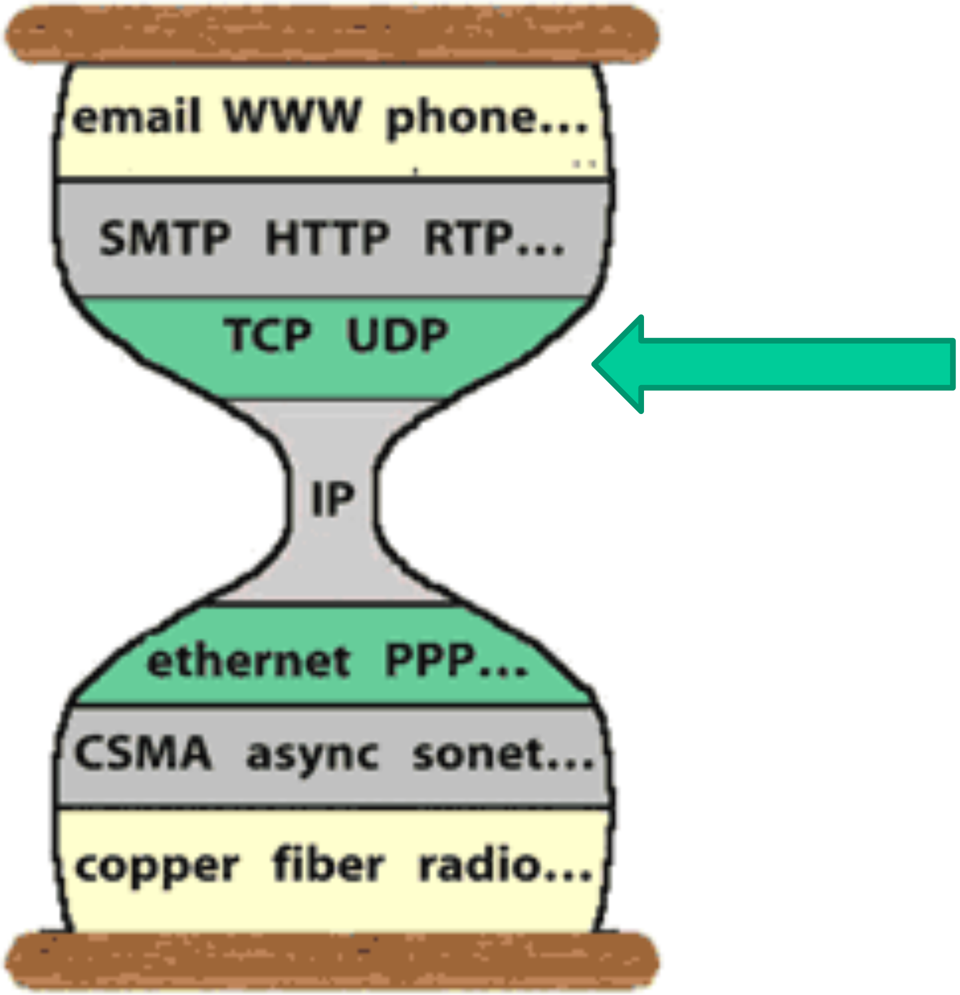
Lecture-6: TCP



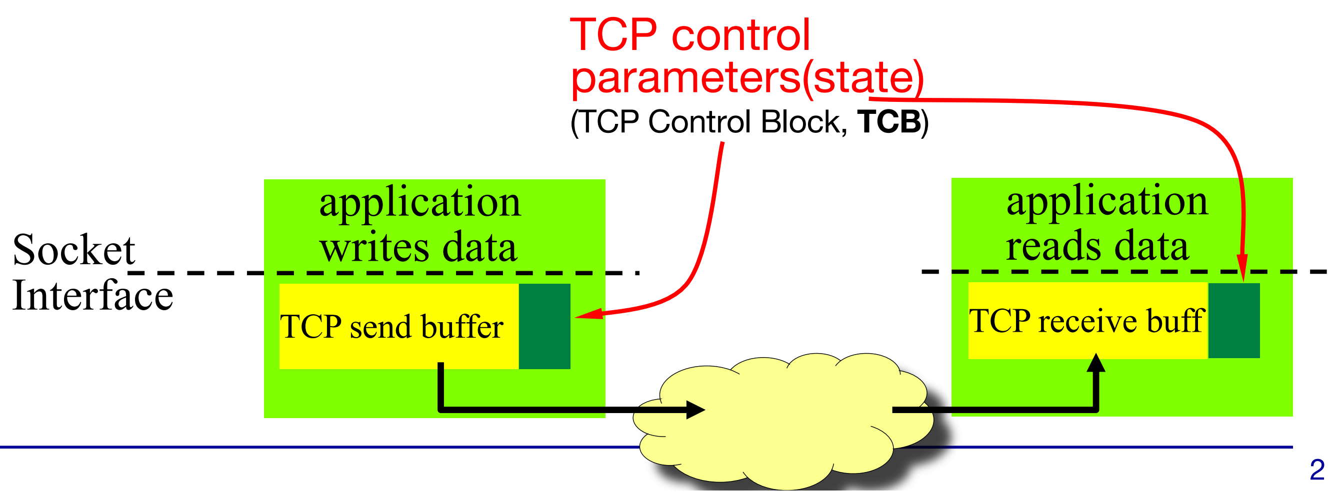
Chapter 3

3.5 TCP

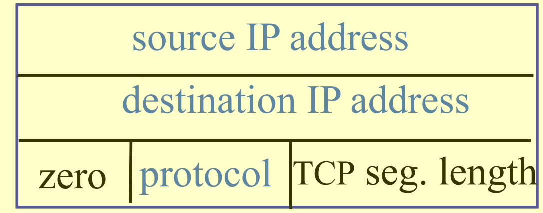
- Protocol format
- Connection management
- Flow control
- Retransmission timer

TCP function Overview

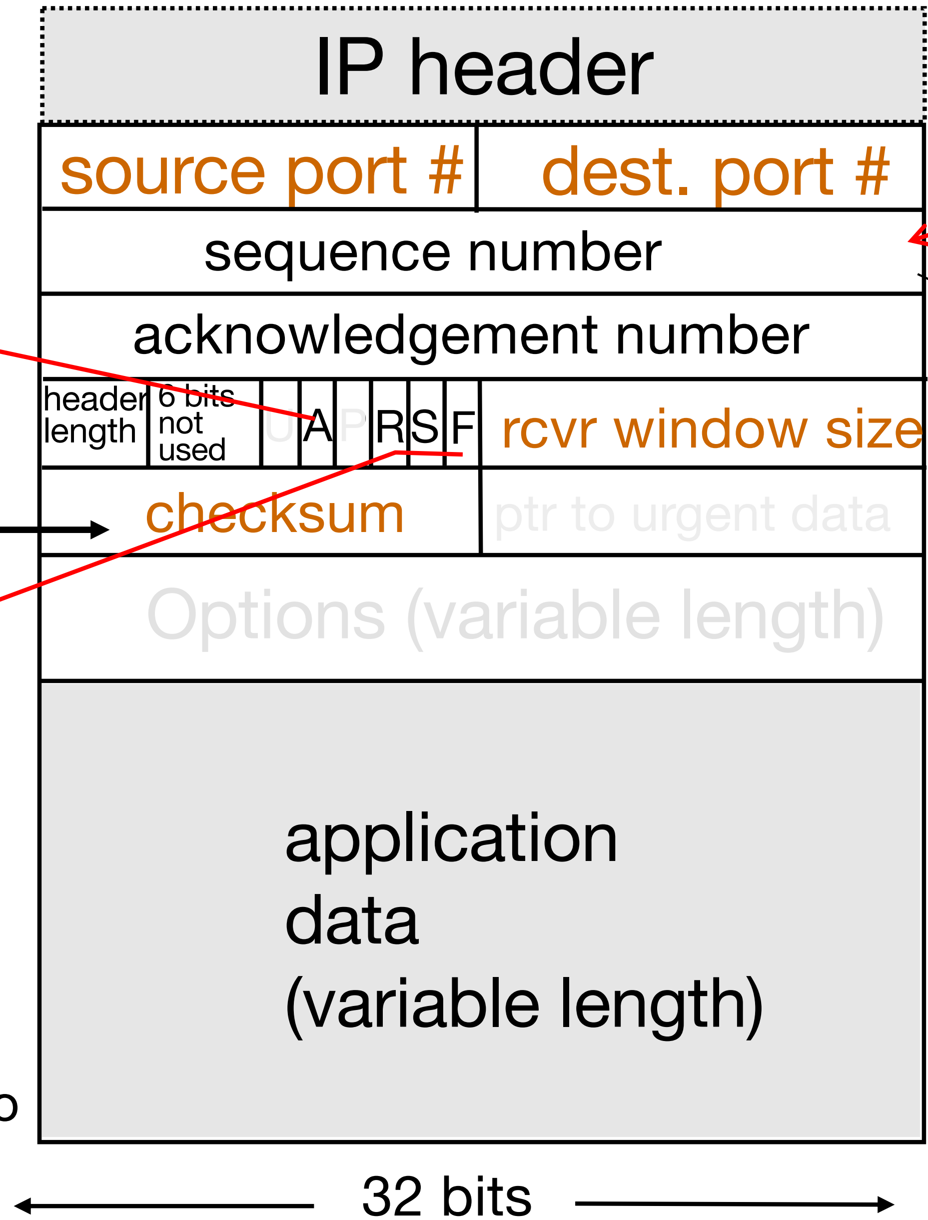
- point-to-point: creating a virtual pipe between 2 processes
- connection-oriented: set up connection first before data transmission, tear down the connection after finish
- bi-directional, reliable byte steam delivery (figure illustrates one way only)
  - no “message” boundaries
- flow controlled: prevent sender from overwhelming receiver
- congestion controlled: mitigating traffic overload *inside the network*



TCP segment format



TCP pseudo header



counting the number of bytes

Seq# of the first byte in the payload

ACK flag: ACK# field valid

Checksum is computed over TCP segment plus pseudo header

SYN, FIN, Reset: connection management flags (Setup, Finish, Reset)

TCP header has no info for congestion control

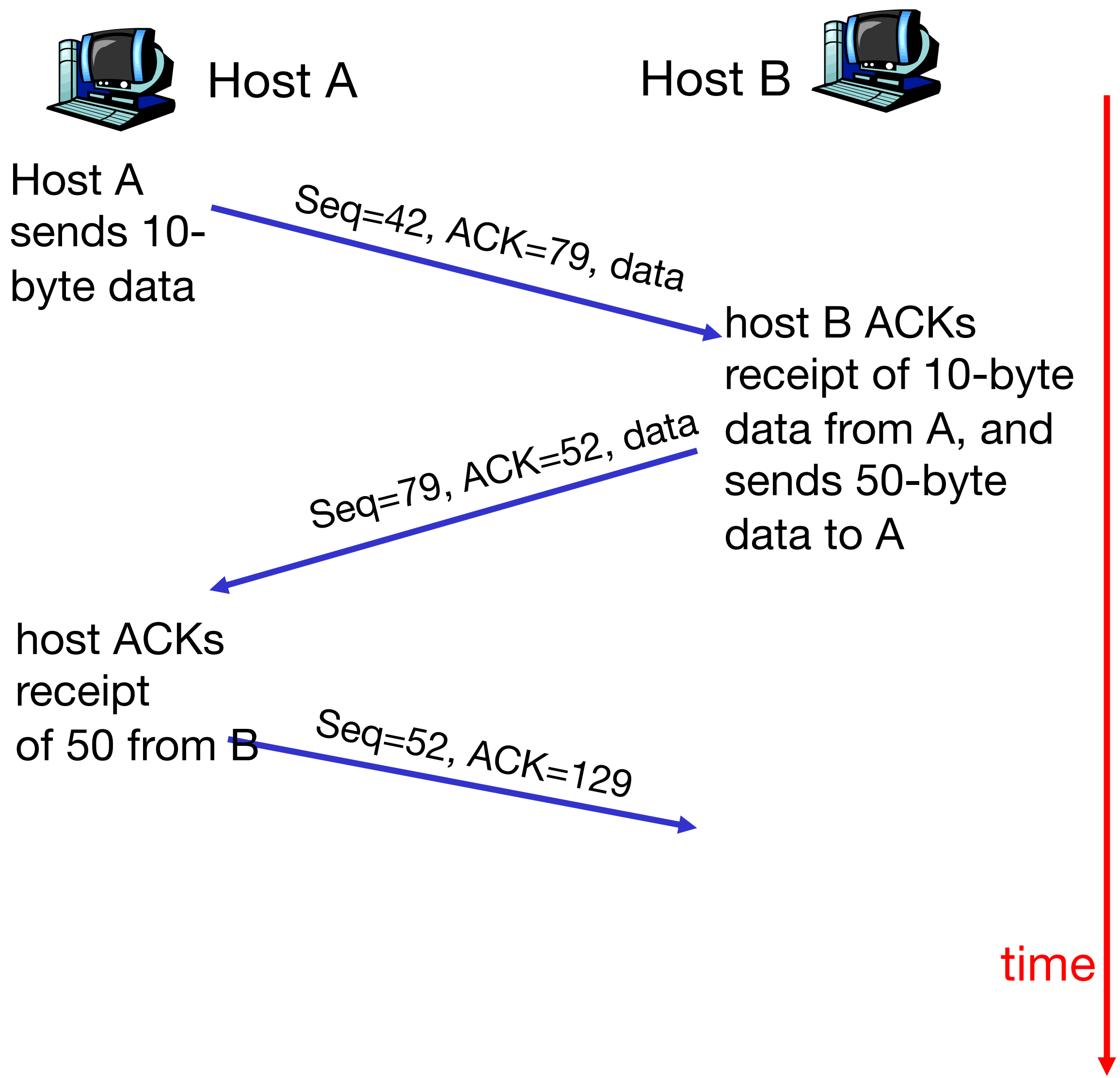
TCP's seq. #s and ACK #s

Lets first assume that a TCP connection between A and B is already setup:

Seq. #: the seq# of the first byte in this segment's data

ACK #: the seq# of next byte expected from the other end

- TCP uses cumulative ACK



TCP Connection Management

- Set up connection before starting data transmission
  - Each of the 2 ends reliably informs the other its initial data byte sequence number value
- Close connection after finishing data transmission
  - Each of the 2 ends reliably informs the other its final data byte sequence number value
- Abort connection
  - When receiving a RST segment
  - When a node may send a RST segment
    - receives a TCP segment of unknown connection
    - TCP retransmission count hits the upper-bound
    - need to reject a new connection request or close an existing TCP connection, due to resource limitation

TCP Connection Setup

Initialize TCP connection variables to get ready before sending data

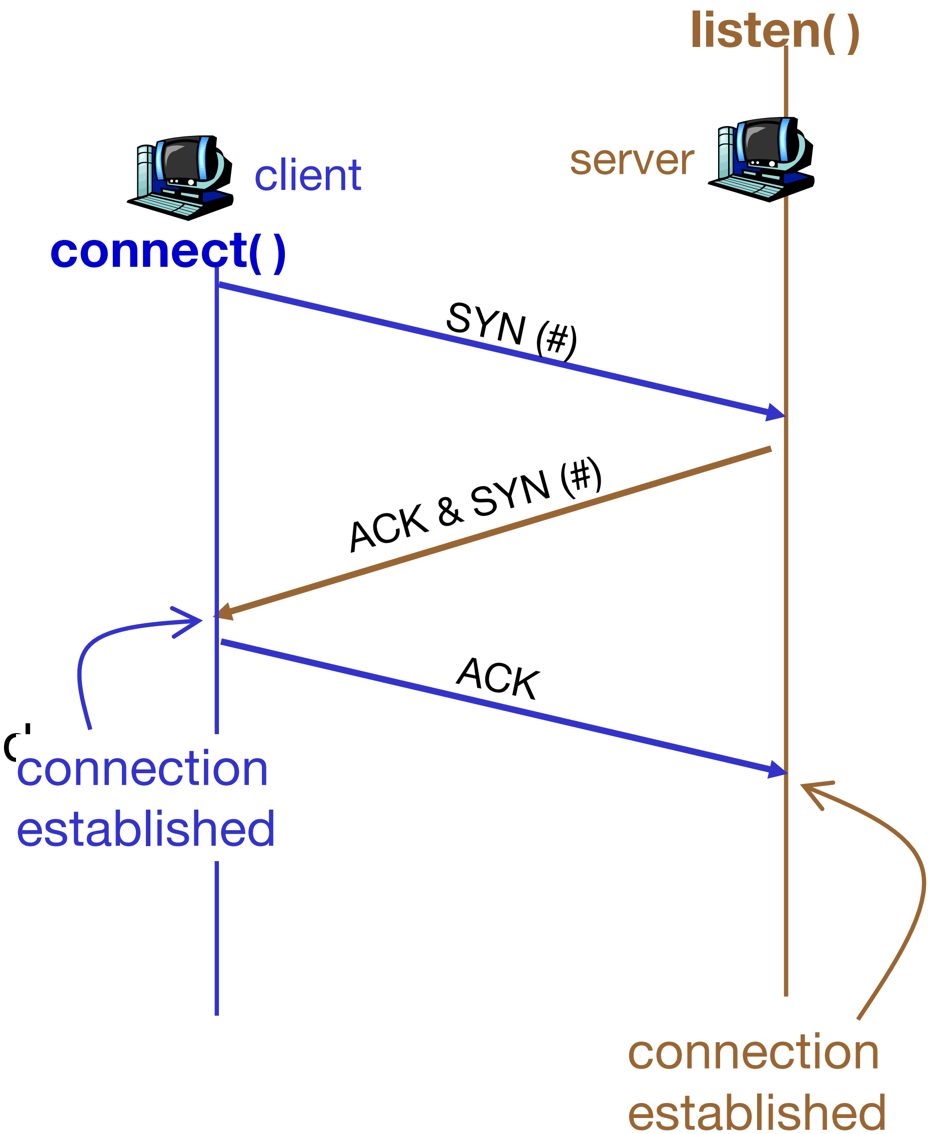
- Initial seq. # used in each direction
- Buffer size (rcvWindow)

3-way handshake in setting up a connection

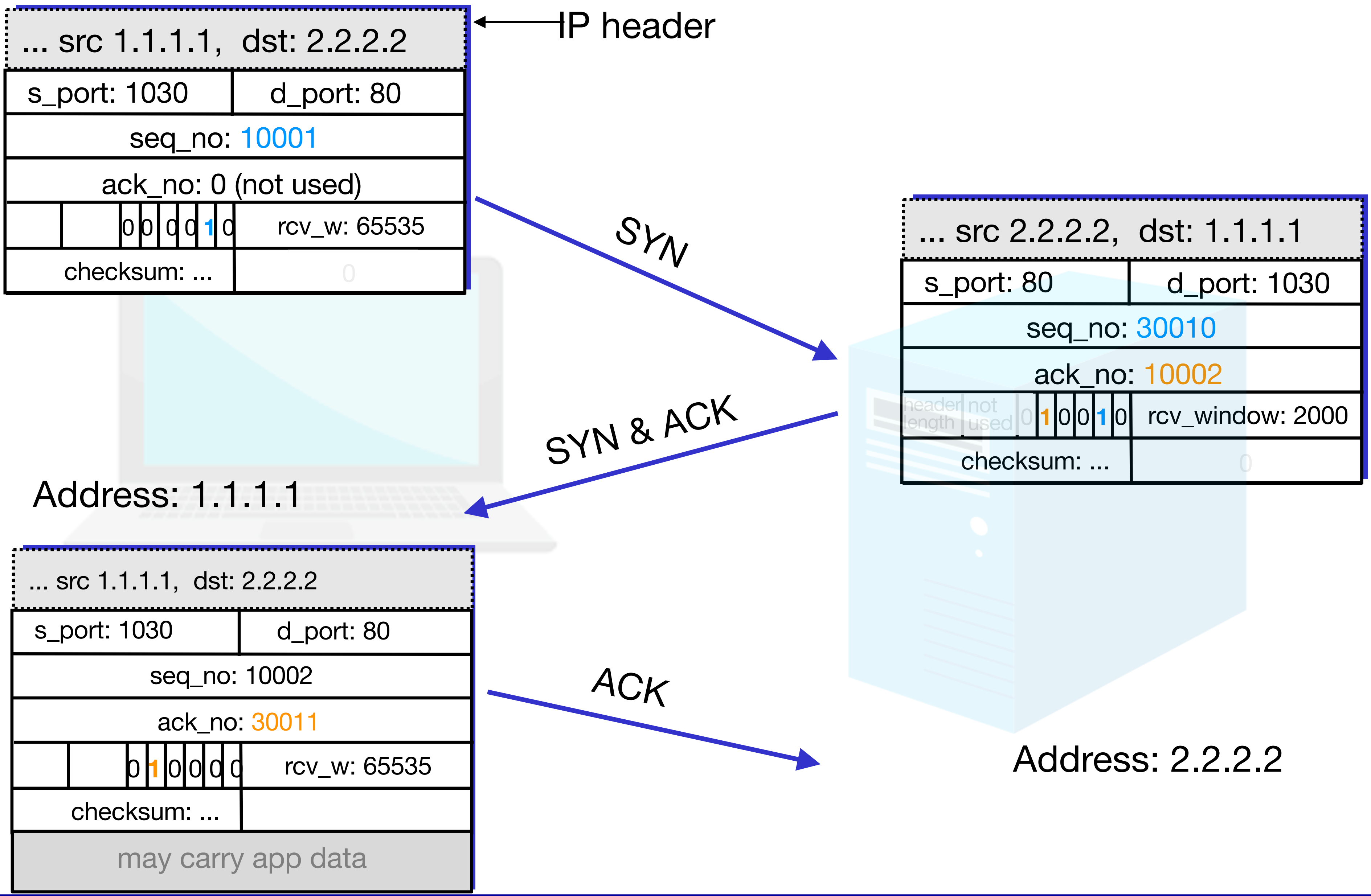
1: client host sends TCP SYN segment to server
  - SYN flag sets to 1
  - specifies client's initial seq #
    - a random number
  - does *not* carry data

2: server receives SYN, replies with **ACK** and **SYN** control segment
  - SYN and ACK flags set to 1
    - ACK received seq#
  - Specifies its own initial seq #
    - also selected randomly

3: client host sends ACK
  - ACK flag sets to 1
    - ACK received seq#
  - May carry data



A TCP connection setup example



TCP Connection Close

Either end can initiate the close of ***its end*** of the connection at any time

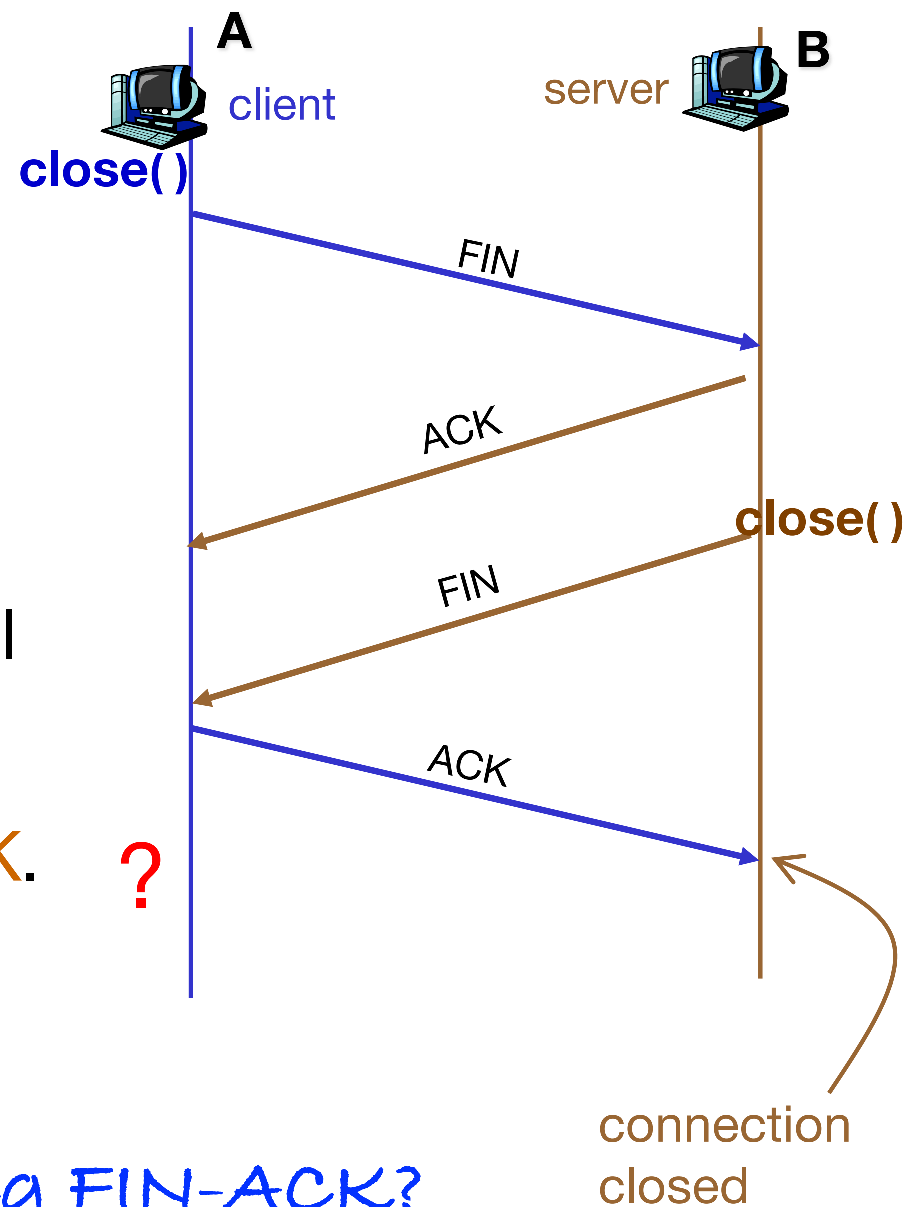
1: A sends TCP FIN control segment to the other
- FIN flag sets to 1
- This segment must not carry data

2: the other end (B) receives FIN segment, replies with ACK
- regular ACK, ACK A's FIN

3: later when B finishes sending all its data and ready to close, it sends FIN segment

4: A receives FIN, replies with ACK.

5: B receives FIN-ACK, closes connection



what should A do after sending FIN-ACK?

TCP Connection Close

Either end can initiate the close of ***its end*** of the connection at any time

1: A sends TCP FIN control segment to the other
- FIN flag sets to 1
- This segment must not carry data

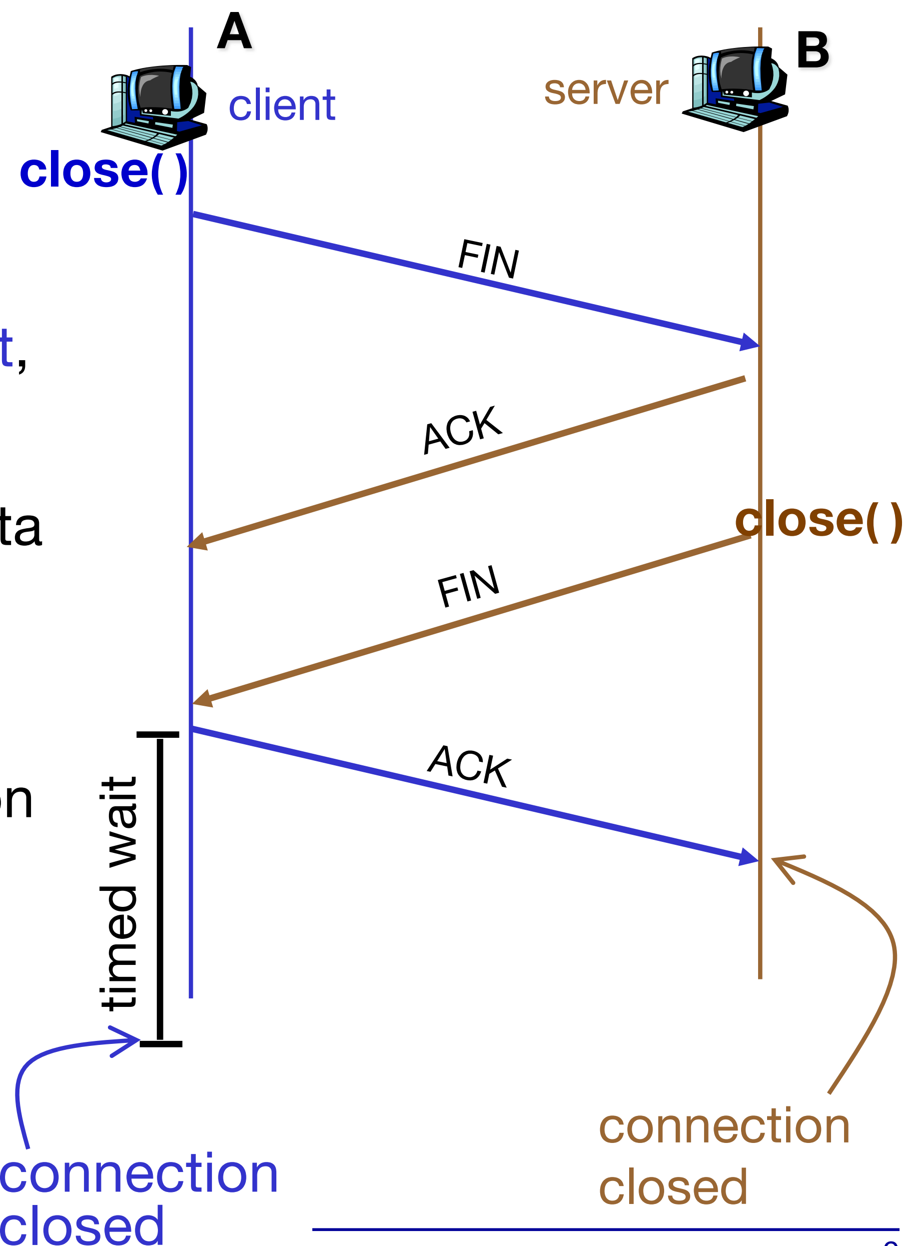
2: the other end (B) receives FIN segment, replies with ACK
- regular ACK, ACK A's FIN

3: later when B finishes sending all its data and ready to close, it sends FIN segment

4: A receives FIN, replies with ACK.

5: B receives FIN-ACK, closes connection

6: A closes the connection after waiting for "long enough" time w/o receiving retransmitted FIN
- Long enough = 2 x Max. Seg. Lifetime
  Max. Seg. Lifetime = 2 minutes

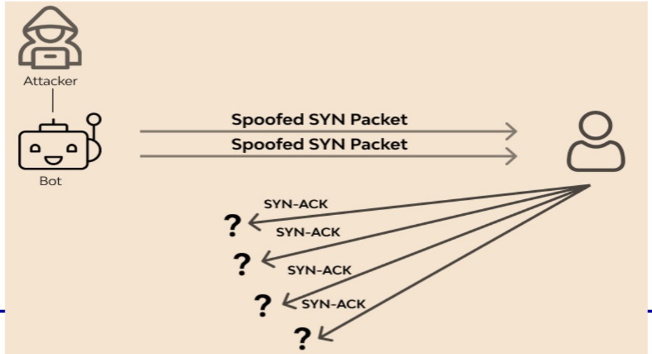


When to send "connection reset"

- Upon a TCP connection setup request: the system sets up a TCP Control Block (TCB)
  - Identified by: source+dest. addresses, source+dest. ports
  - Connection state includes info such as
    - Receiver flow control window size
    - Seq# of data
      - oldest sent but unacked
      - Latest sent, unacked
    - segments that arrived out of order
    - etc

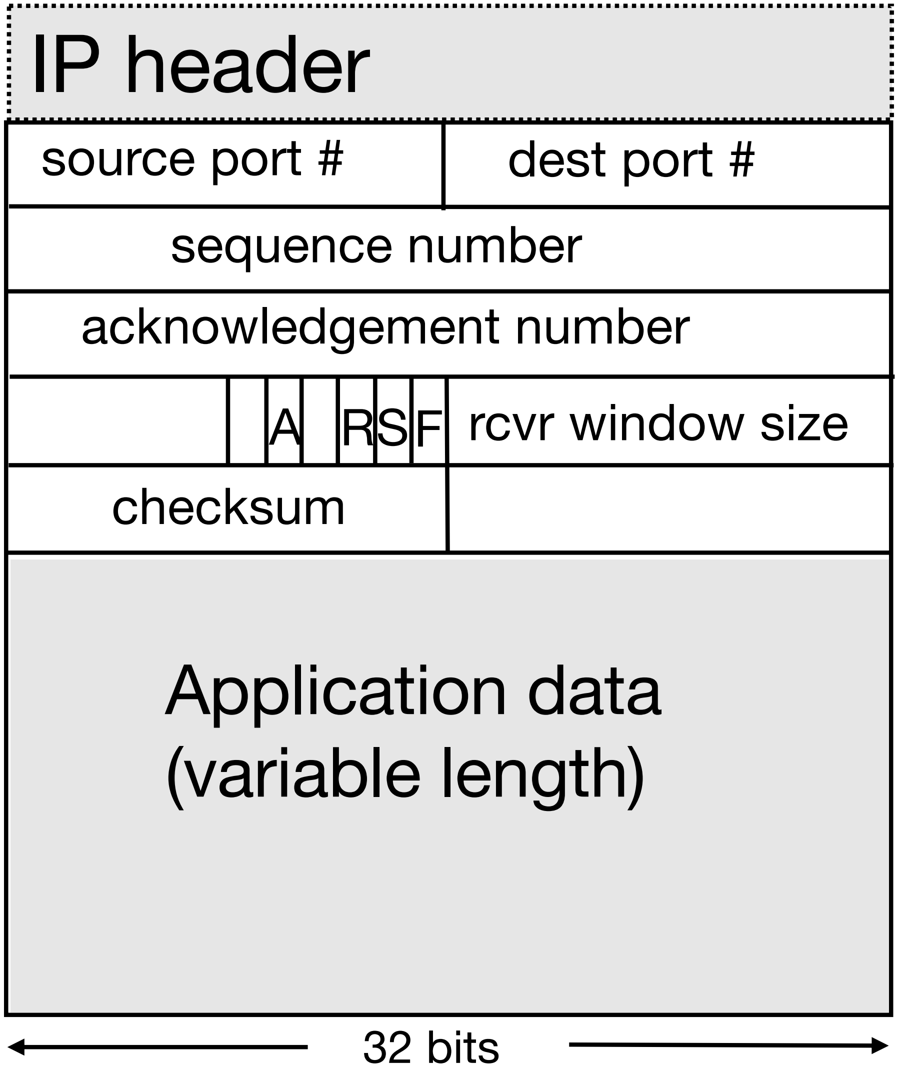
- If TCP receives a segment (other than SYN) it *cannot* find corresponding TCB: reply with RST
  - Receiver of RST aborts the connection, all data on the connection considered lost

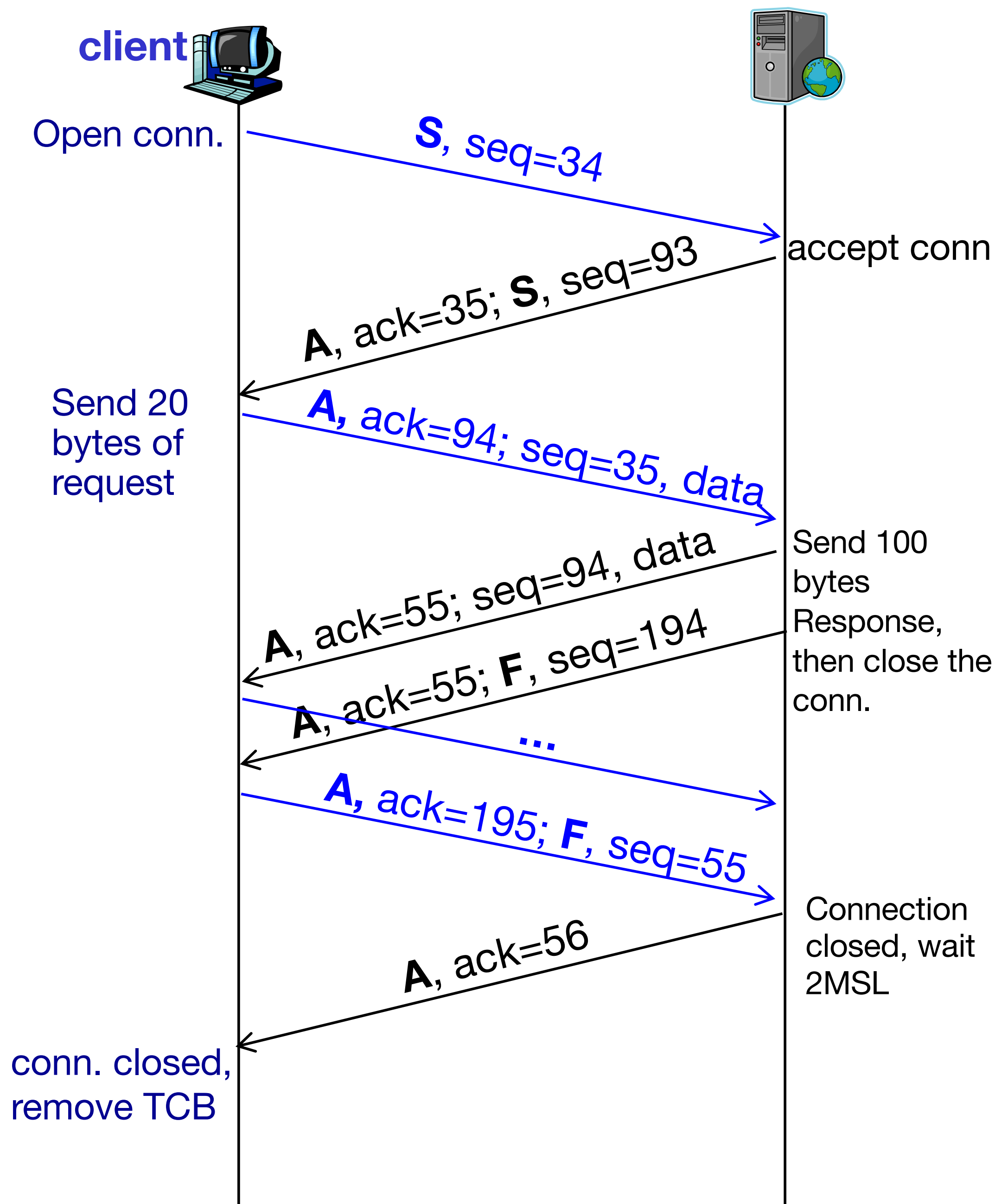
This can happen
- Due to bit errors
- By attacks:



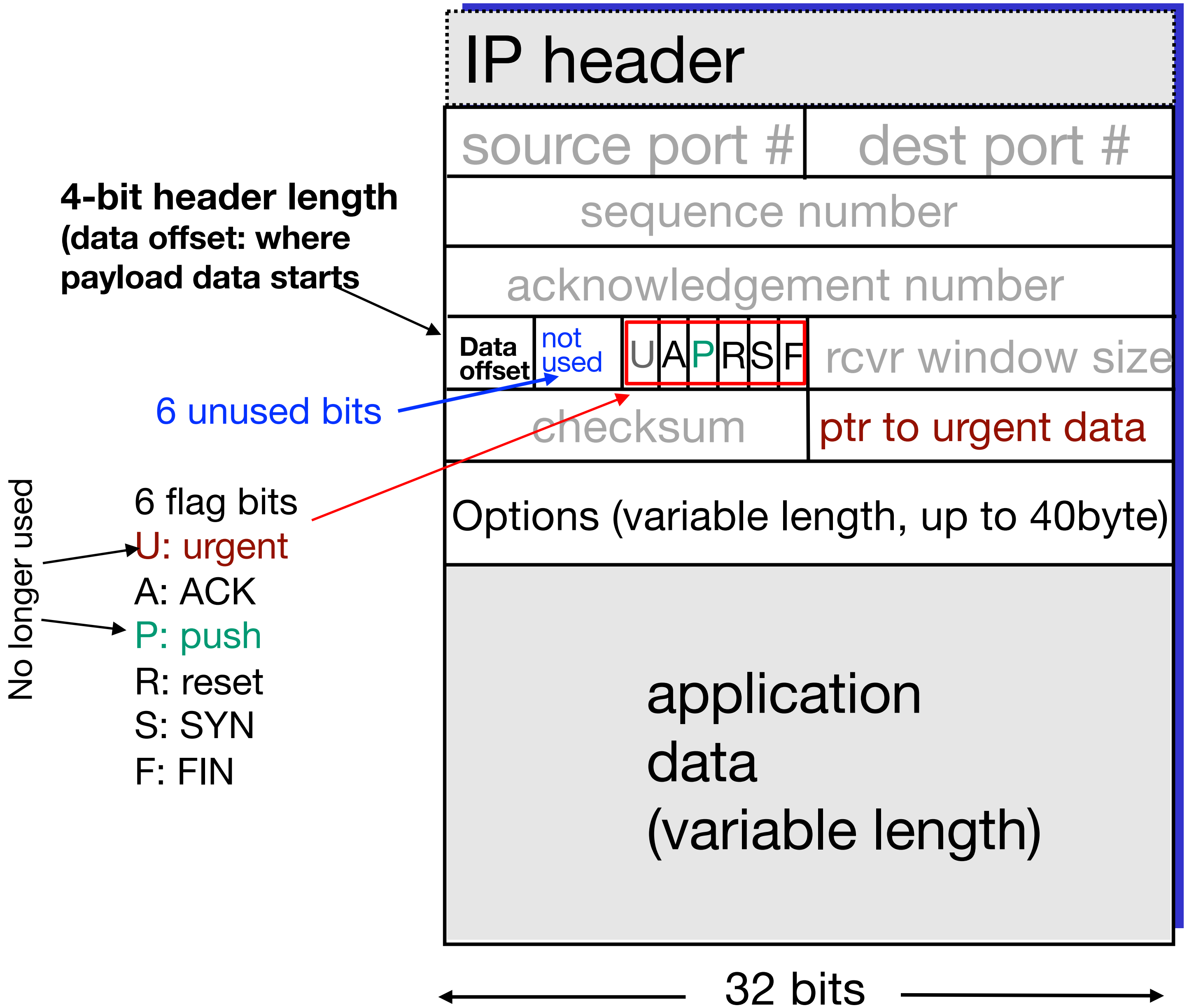
An HTTP 1.0 connection example

important





TCP segment format: the remaining parts



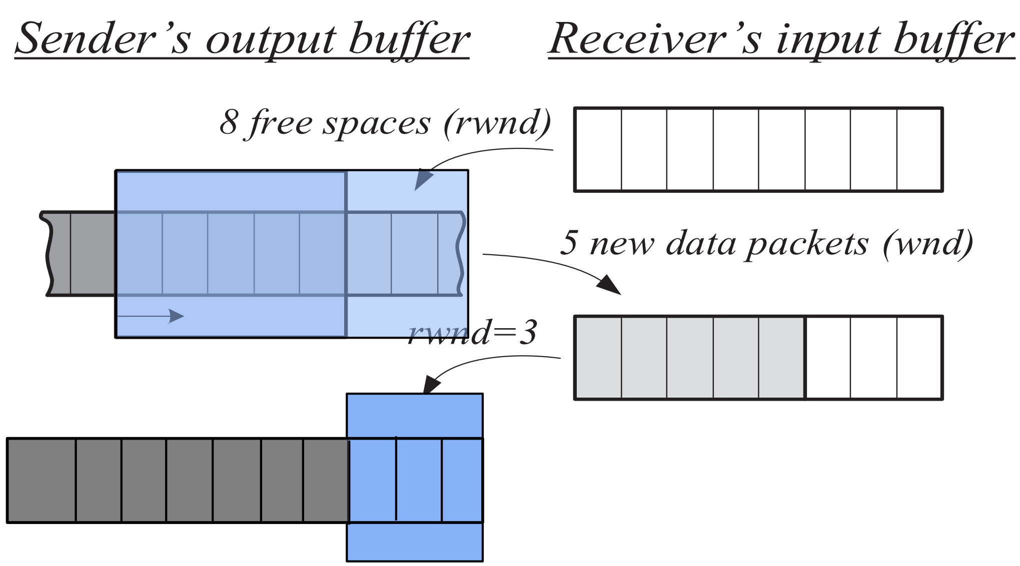
TCP Flow Control

Flow control: Prevent sender from overrunning receiver by transmitting too much data too fast

receiver: informs sender of amount of free buffer space

- Carried in **`RcvWindow`** field of TCP header of every arriving segment, **can change dynamically**

sender: keeps the amount of transmitted, unACKed data no more than most recently received **`RcvWindow`** value



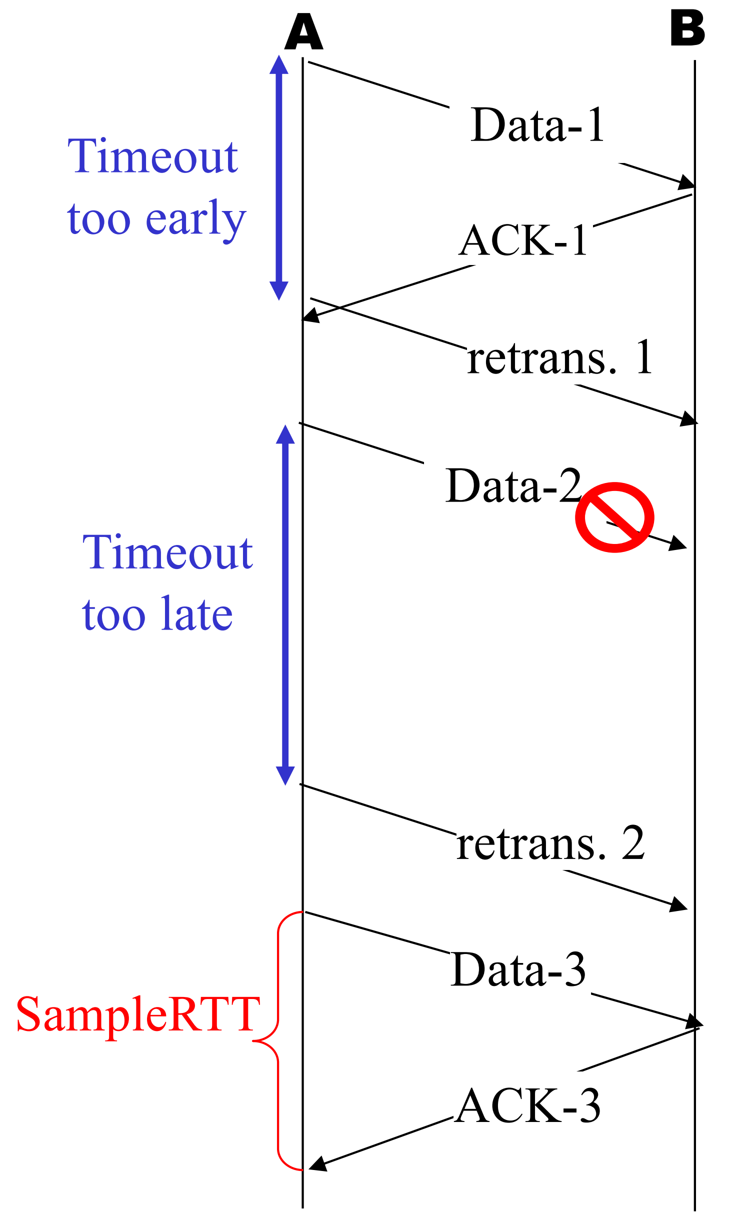
TCP loss detection and recovery

- TCP sets a retransmission timer (RTO) to detect packet losses
- A TCP connections sets one retransmission timer on the earliest sent, but unACKed segment **S**
  - If **S** gets ACKed, restart the timer on next unACKed segment
  - (reset timer when receiving ACK for new data)
- When the timer expires, retransmit starting from **S**
- *How many segments to retransmit?*
  - Receiver flow control window, rwnd
  - Congestion control window, cwnd (next lecture)
  - the number of segments that can be retransmitted: `min[cwnd, rwnd]`
    - Dependent on how segment loss is detected, see next lecture

important

Setting TCP Retransmission Timer

- TCP sets retrains. timer (RTO) based on estimated RTT
  - plus a "safety margin" (DevRTT)
- SRTT: estimated "smoothed" RTT
  - $SRTT = (1 - \alpha) \cdot SRTT + \alpha \cdot SampleRTT$
  - Exception: for the first measurement, $SRTT = SampleRTT$
- DevRTT: estimated RTT deviation
  - $DevRTT = (1 - \beta) \cdot DevRTT + \beta \cdot |SRTT - SampleRTT|$
  - Exception: for the first measurement: $DevRTT = \frac{SRTT}{2}$
- RTO: Retransmission timeout
  - $RTO = SRTT + 4 \cdot DevRTT$
- Typical parameters:
  - $\alpha = \frac{1}{8}, \quad \beta = \frac{1}{4},$



No need to remember details
Just understand the basic idea

- Network delay: random
- How to set retransmission timer:
  - Take measurements
  - Set the timer based on both average, and the variation
- Start the ball rolling: how to set the retransmission timer for the first packet of a connection?

One more question

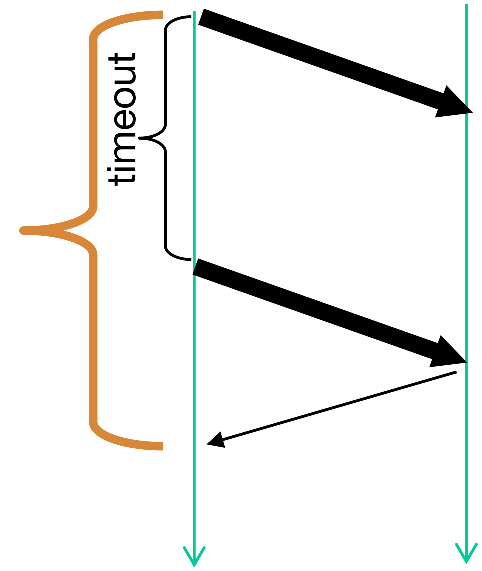
How to set the RTO value for the first segment?

Set a default value by some engineered guessing

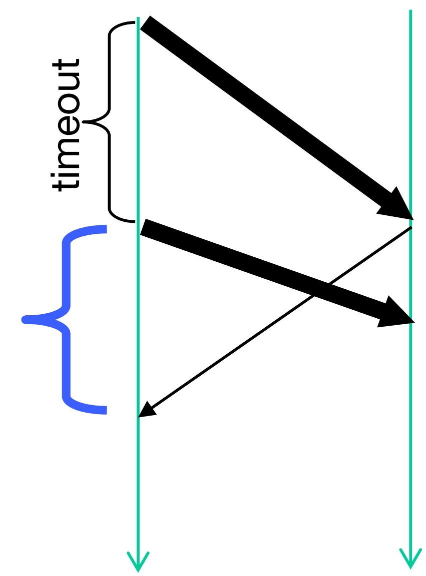
- what if the guessed value too small?
  - Unnecessary retransmissions
- what if the guessed value too large?
  - In case of first, or first few, packets being lost, wait longer than necessary before retransmission
- Current practice:
  - initial RTO = 1 sec (see RFC6298)
  - Once get the first sample RTT: SRTT$\leftarrow$sample RTT, DevRTT =SRTT/2

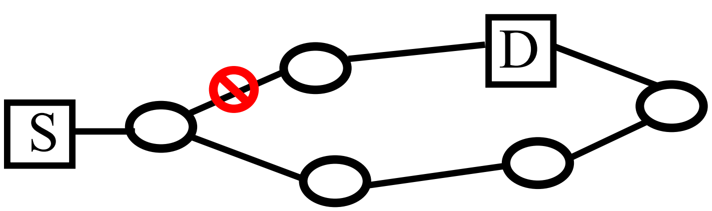
What to do in cases of *retransmissions*

- Taking measurement seems infeasible
  - take the delay between first transmission and final ACK?
  - take the delay between last retransmission of segment(n) and ACK(n)?



- Don’t measure?
  - Original path failed
  - New path is much longer
  - Without taking measurement, RTO got stuck with being too short





Karn's algorithm

in case of segment retransmission:

- do not take the RTT sample (i.e. no update to SRTT or DevRTT)
- double the retransmission timeout value (RTO) after each timeout
- Take RTT measure again upon next successful data transmission (receiving ACK without retransmission)

Computing RTO: an example

difference = SampleRTT – SRTT

SRTT = SRTT + 1/8 x difference

DevRTT = DevRTT +
1/4 (|difference| - DevRTT)

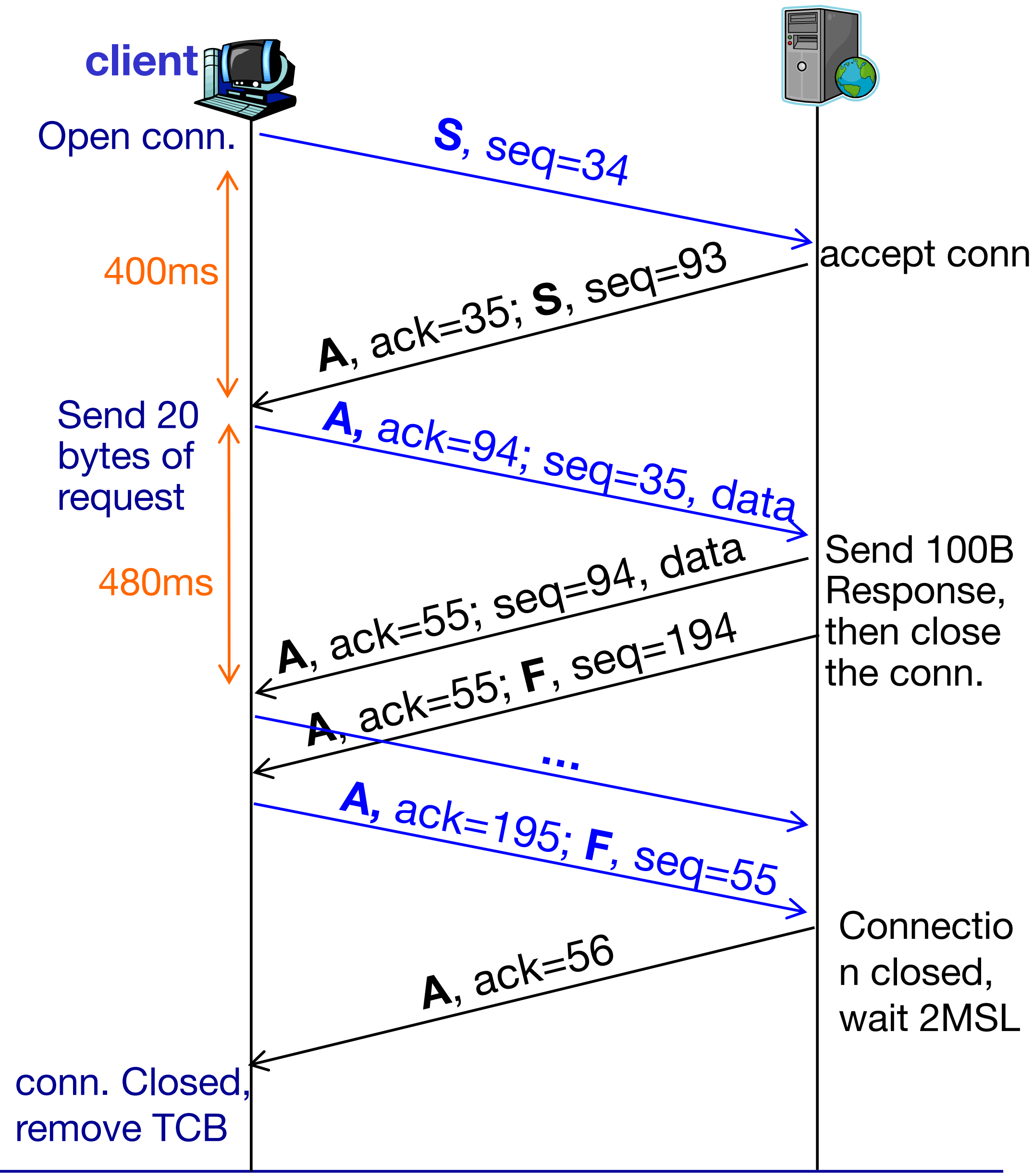
RTO = SRTT + 4 x DevRTT

Initialize: RTO = 1 second

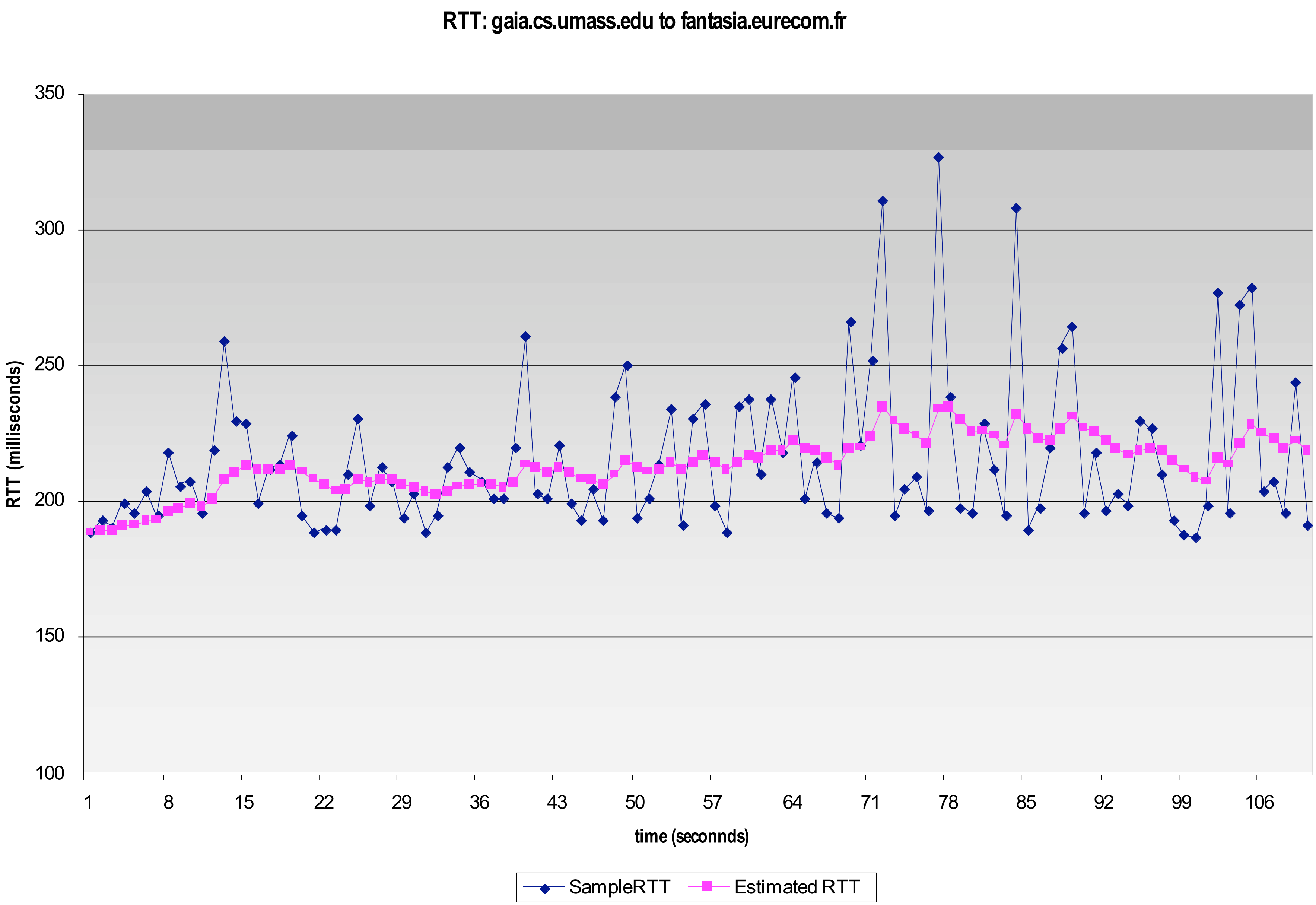
Upon receiving first packet:
SRTT = sample RTT
DevRTT = sample RTT / 2
SRTT = 400, DevRTT = 200

Upon receiving second packet:
diff = 480 – 400 = 80
SRTT = 400 + 10 = 410
DevRTT = 200 + ¼ (80-200) = 170

(from the earlier HTTP 1.0 connection example)



Example RTT estimation:

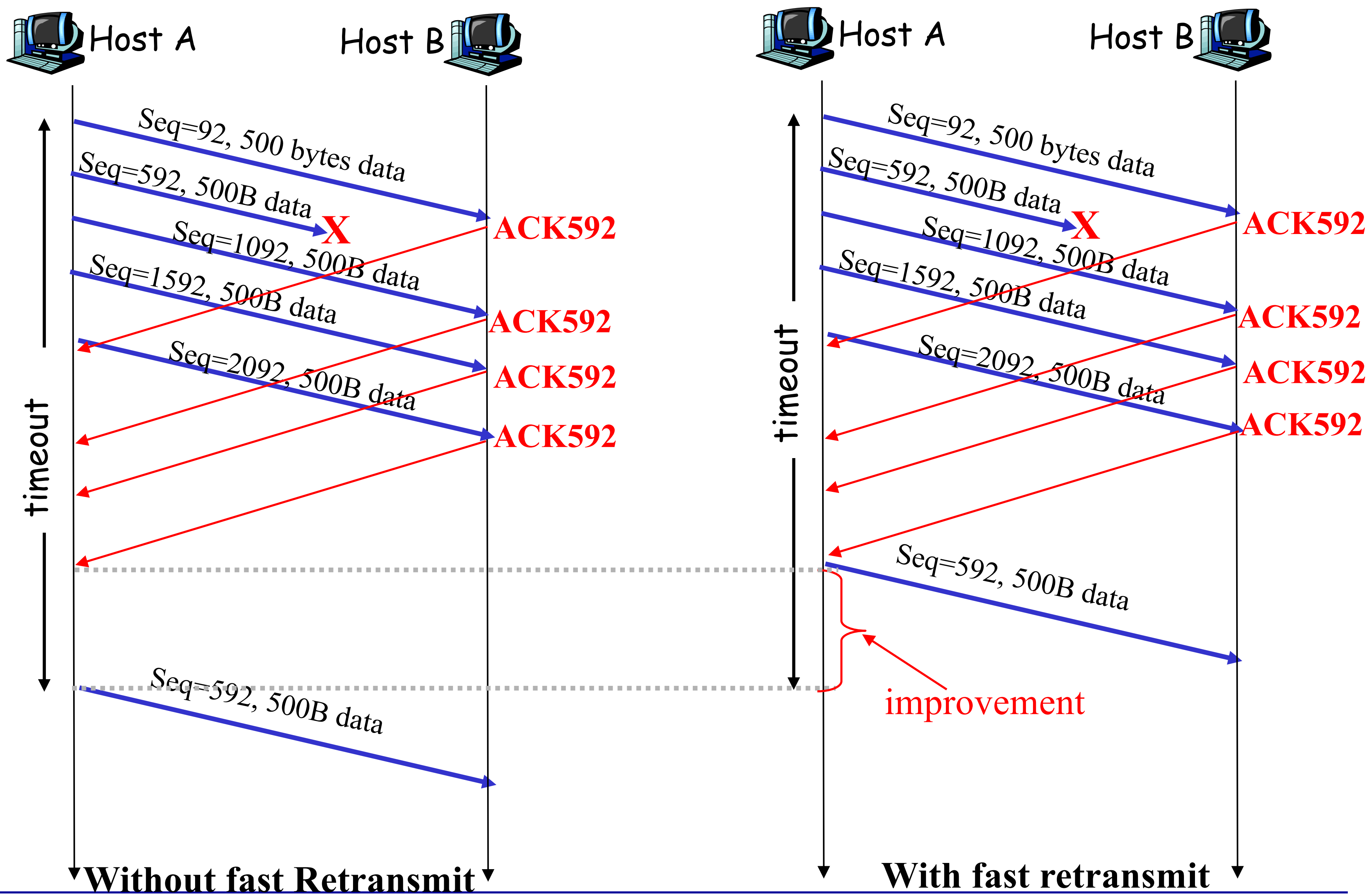


TCP Fast Retransmit

- RTO is set to a relatively long value
  - Aim at minimizing superfluous retransmission
  - long delay before resending lost packet
- Can detect lost segments via duplicate ACKs.
  - When a segment is lost, next arrival at receiver is out of order
  - When a segment arrives out of order, receiver can immediately sends an ACK indicating seq. # of next byte it is expecting
- When sender receives 3 duplicate ACKs for the same seq#(n), it assumes the segment with seq#(n) was lost

→fast retransmit: start retransmitting without waiting for the timer to expire
  - How many segments to retransmit? One only

TCP fast retransmit example



Yet another tweak of TCP: delayed ACK

FYI

- If a TCP connection carries traffic in both directions: ACKs are piggybacked on data segments
- For one-way data flow: If receiver sends an ACK after receiving everyone segment → double the packet count across the Internet
- **Delayed ACK**: after connection setup, upon receive one data segment S_1 :
  - wait a bit, see if next segment S_2 will arrive soon
  - If yes: sends an ACK for both
  - If no: send an ACK for S_1

Does this delayed-ACK screw up RTT measurement? Maybe a little

TCP Receiver: when to send ACK?

Event at TCP receiver	TCP Receiver action
in-order segment arrival, no gaps, everything earlier already ACKed 	delayed ACK: wait up to 500ms, If nothing arrived, send ACK
in-order segment arrival, no gaps, one delayed ACK pending	immediately send one cumulative ACK
out-of-order arrival: higher-than-expect seq. #, gap detected 	Immediately send ACK, indicating seq. # of next expected byte
arrival of segment that partially or completely fills a gap 	immediate send ACK if segment starts at the lower end of the gap

Summary

- Connection management (SYN, FIN)
- Flow control for reliable delivery (sequence numbers, ACK)
  - ACK is a flag in the header; ACK flag == 0, ACK number in the header makes no sense (value ignored)
- Two-way communication
  - Separate sequence number management for both directions
- Error detection and recovery
  - Retransmission timer
  - Fast retransmit
- Receiver’s flow control
  - Avoid overwhelming the receiver
- Congestion control
  - Avoid overwhelming the network

important

After obtain a new RTT sample:

- difference = SampleRTT - SRTT
- SRTT' = (1-α) x SRTT + α x SampleRTT

  = SRTT + α x difference
- DevRTT' = (1-β) x DevRTT + β x |difference|

  = DevRTT + β (|difference| - DevRTT)

- Retransmission Timer (RTO) = SRTT + 4 x DevRTT

Typically: α = 1/8, β = 1/4