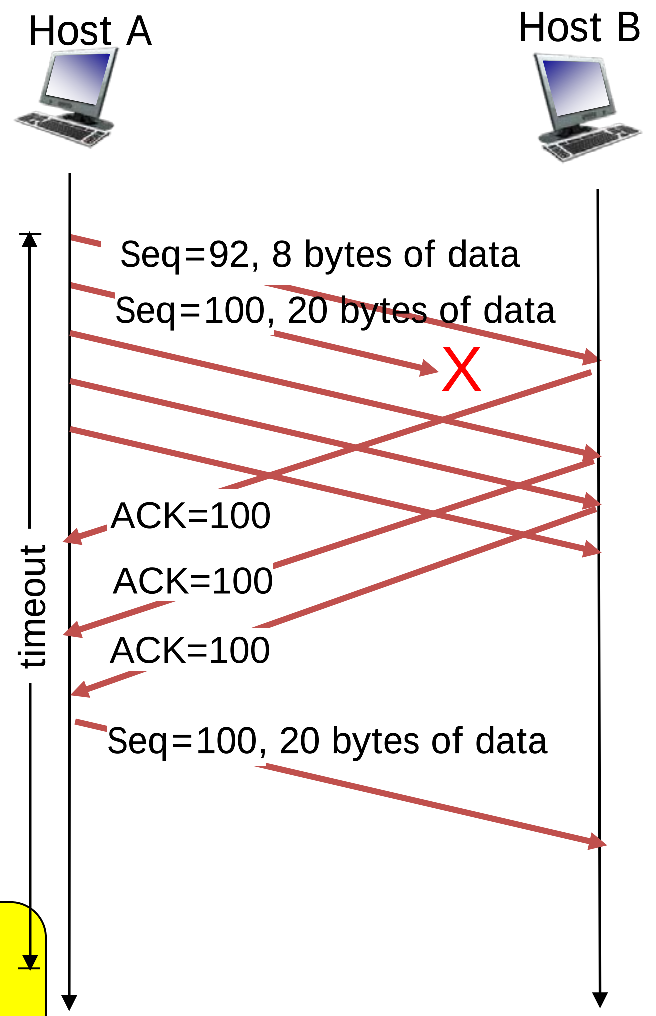
- time-out can be long:
  - long delay before resending lost packet
- IMPROVEMENT: detect lost segments via duplicate ACKs

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?



Q: Is TCP stateful or stateless?

Is it possible to have reliable transfer over UDP?

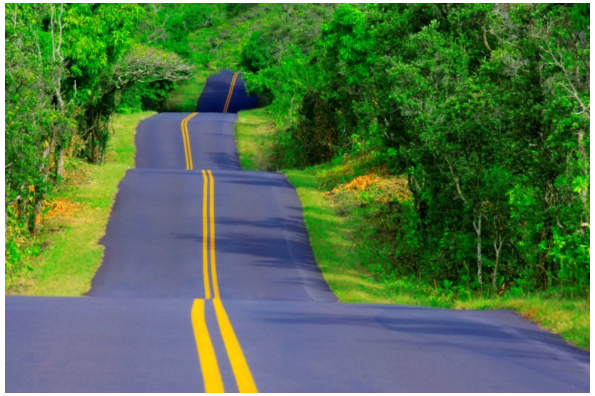
reliable transfer over UDP?

- add reliability at application layer
- top of UDP

ons and implementations for types of
ed and not need to be implemented
atch)

Roadmap Transport Layer

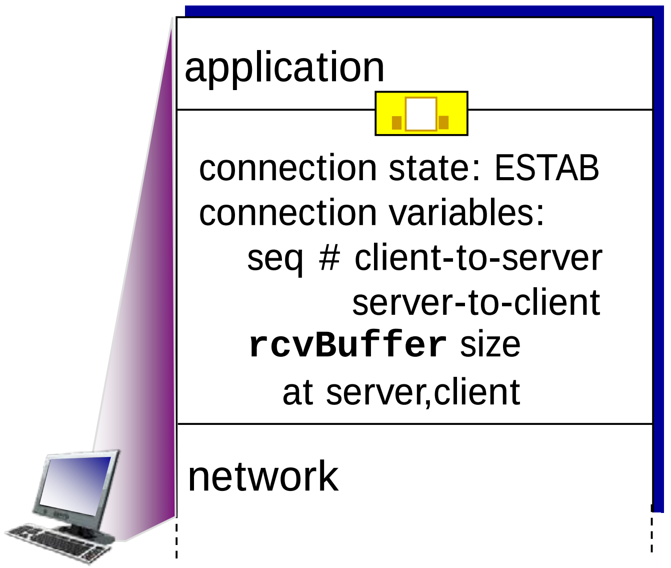
- transport layer services
- multiplexing/demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
  - reliable transfer
    - Acknowledgements
    - Retransmissions
    - Connection management
    - Flow control and buffer space
  - Congestion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



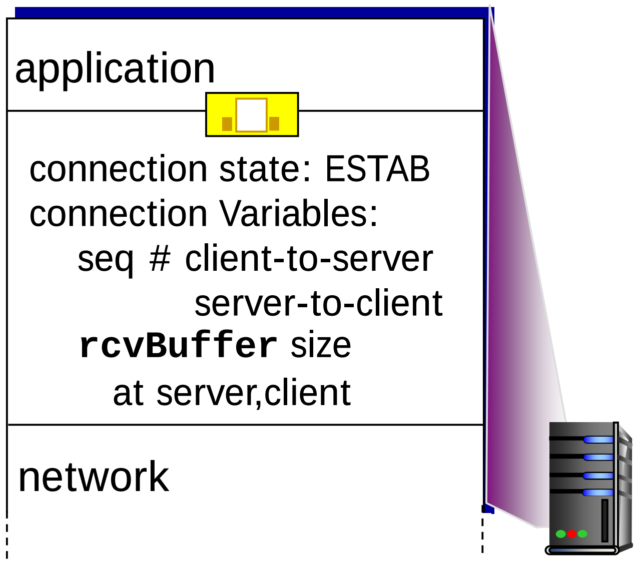
Connection Management

before exchanging data, sender/receiver "handshake":

- agree to establish connection + connection parameters

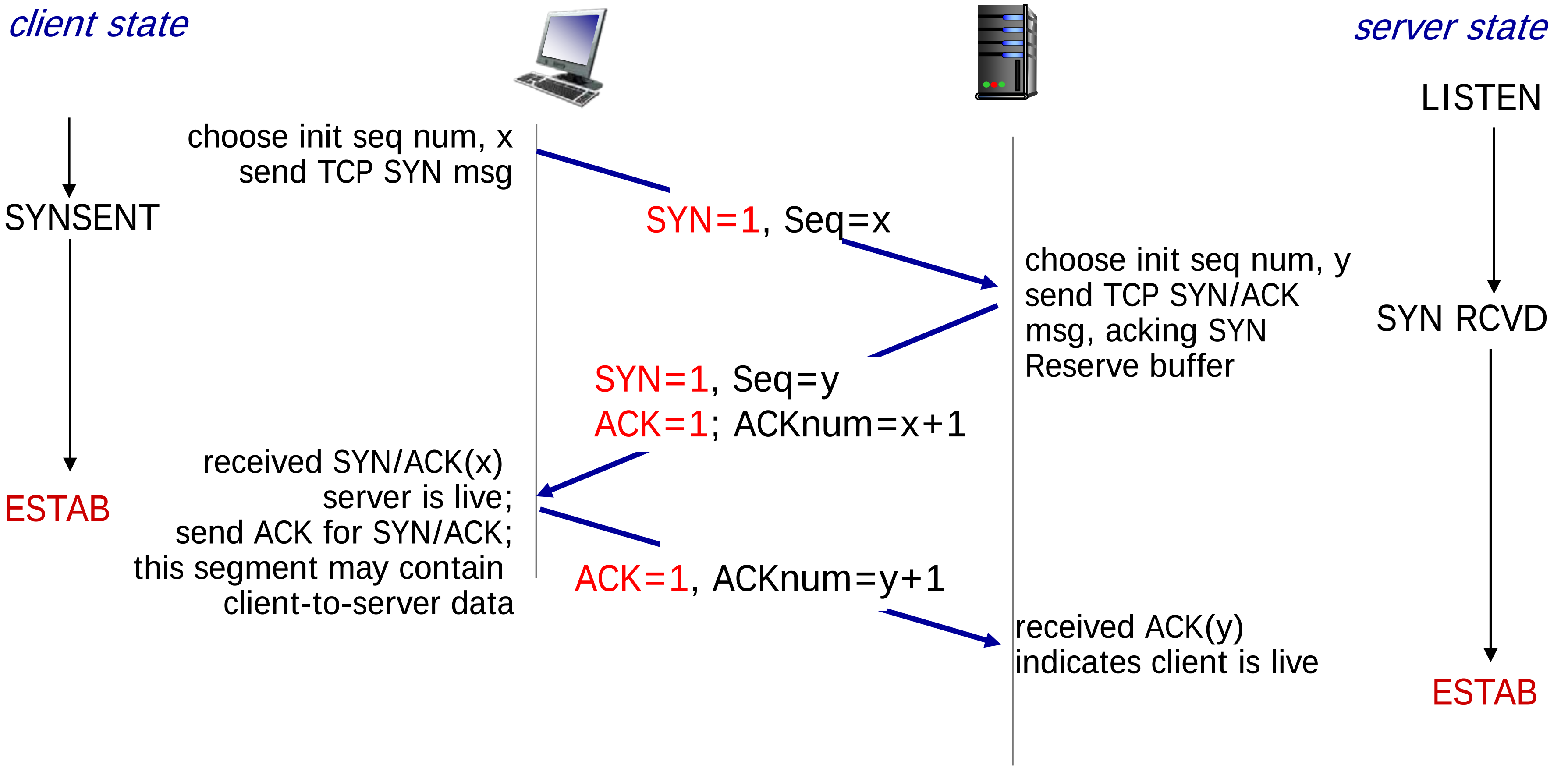



```
Socket clientSocket =
  newSocket("hostname","port
   number");
```

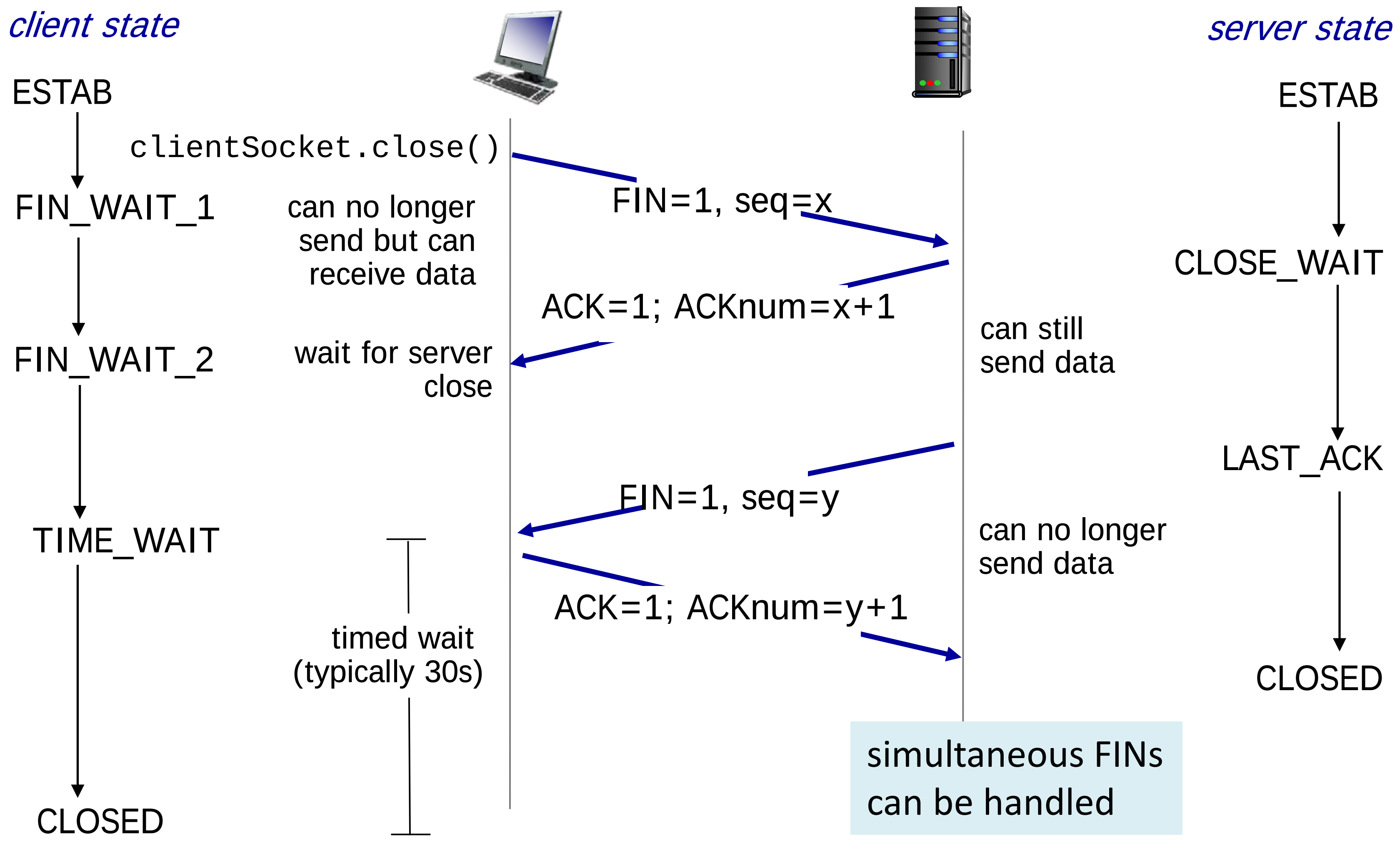



```
Socket connectionSocket =
   welcomeSocket.accept();
```

Setting up a connection: TCP 3-way handshake



TCP: closing a connection

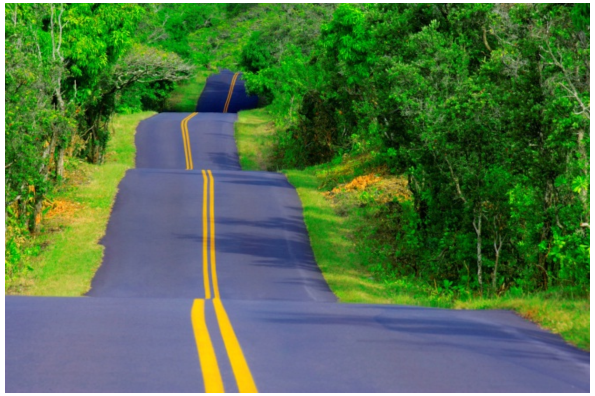


simultaneous FINs can be handled

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

Roadmap Transport Layer

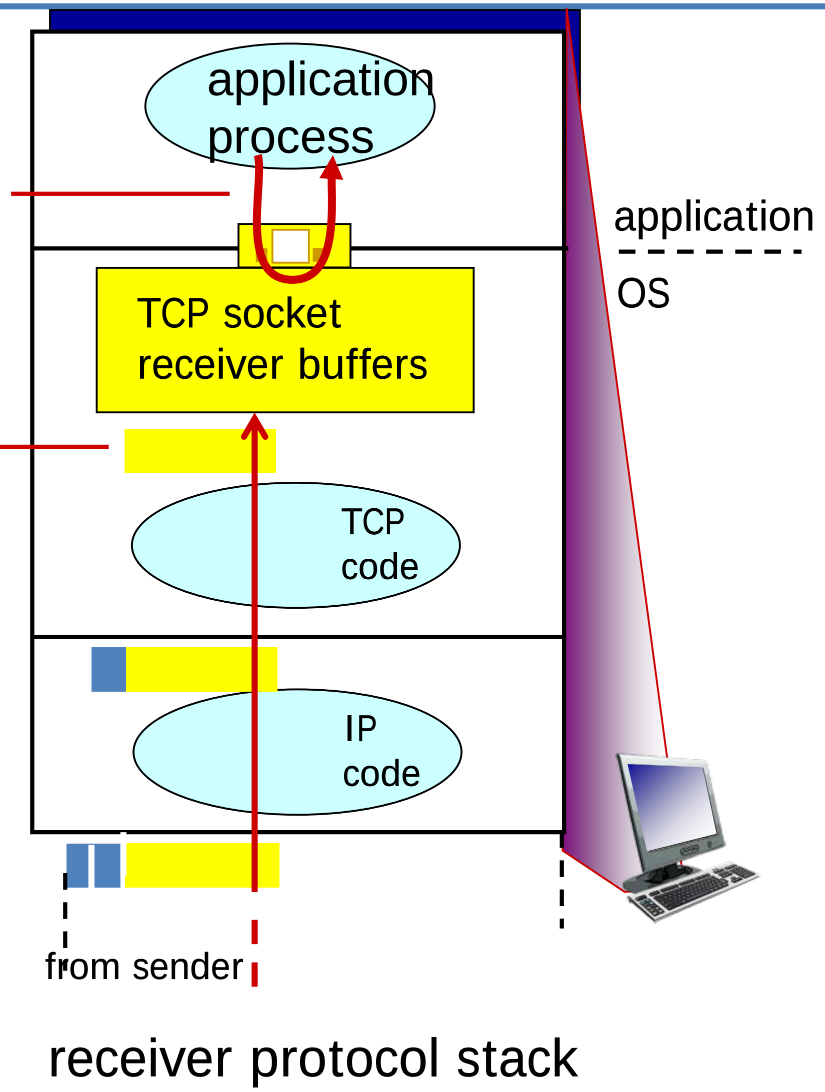
- transport layer services
- multiplexing/demultiplexing
- connectionless transport: UDP
- principles of reliable data transfer
- connection-oriented transport: TCP
  - reliable transfer
    - Acknowledgements
    - Retransmissions
    - Connection management
    - Flow control and buffer space
  - Congestion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



TCP flow control

application might remove data from TCP socket buffers ....

... what if slower than TCP is delivering (i.e. slower than sender is sending)



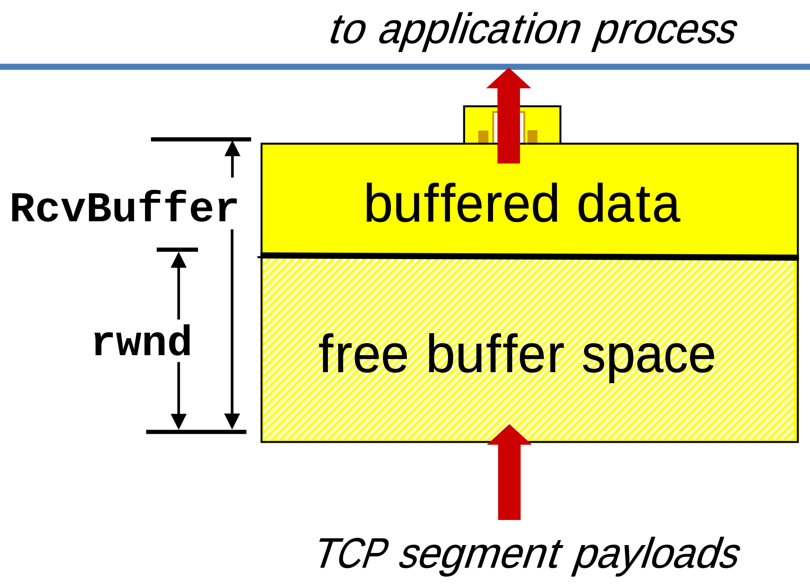
flow control

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

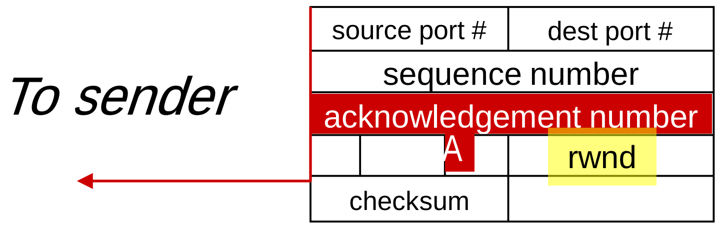
TCP flow control

- receiver "advertises" free buffer space through **rwnd** value in header
  - **RcvBuffer** size set via socket options (typical default 4 Kbytes)
  - OS can autoadjust **RcvBuffer**

- sender limits unacked ("in-flight") data to receiver's **rwnd** value
  - s.t. receiver's buffer will not overflow



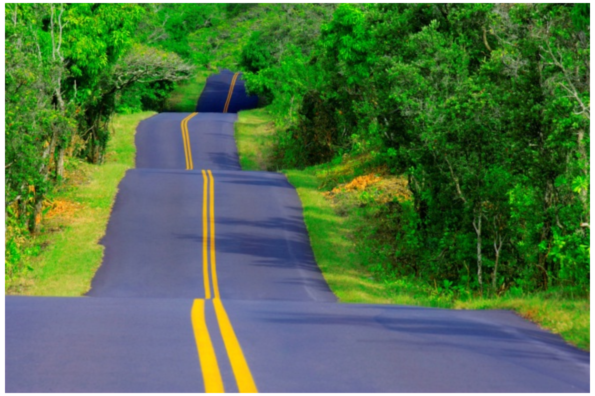
receiver-side buffering



3-36

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



Question:

Difference between congestion control and flow control?

Congestion control = ***Avoid congesting the network***

Congestion is network-core issue

in contrast to

flow-control, which is sender-receiver (i.e. network edge) issue

Principles of congestion control

congestion:

- informally: "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)

Distinction between flow control and congestion control

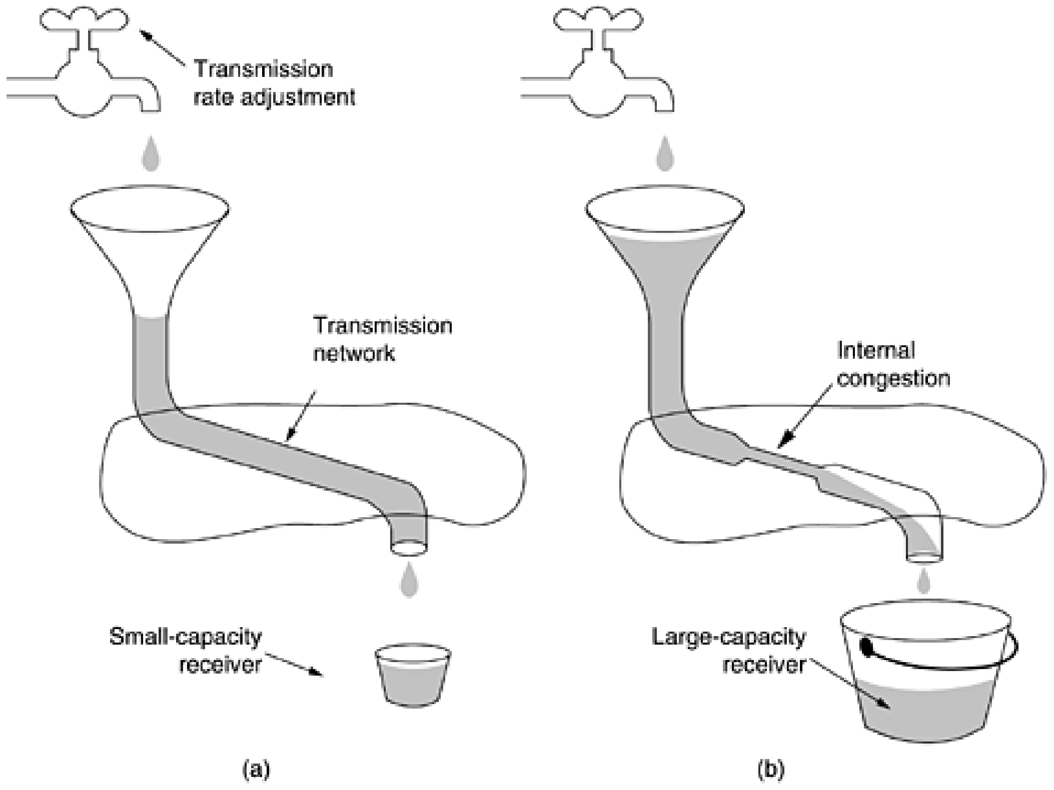
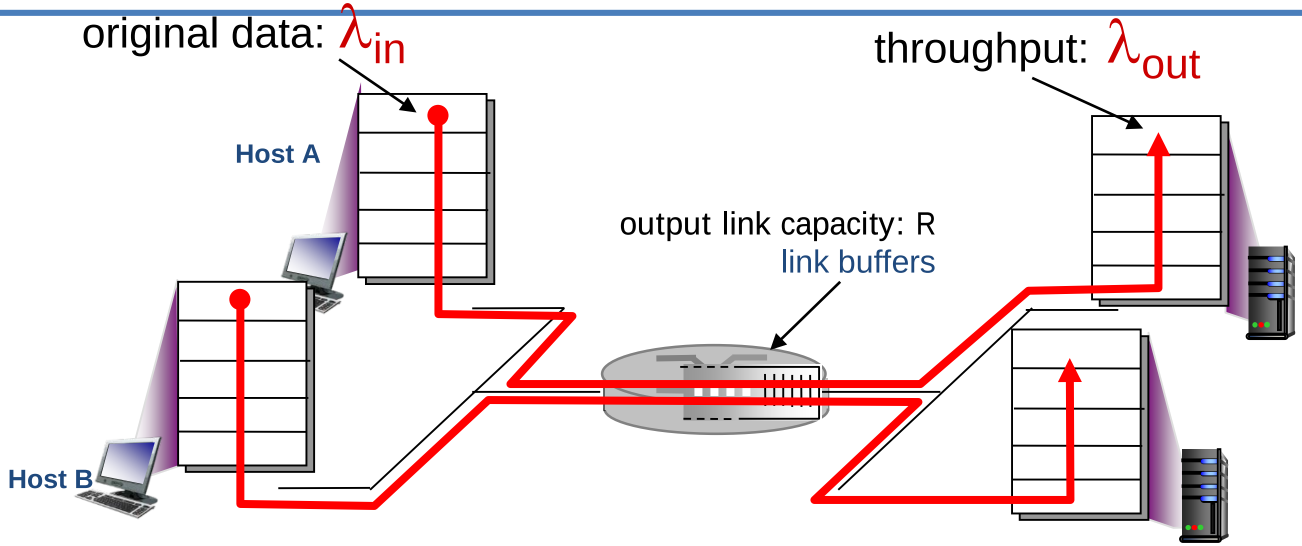


Fig. A. Tanenbaum
Computer Networks

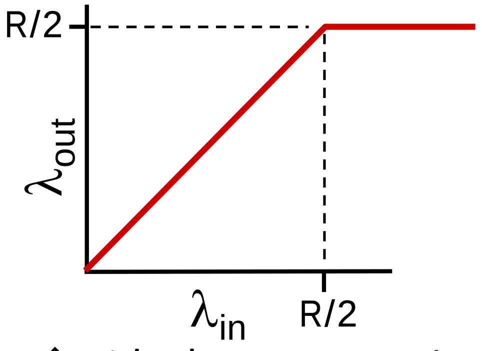
Need for flow control

Need for congestion control

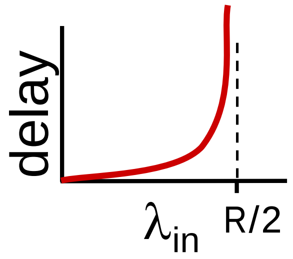
Causes/costs of congestion

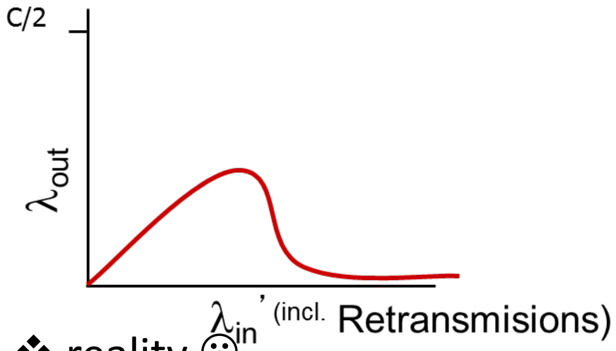


- Recall queueing behaviour + losses
- Losses => retransmissions => even higher load…



- Ideal per-connection throughput: R/2 (if 2 connections)





- reality ☹

Approaches towards congestion control

approach taken by TCP

end-end congestion control:

- ❖ no explicit feedback from network
- ❖ congestion inferred from end-system observed loss, delay

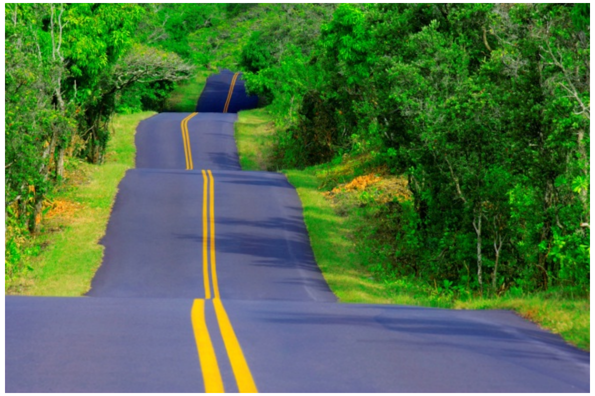
Not present in Internet's network layer protocols

network-assisted congestion control:

- ❖ routers collaborate for optimal rates + 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 (AIMD)

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

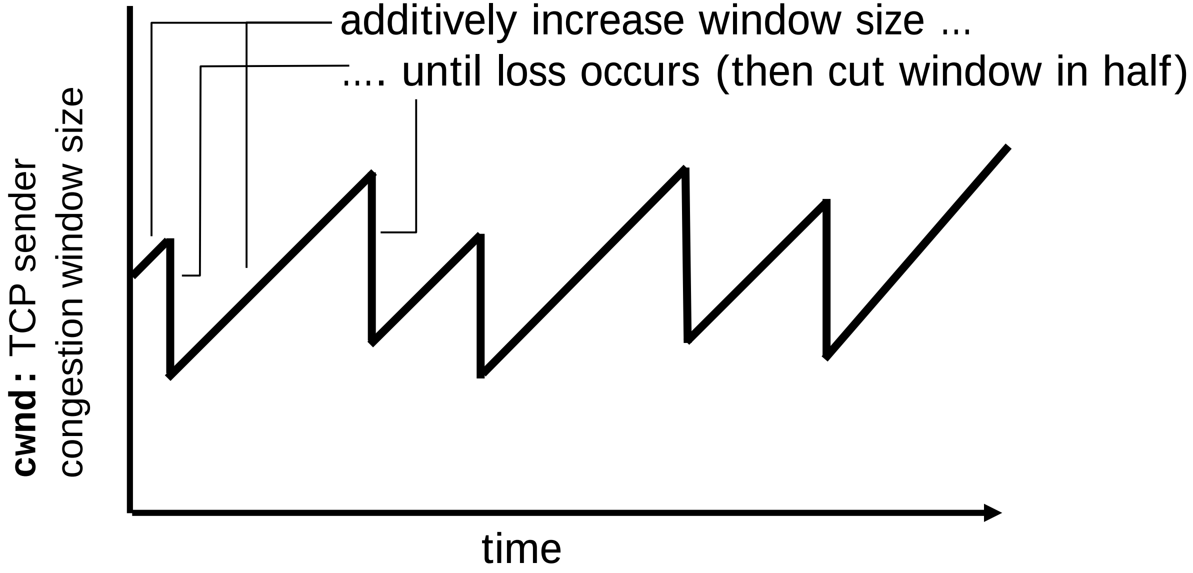
How does sender perceive congestion?

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

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

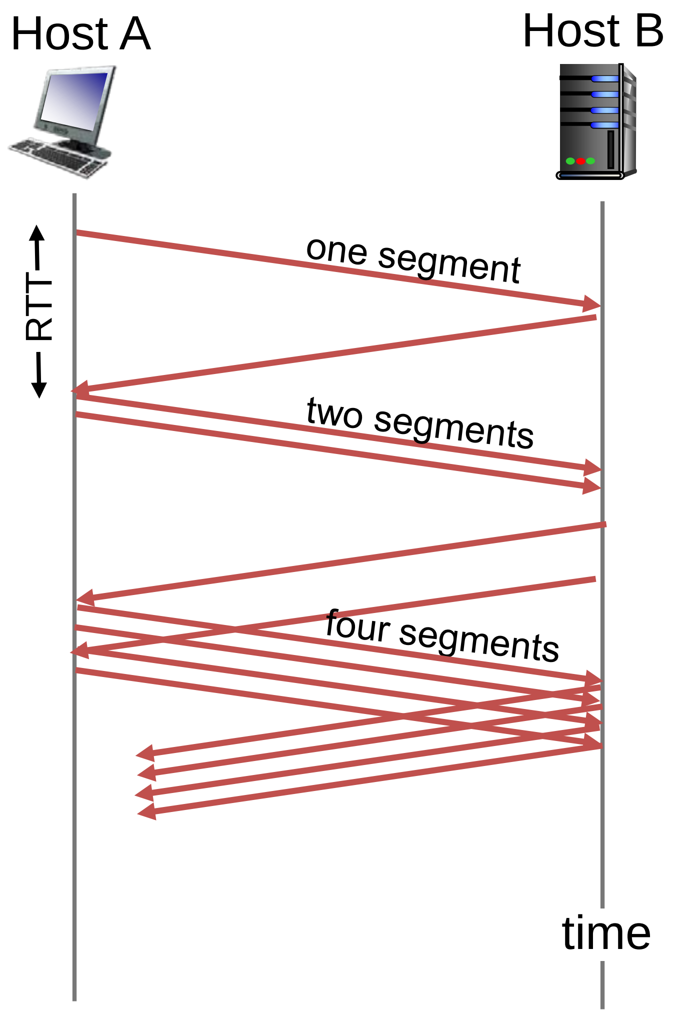
- *Additive Increase:* increase **cwnd** by 1 MSS every RTT until loss detected
- *Multiplicative Decrease*: cut **cwnd** in half after loss
- **To start with: slow start**

AIMD saw tooth behavior: probing for bandwidth

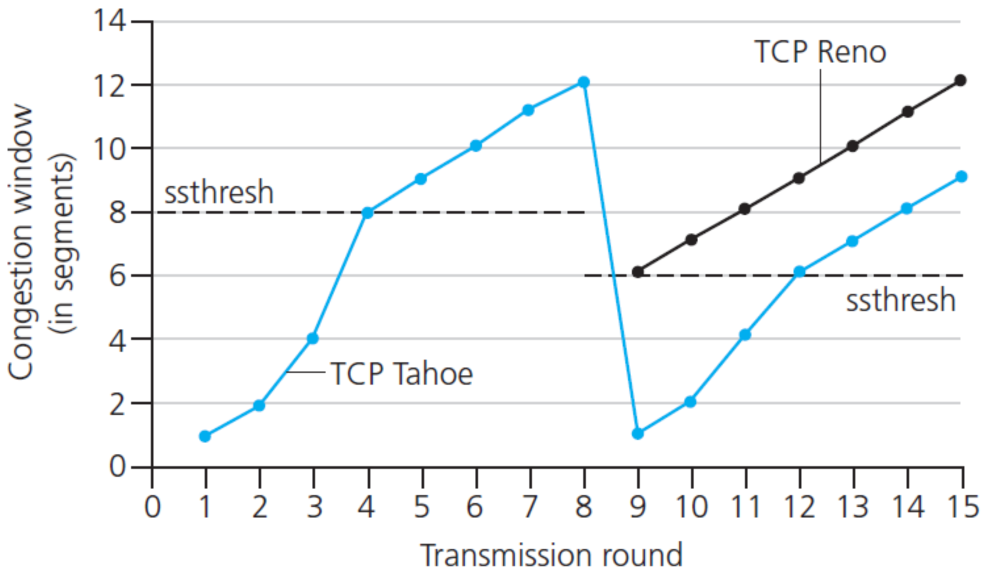


TCP Slow Start

- when connection begins, increase rate exponentially until first loss event:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every ack of previous "batch"
  - done by incrementing **cwnd** for every ACK received
- *summary:* initial rate is slow but ramps up exponentially fast
  - then, saw-tooth



TCP cwnd: from exponential to linear growth + reacting to loss



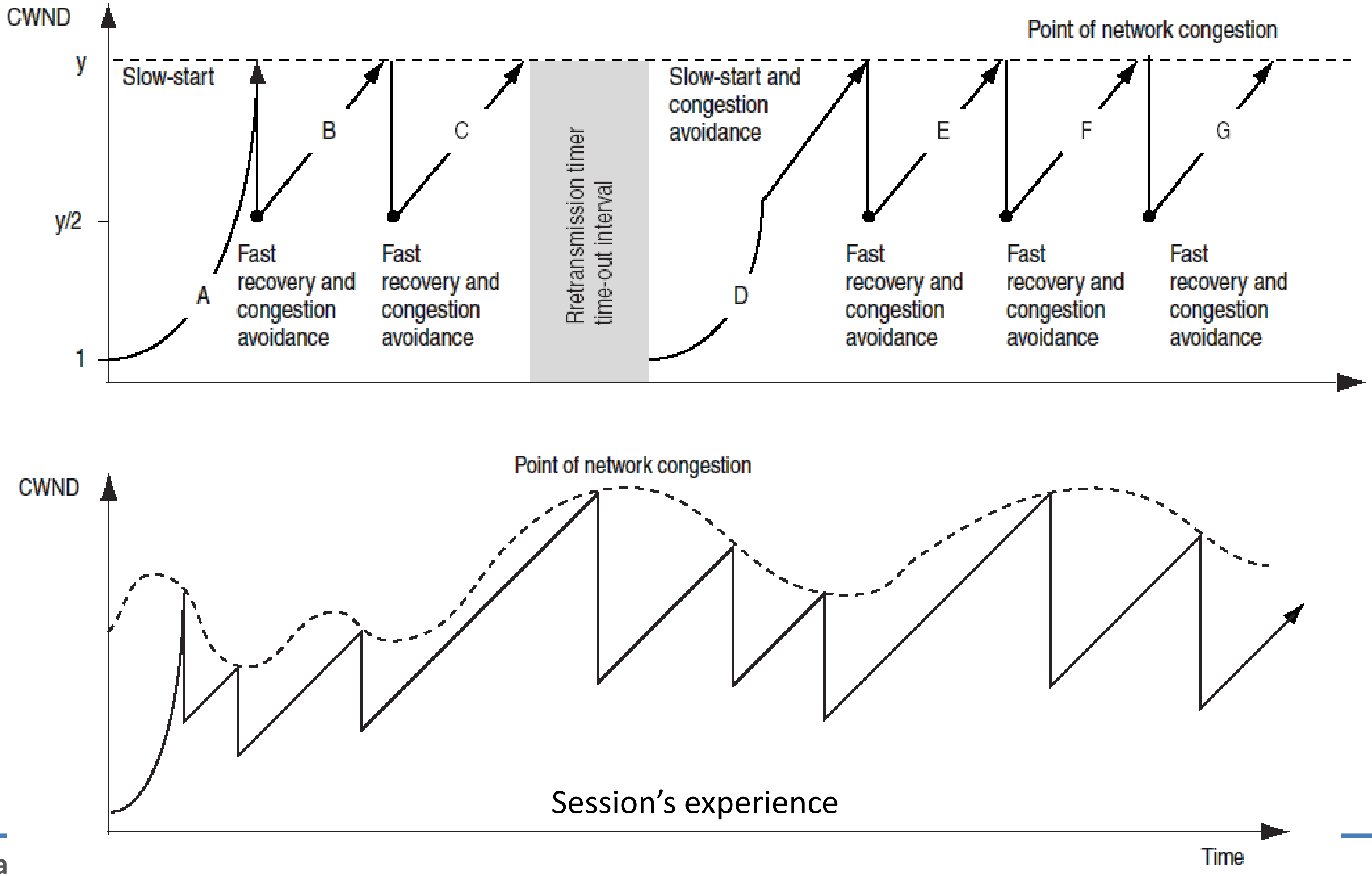
Reno: loss indicated by timeout or 3 duplicate ACKs: cwnd is cut in half; then grows linearly

Implementation:

- variable **ssthresh** (slow start threshold)
- on loss event, **ssthresh** = ½***cwnd**

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

TCP's throughput (Fast recovery - Reno)

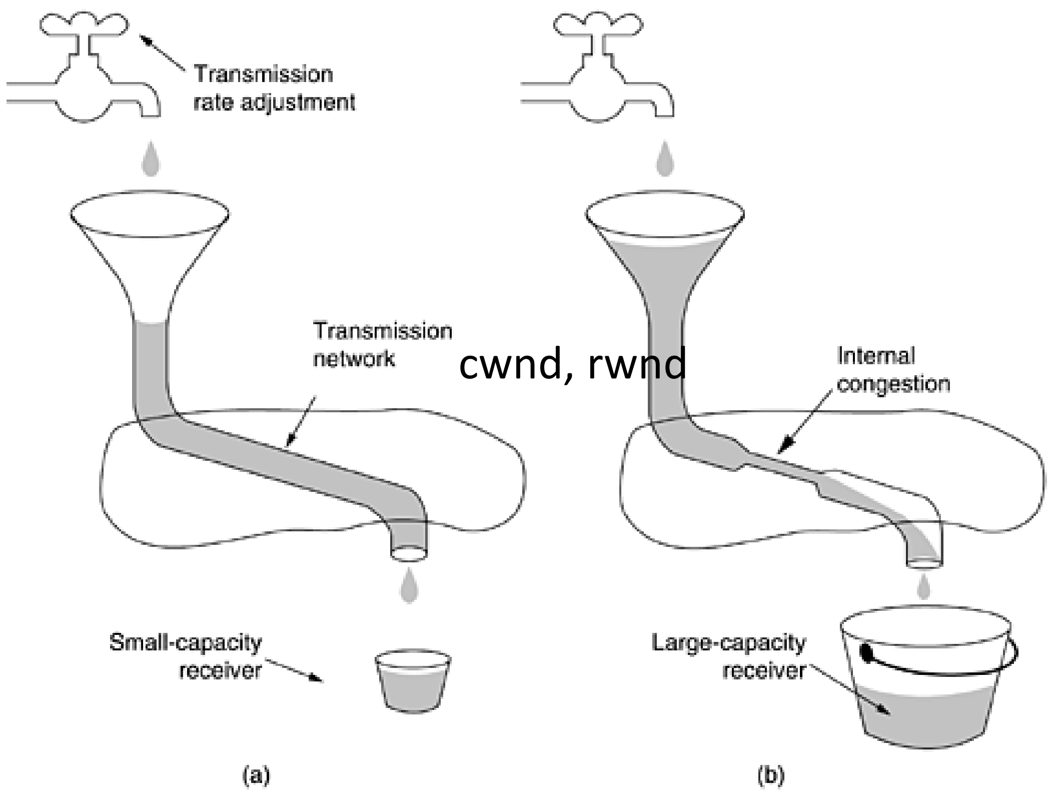


2 problems, joint solution: limit the rate of the sender!

(or ”How many windows does a TCP’s sender maintain?”)

rwnd

cwnd



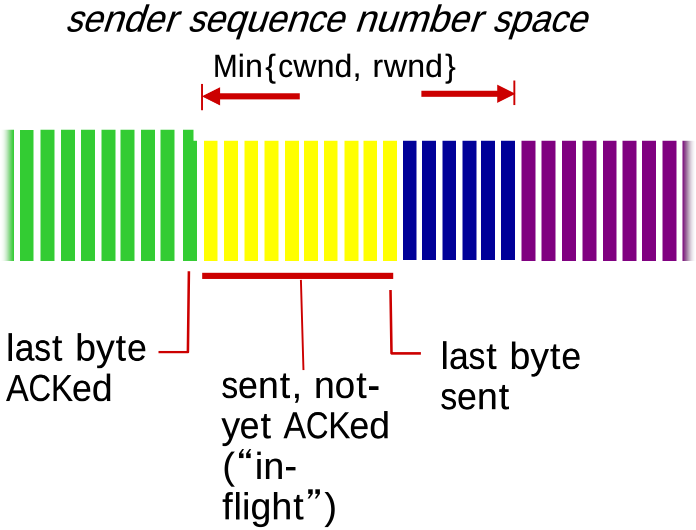
cwnd, rwnd

Fig. A. Tanenbaum
Computer Networks

Need for flow control

Need for congestion control

TCP combined flow-ctrl, congestion ctrl windows



TCP sending rate:

- send min {cwnd, rwnd} bytes, wait for ACKS, then send more

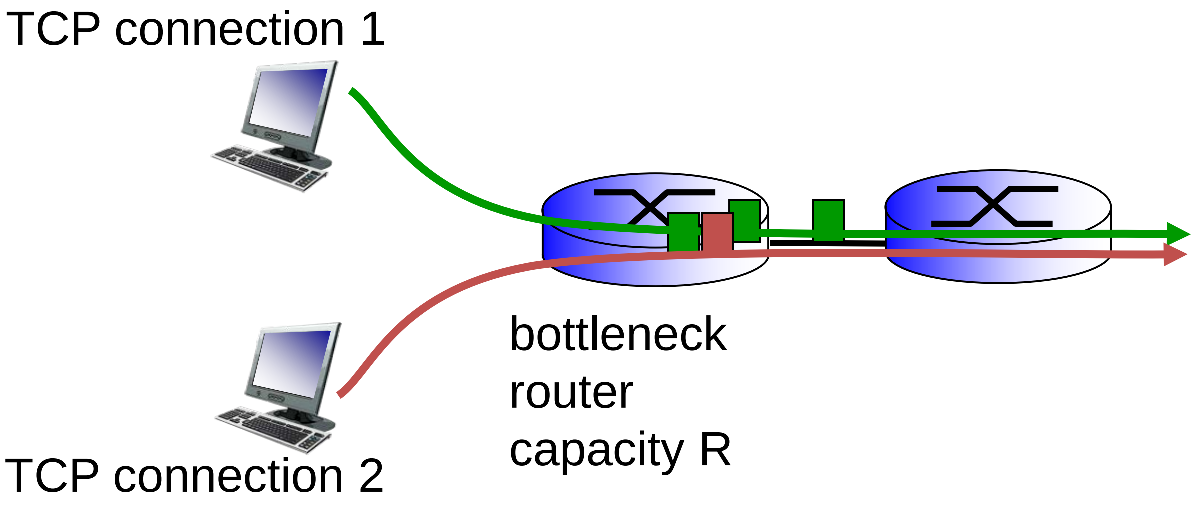
sender limits transmission:

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

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

TCP Fairness

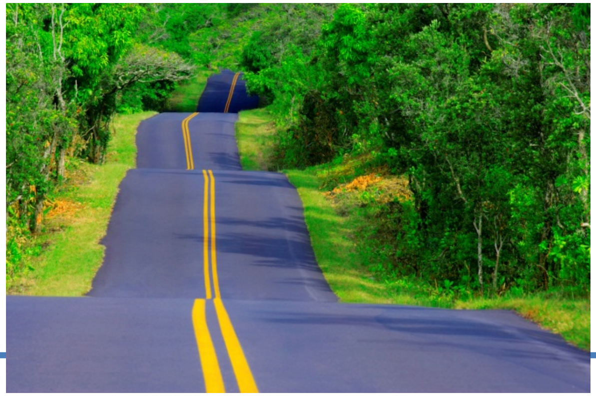
fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Q: Can a TCP implementation deviate from the Congestion-Control standard?

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

next:

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

Some more review questions on this part

-
- rd ack and not a 2nd?
- le, method for detection of
- ndefinitely?
- ent?
- t and the end of connection?
- sfer 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 extra
  - slides

Extra slides, for further study

From RFC 1122: TCP Ack

- 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.

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

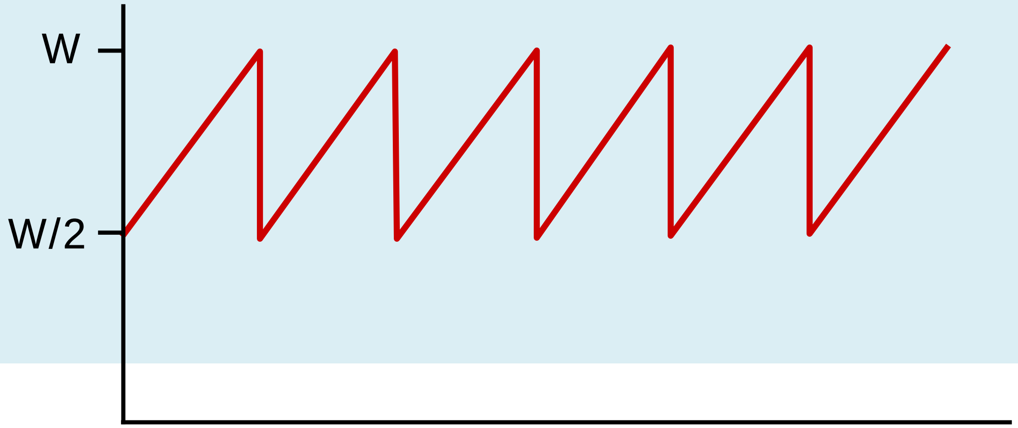
- RST packets are not supposed to carry data payload, except for an optional human-readable description of what was the reason for dropping this connection.

- Examples:
  - A TCP data segment when no session exists
  - Arrival of a segment with incorrect sequence number
  - Connection attempt to non-existing port
  - Etc.

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]:

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

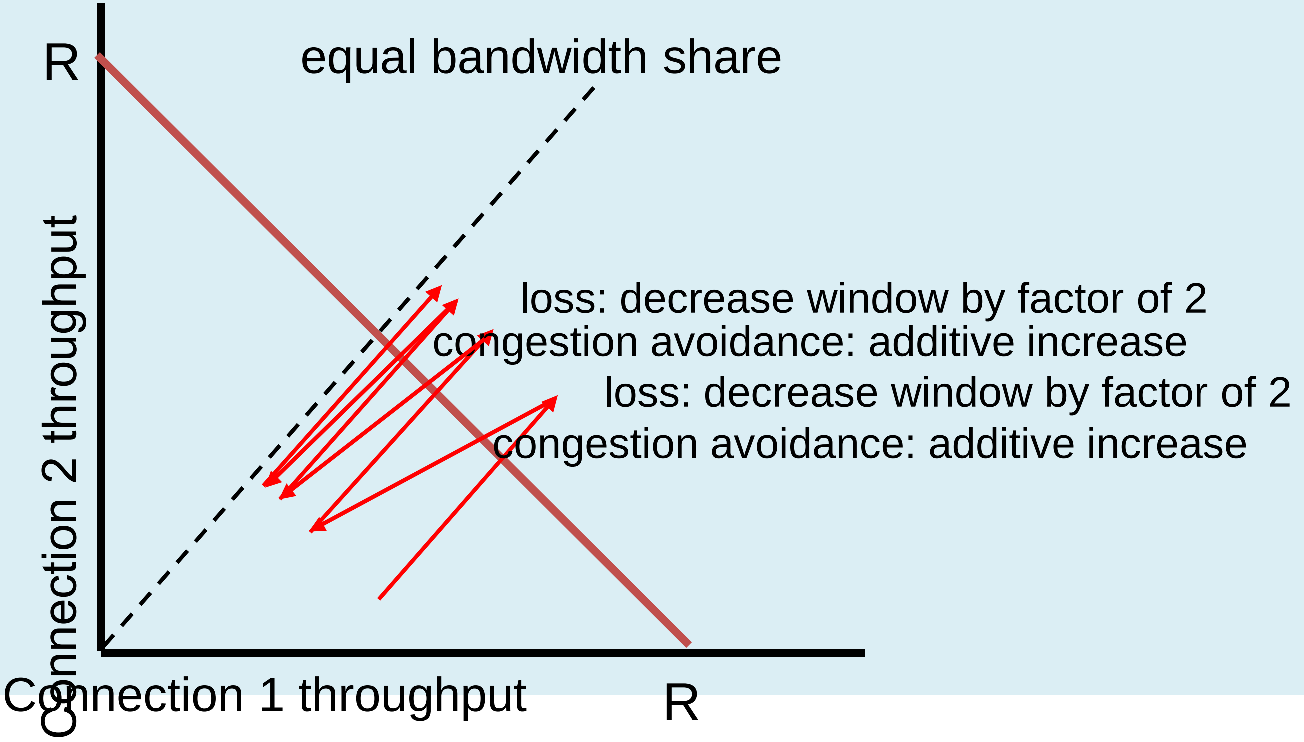
➜ to achieve 10 Gbps throughput, need a loss rate of L = $2 \cdot 10^{-10}$ *– a very small loss rate!*

- 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?

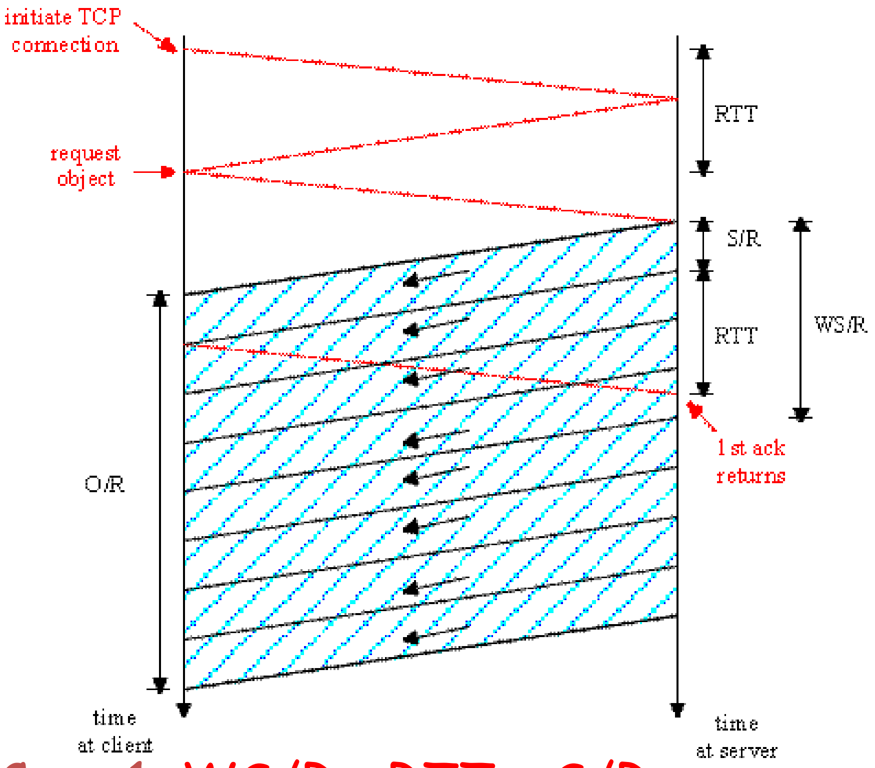
- TCP connection establishment
- data transfer delay

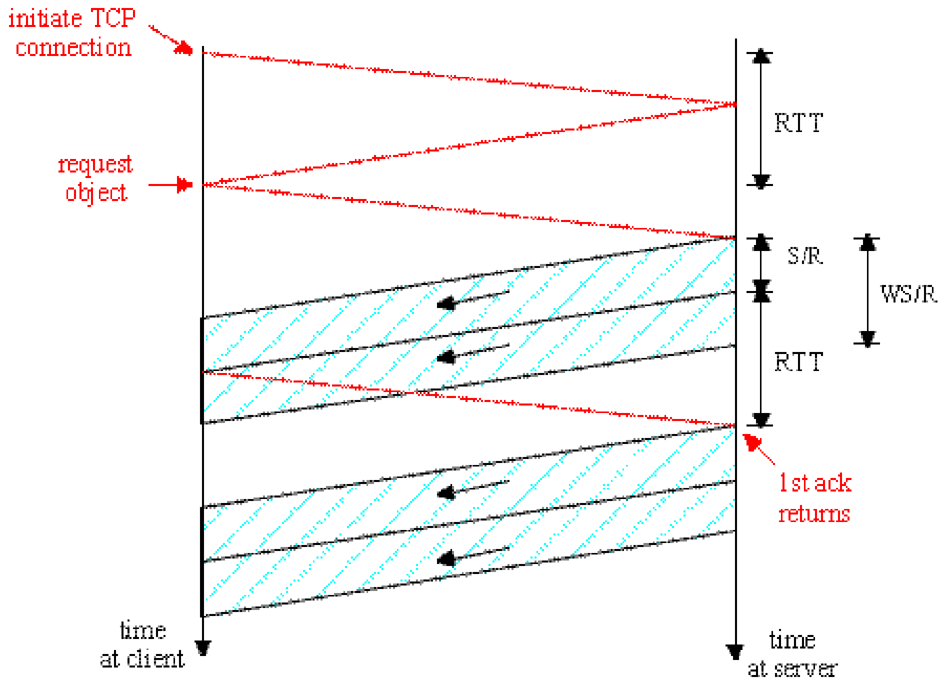
Notation, assumptions:

- 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

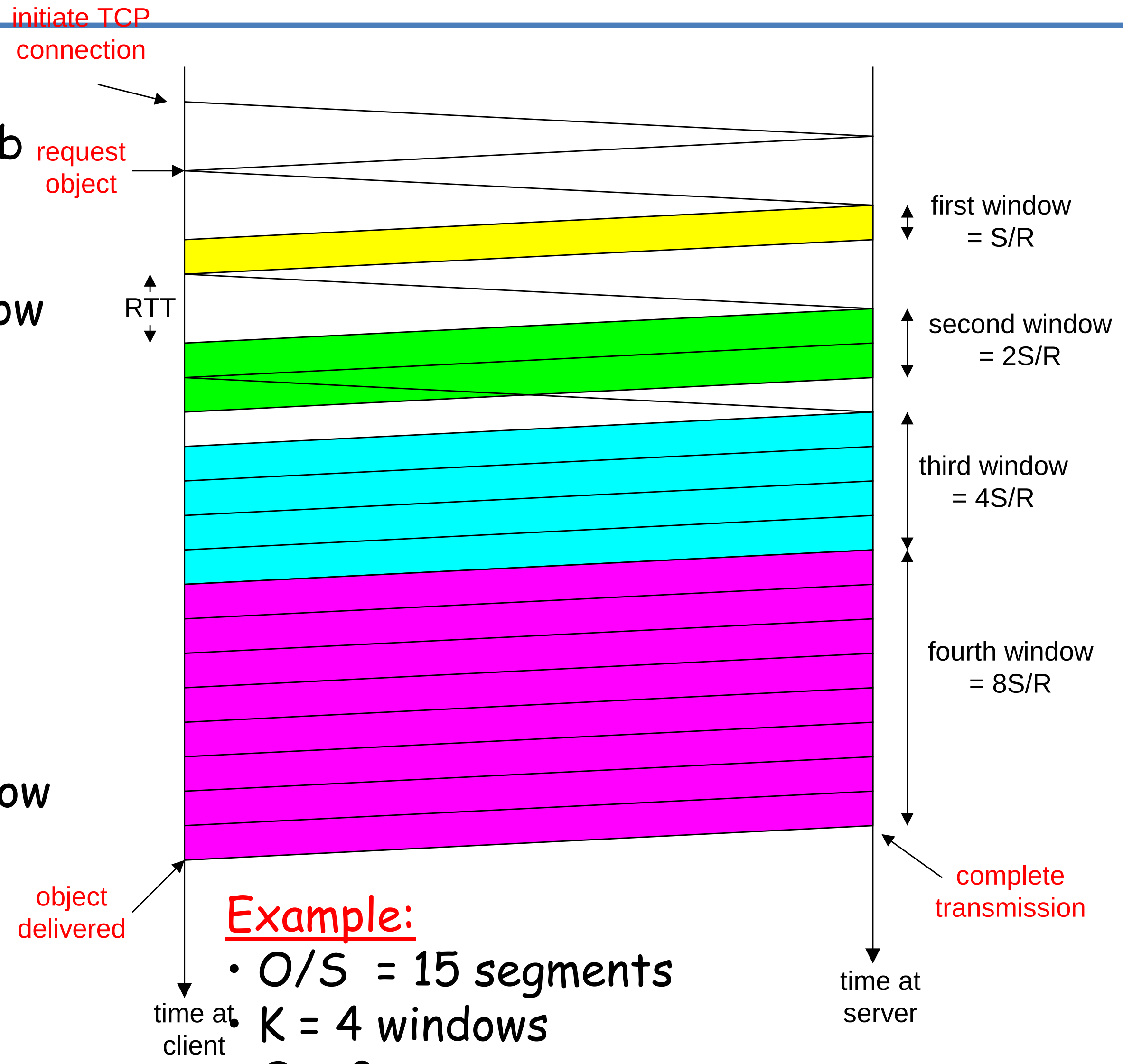
Delay components:

- 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

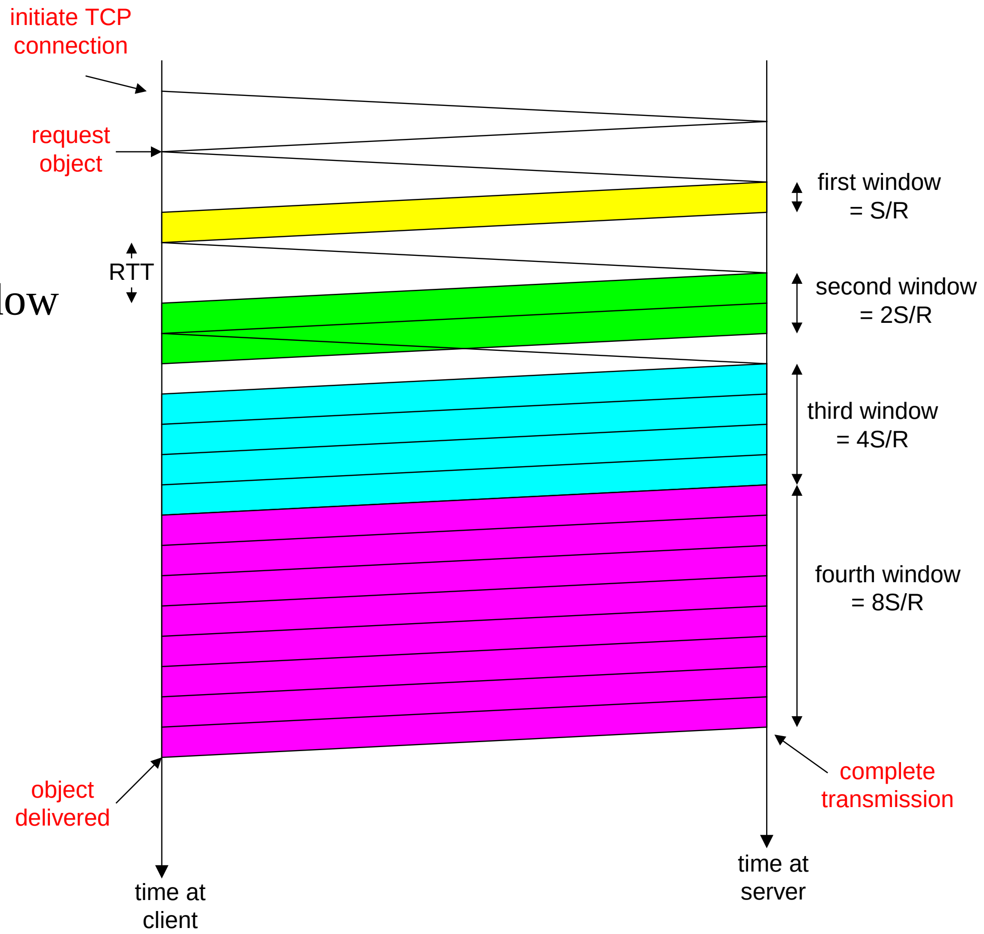
TCP Delay Modeling (slow start - cont)

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

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

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

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



TCP Delay Modeling

Recall K = number of windows that cover object

How do we calculate K ?

$$K = \min\{k : 2^0 S + 2^1 S + \dots + 2^{k-1} S \geq O\}$$

$$= \min\{k : 2^0 + 2^1 + \dots + 2^{k-1} \geq O / S\}$$

$$= \min\{k : 2^k - 1 \geq \frac{O}{S}\}$$

$$= \min\{k : k \geq \log_2(\frac{O}{S} + 1)\}$$

$$= \left\lceil \log_2(\frac{O}{S} + 1) \right\rceil$$

Calculation of Q, number of idles for infinite-size object, is similar.