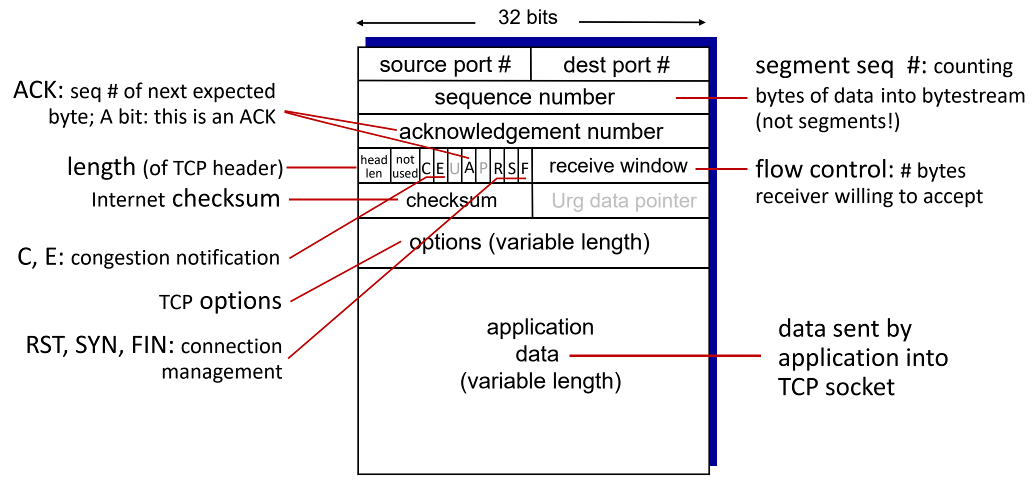
Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- Principles of congestion control
- TCP congestion control

TCP: overview RFCs: 793,1122, 2018, 5681, 7323

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte stream:*
  - no “message boundaries“
  - MSS: maximum segment size
- full duplex data:
  - bi-directional data flow in same (TCP) connection
- cumulative ACKs
- pipelining:
  - TCP congestion and flow control set window size
- connection-oriented:
  - handshaking (exchange of control messages) initializes sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP segment structure

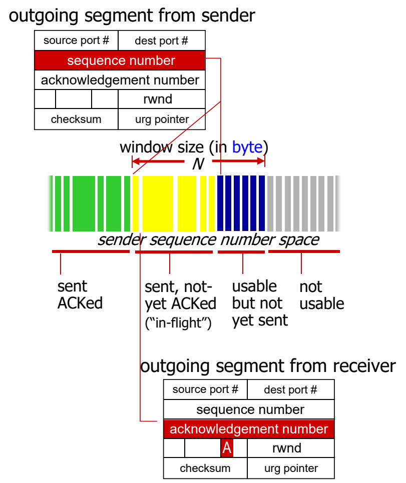

32 bits
source port #
dest port #
sequence number
acknowledgement number
head len
not used
C E U A P R S F
receive window
checksum
Urg data pointer
options (variable length)
application data (variable length)
ACK: seq # of next expected byte; A bit: this is an ACK
length (of TCP header)
Internet checksum
C, E: congestion notification
TCP options
RST, SYN, FIN: connection management
segment seq #: counting bytes of data into bytestream (not segments!)
flow control: # bytes receiver willing to accept
data sent by application into TCP socket

TCP sequence numbers and ACKs

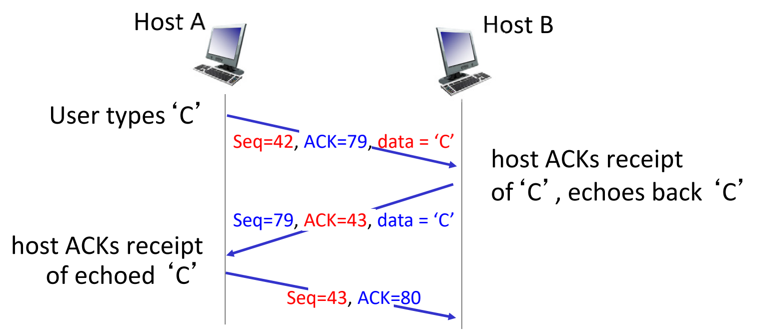
Sequence numbers:
- byte stream “number” of first byte in segment’s data

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



TCP sequence numbers and ACKs


Host A
Host B
User types ‘C’
Seq=42, ACK=79, data = ‘C’
host ACKs receipt of ‘C’, echoes back ‘C’
Seq=79, ACK=43, data = ‘C’
host ACKs receipt of echoed ‘C’
Seq=43, ACK=80

simple telnet scenario

TCP round trip time and timeout

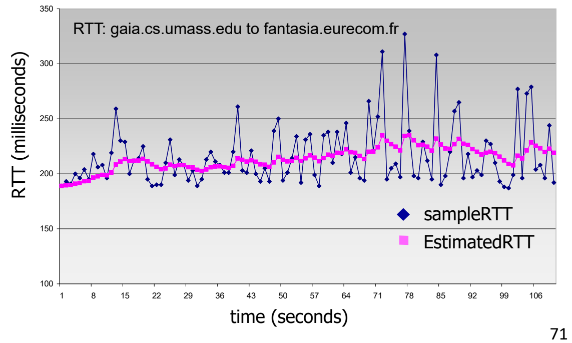
Q: how to set TCP timeout value?
- longer than RTT, but RTT varies!
- *too short:* premature timeout, unnecessary retransmissions
- *too long:* slow reaction to segment loss

Q: how to estimate RTT?
- `SampleRTT`: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- `SampleRTT` will vary and we want estimated RTT “smoother”
  - average several *recent* measurements, not just current `SampleRTT`

TCP round trip time and timeout

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

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



TCP round trip time and timeout

- timeout interval: **EstimatedRTT** plus “safety margin”
  - large variation in **EstimatedRTT**: want a larger safety margin

TimeoutInterval = EstimatedRTT + 4*DevRTT

estimated RTT “safety margin”

- **DevRTT**: EWMA of **SampleRTT** deviation from **EstimatedRTT**:

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

(typically, β = 0.25)

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

TCP Sender (simplified)

event: data received from application

- 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`**

event: timeout

- retransmit segment that caused timeout
- restart timer

event: ACK received

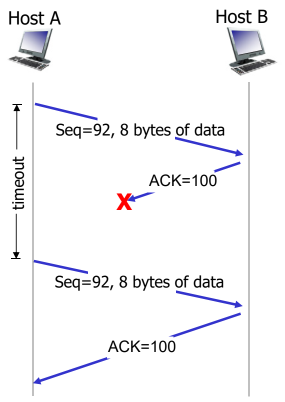
- if ACK acknowledges previously unACKed segments
  - update what is known to be ACKed
  - start timer if there are still unACKed segments

TCP Receiver: ACK generation [RFC 5681]

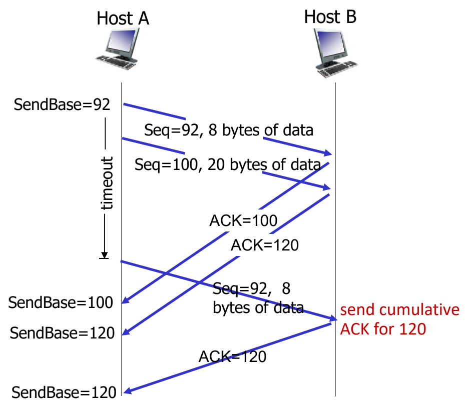
Event at receiver	*TCP receiver action*
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately 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: retransmission scenarios

lost ACK scenario

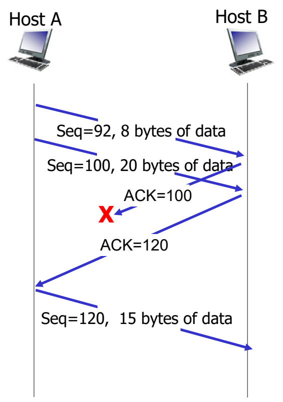


premature timeout



TCP: retransmission scenarios

cumulative ACK covers for earlier lost ACK

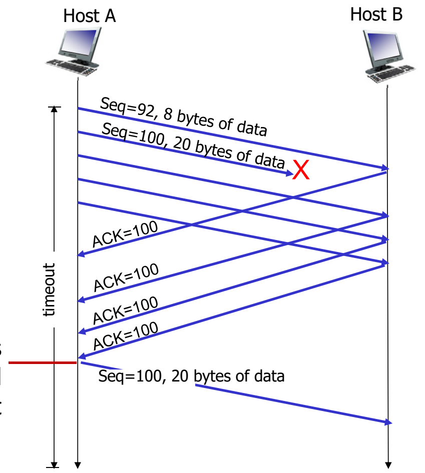


TCP fast retransmit

TCP fast retransmit

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

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



Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!