- Protocol design for reliability
- Ack, seq, sliding window (pipe-line), time out, error detection (checksum)
- Stop & wait, go back N (GBN), selective repeat (SR)
- Efficiency – ratio of time used by an ideal protocol and your protocol
- Formal Representation of protocols – FSM

- ---
- We will look at Internet Transport Protocol Standard – TCP
- Reliability , Flow Control, Connection Management (Internet is connection-less), Congestion/Traffic-Management

TCP –reliability, connections

TCP: Overview RFCs: 793,1122,1323, 2018, 2581

TCP payload

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"

every byte has a seq. No. + cumulative ack

- 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 controlled:
  - sender will not overwhelm receiver

TCP segment structure

FRAME data

MAC IP TCP HTTP hdr

LLP Ethernet

segment

packet

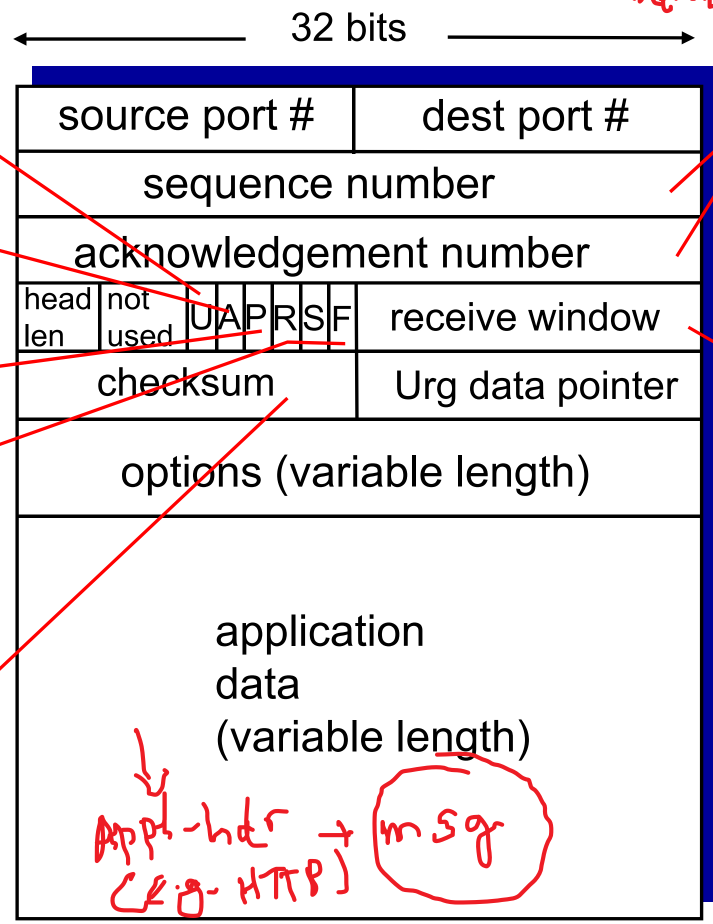
URG: urgent data (generally not used)

ACK: ACK # valid

PSH: push data now (generally not used)

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

Internet checksum (as in UDP)



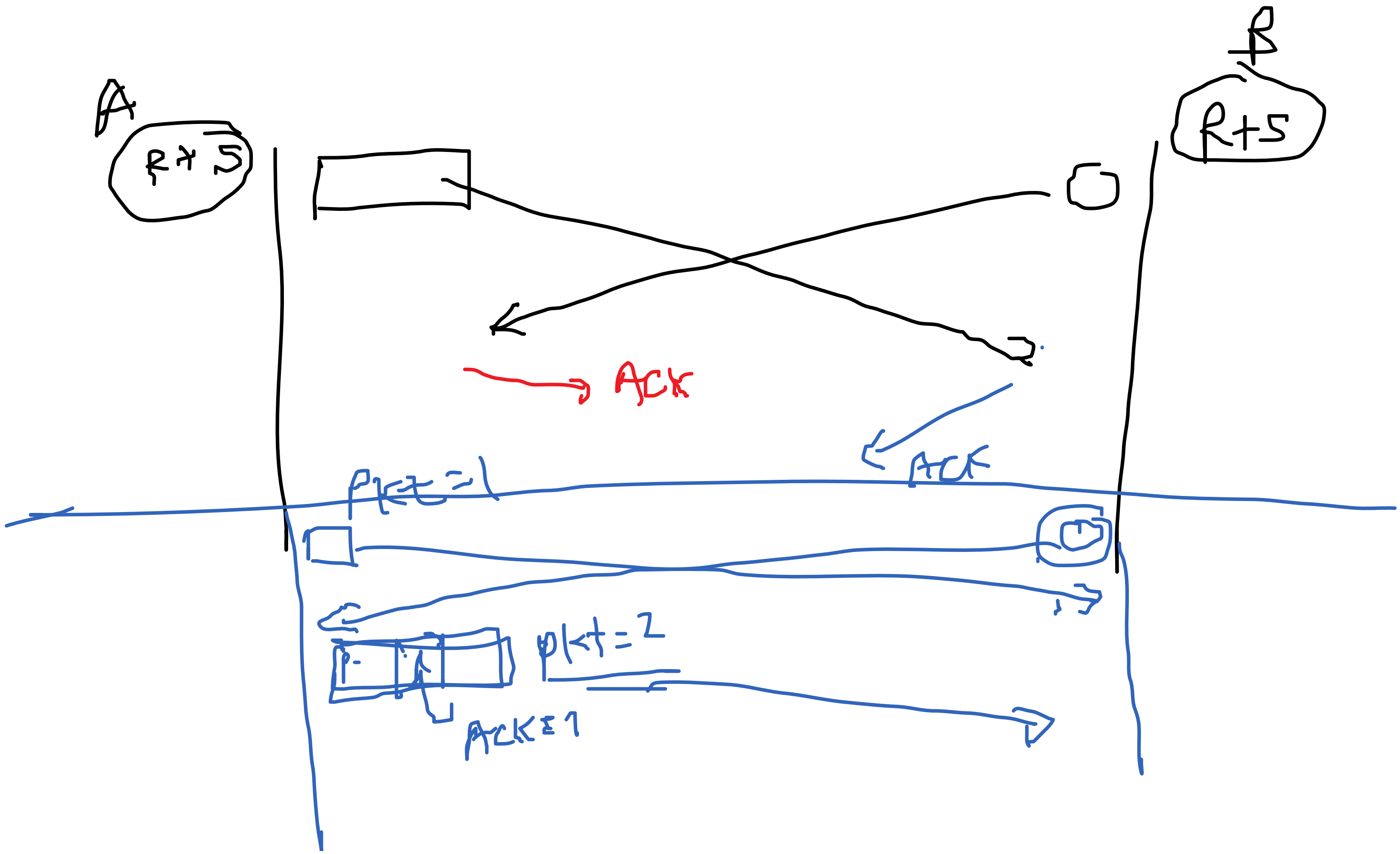
counting by bytes of data (not segments!)

bytes rcvr willing to accept

Full-duplex

node is both a sender & recv

→ PIGGY-BACKING



Piggybacking

TCP seq. numbers, ACKs

sequence numbers:

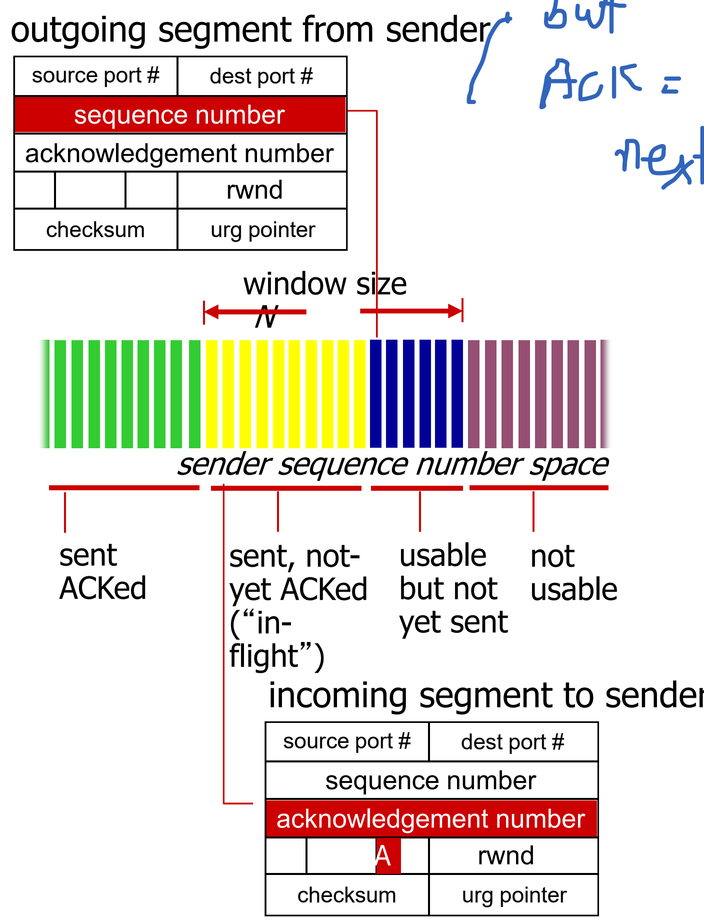
- byte stream "number" of first byte in segment's data

acknowledgements:

- seq # of next byte expected from other side
- cumulative ACK

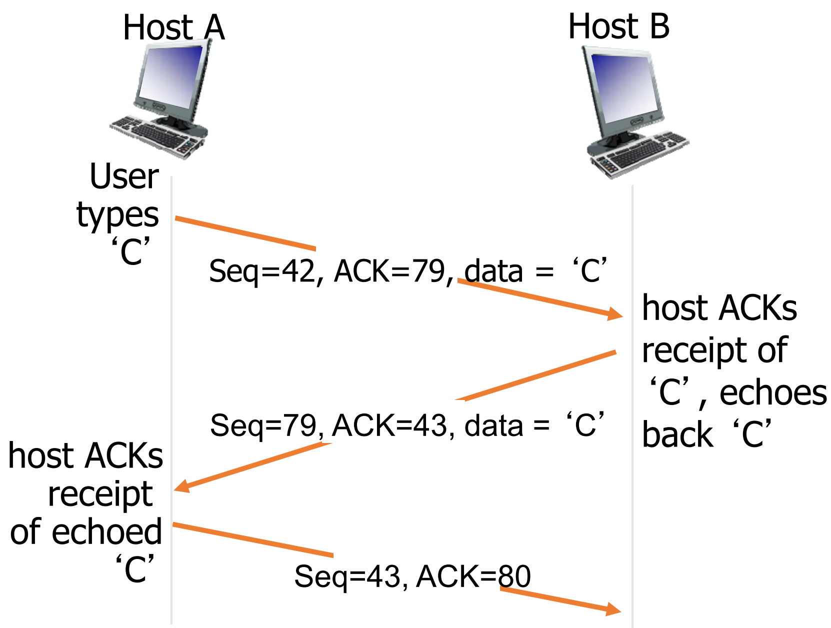
Q: how receiver handles out-of-order segments

- A: TCP spec doesn't say, - up to implementor



Ack ≠ last byte Seq No.
but ACK = Seq. no. of next expected byte.

TCP seq. numbers, ACKs



simple telnet scenario

TCP round trip time, timeout

T_0 : RTT

$\gg$ efficiency

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

Q: how to estimate RTT?

$T_0 < RTT$

Retransmission

RTT variable

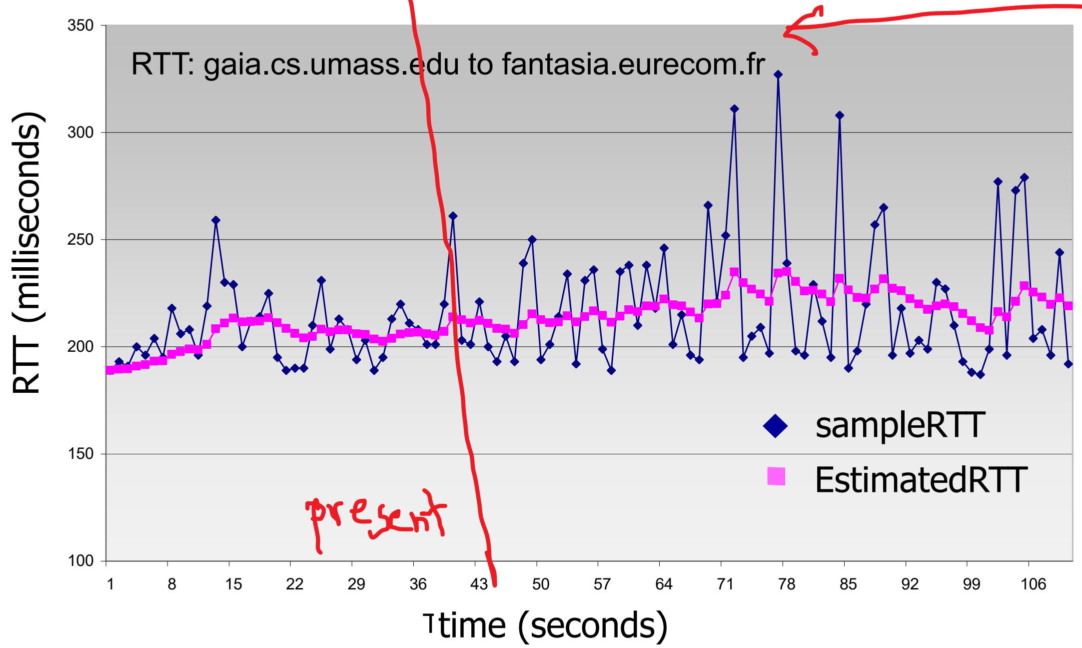
- **`SampleRTT`**: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **`SampleRTT`** will vary, want estimated RTT "smoother"
  - average several *recent* measurements, not just current **`SampleRTT`**

TCP round trip time, timeout

new old

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

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

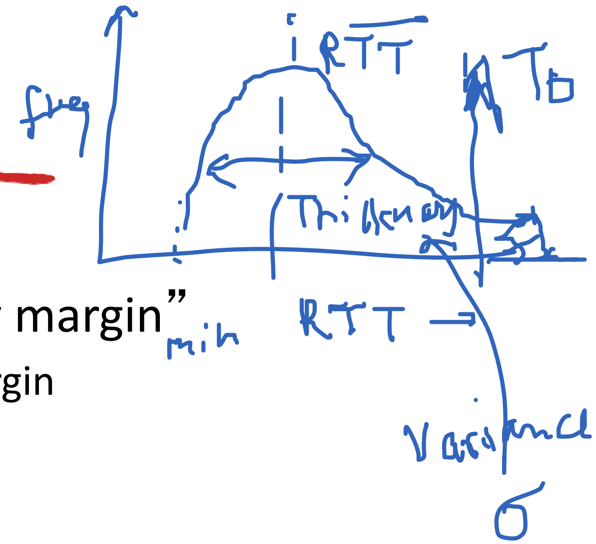


To not good

good

not good

TCP round trip time, timeout



- timeout interval: **EstimatedRTT** plus "safety margin"
  - large variation in **EstimatedRTT ->** larger safety margin
- estimate SampleRTT deviation from EstimatedRTT:

```
DevRTT = (1-β)*DevRTT +
              β*|SampleRTT-EstimatedRTT|
(typically, β = 0.25)
```

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

estimated RTT   "safety margin"

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

$$\overline{RTT}^{(n+1)} = (1-\alpha)\,\overline{RTT}^{(n)} + \alpha . RTT^{n} \qquad \underline{0<\alpha<1}$$

$$\alpha = \frac{1}{2}$$

$$\overline{RTT}^{n+1} = \frac{1}{2}\left[\frac{1}{2}\,\overline{RTT}^{(n-1)} + \frac{1}{2} RTT^{n-1}\right] + \frac{1}{2} RTT^{n}$$

$$\overline{RTT}^{(n+1)} = \frac{1}{2} RTT^{(n)} + \frac{1}{4} RTT^{(n-1)} + \frac{1}{4}\overline{RTT}^{(n-1)}$$

$$= \left[\frac{1}{2}\right] RTT^{(n)} + \frac{1}{4} RTT^{(n-1)} + \frac{1}{8} RTT^{(n-2)} + \cdots$$

TCP reliable data transfer

- TCP creates rdt service on top of IP's unreliable service
  - pipelined segments
  - cumulative acks
  - single retransmission timer
- retransmissions triggered by:
  - timeout events
  - duplicate acks

let's initially consider simplified TCP sender:
- ignore duplicate acks
- ignore flow control, congestion control

TCP sender events:

data rcvd from app:

- create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running
  - think of timer as for oldest unacked segment
  - expiration interval: **`TimeOutInterval`**

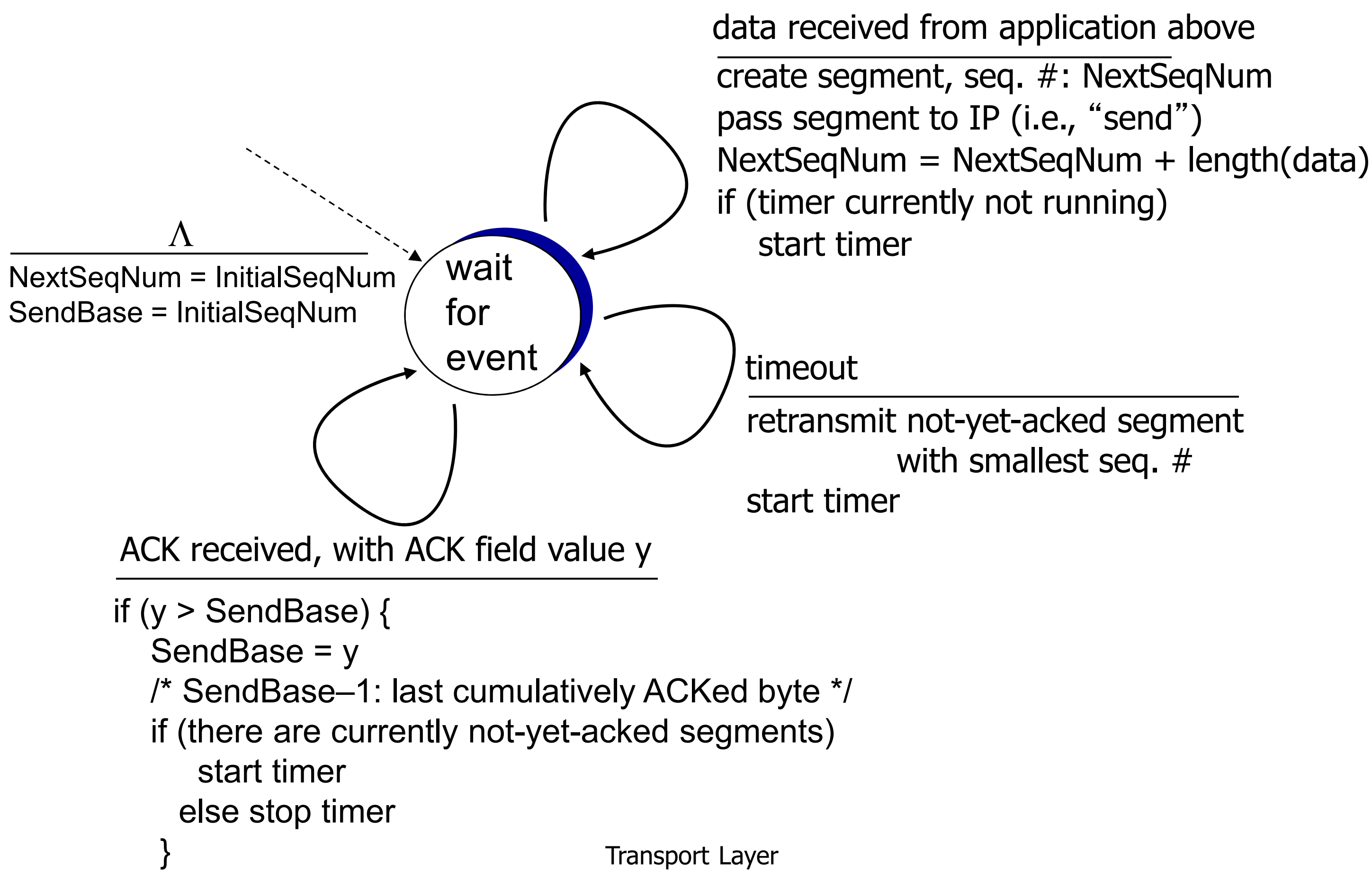
timeout:

- retransmit segment that caused timeout
- restart timer

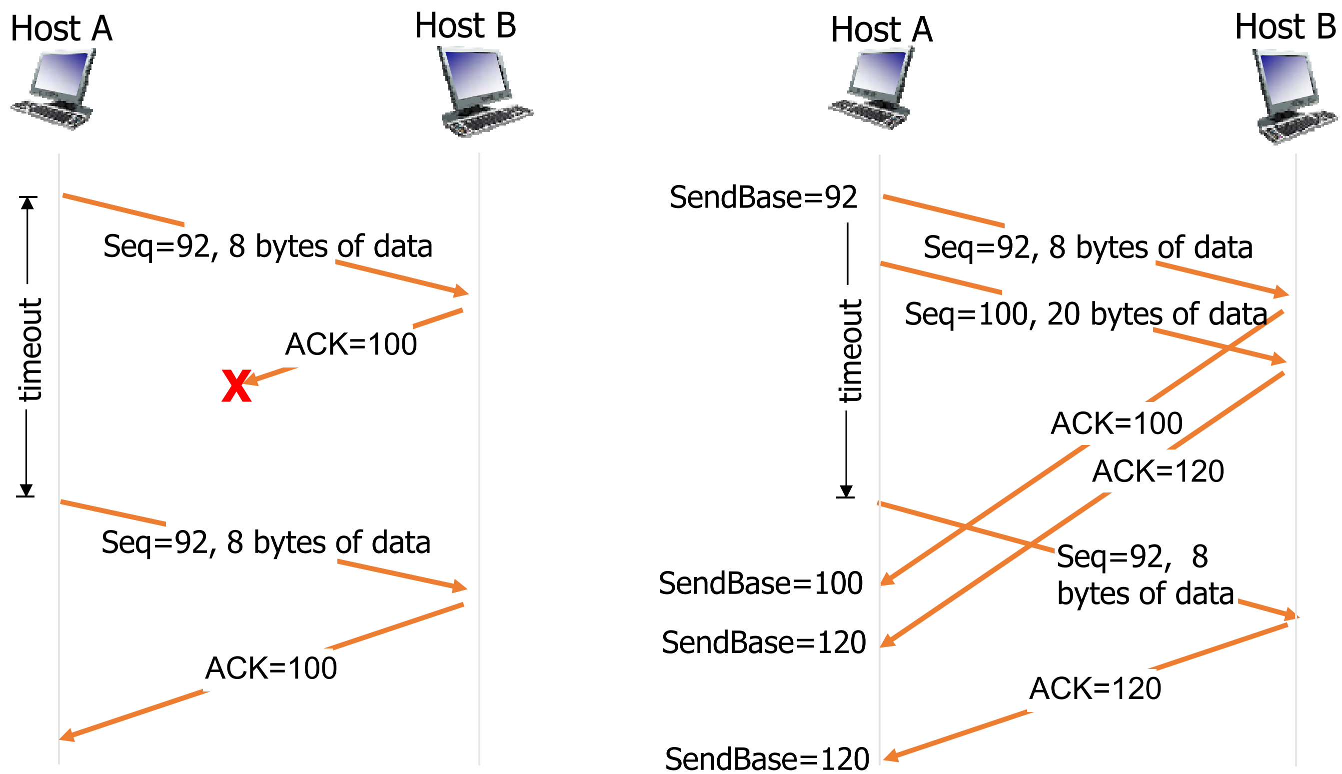
ack rcvd:

- if ack acknowledges previously unacked segments
  - update what is known to be ACKed
  - start timer if there are still unacked segments

TCP sender (simplified)



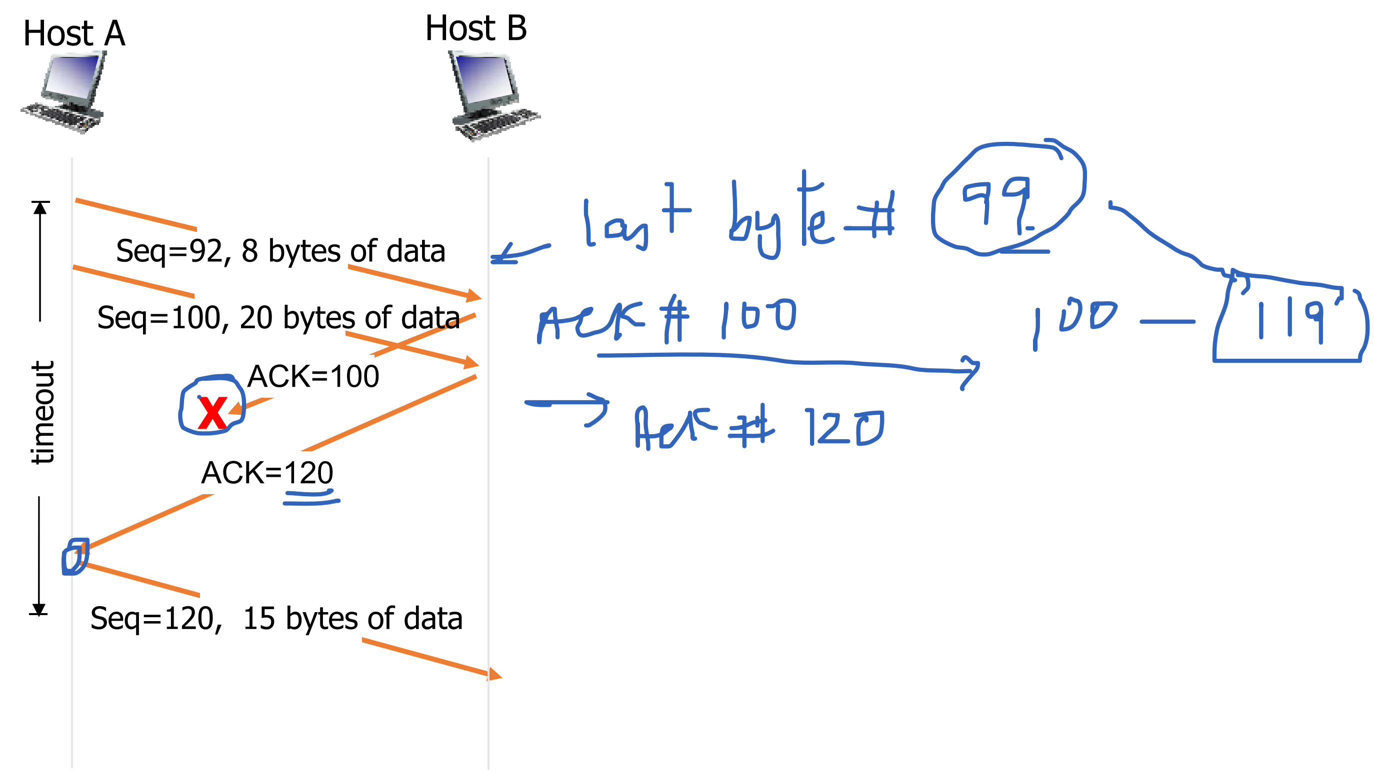
TCP: retransmission scenarios



lost ACK scenario

premature timeout

TCP: retransmission scenarios



cumulative ACK

TCP ACK generation [RFC 1122, RFC 2581]

event at receiver	*TCP receiver action*
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send single cumulative ACK, ACKing both in-order segments
arrival of out-of-order segment higher-than-expect seq. # . Gap detected	immediately send *duplicate ACK,* indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	immediate send ACK, provided that segment starts at lower end of gap

TCP fast retransmit

- time-out period often relatively long:
  - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs.

TCP fast retransmit

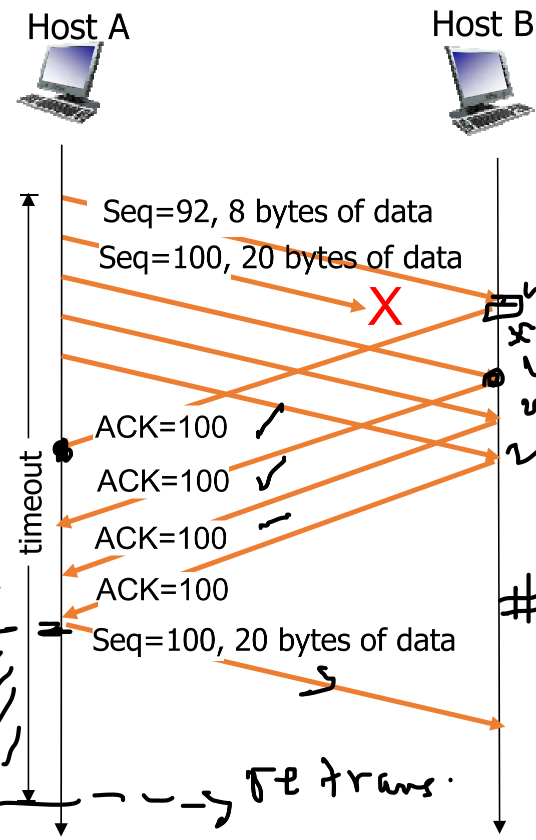
if sender receives 3 ACKs for same data ("triple duplicate ACKs"), resend unacked segment with smallest seq #

- likely that unacked segment lost, so don't wait for timeout

TCP fast retransmit

TCP - Tahoe → Vegas, . . .
Reno

"100-120"
→ upto 99 ✓
||| gap
→ ACK=100



Time saved

re trans.

pipelined

Repeat prev. Ack

gap 99 119

Repeat Ack ⟹ pmg. that a pkt is lost.

→ 3 Rept ⟹ Retransmit

fast retransmit after sender receipt of triple duplicate ACK