The Transport Layer

TCP Service

School of Software Engineering
South China University of Technology
Dr. Chunhua Chen
chunhuachen@scut.edu.cn
2016 Spring

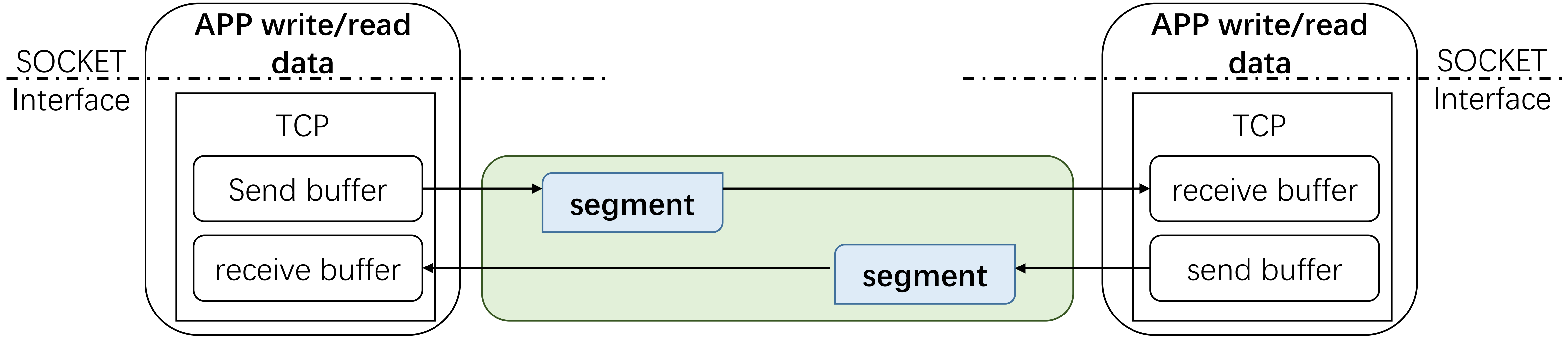
Goals of this Lecture

- learn about transport layer protocols in the Internet:
  - UDP: connectionless transport
  - TCP: connection-oriented transport
  - TCP congestion control
- For TCP, we learn its
  - segment structure
  - reliable data transfer
  - flow control
  - connection management

TCP overview

RFCs: 793, 1122, 1323, 2018, 2581

- Point-to-Point (End-to-End)
  - from one sender to one receiver
- Connection-oriented
  - handshaking (exchange of control msgs) to establish sender, receiver state before data exchange



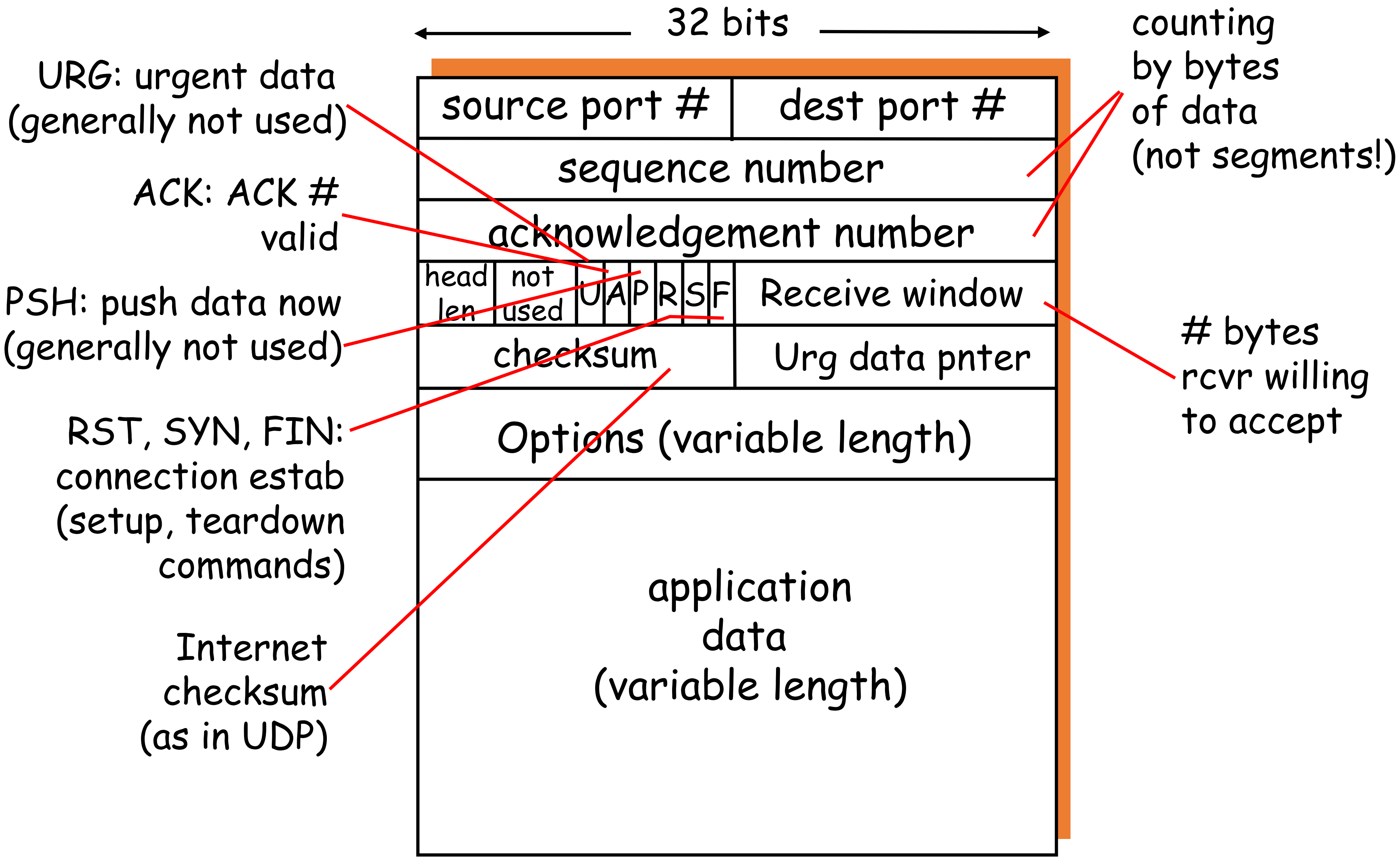
TCP overview

RFCs: 793, 1122, 1323, 2018, 2581

- Send & receive buffers at both ends
- Full duplex data: (ACKing while transmitting app data)
  - bi-directional data flow in same connection
- Reliable, in-order byte steam (Rdt)
  - no “message boundaries”
  - MSS: maximum segment size
- Pipelined:
  - TCP congestion and flow control set window size
- Flow controlled:
  - sender will not overwhelm receiver
- Congestion control
  - do good for the all network

TCP Segment Structure

more fields for Rdt



TCP #seq and #ACK

view data as an unstructured, but ordered, stream of bytes.

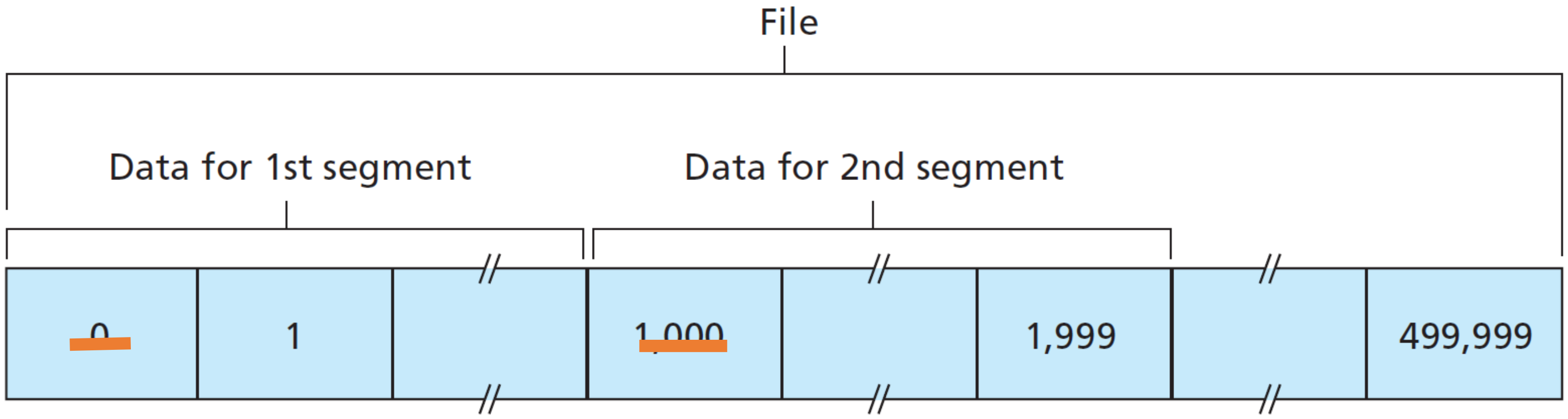


Figure 3.30 ♦ Dividing file data into TCP segments

- #Seq: 1) byte stream "number" of first byte in segment's data 2) random init #seq
- #ACK: 1) seq # of next byte expected from other side 2) cumulative ACK

Q: how receiver handles out-of-order segments

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

A Telnet case:

a char is one byte

Starting #seq for
A and B are 42 and 79

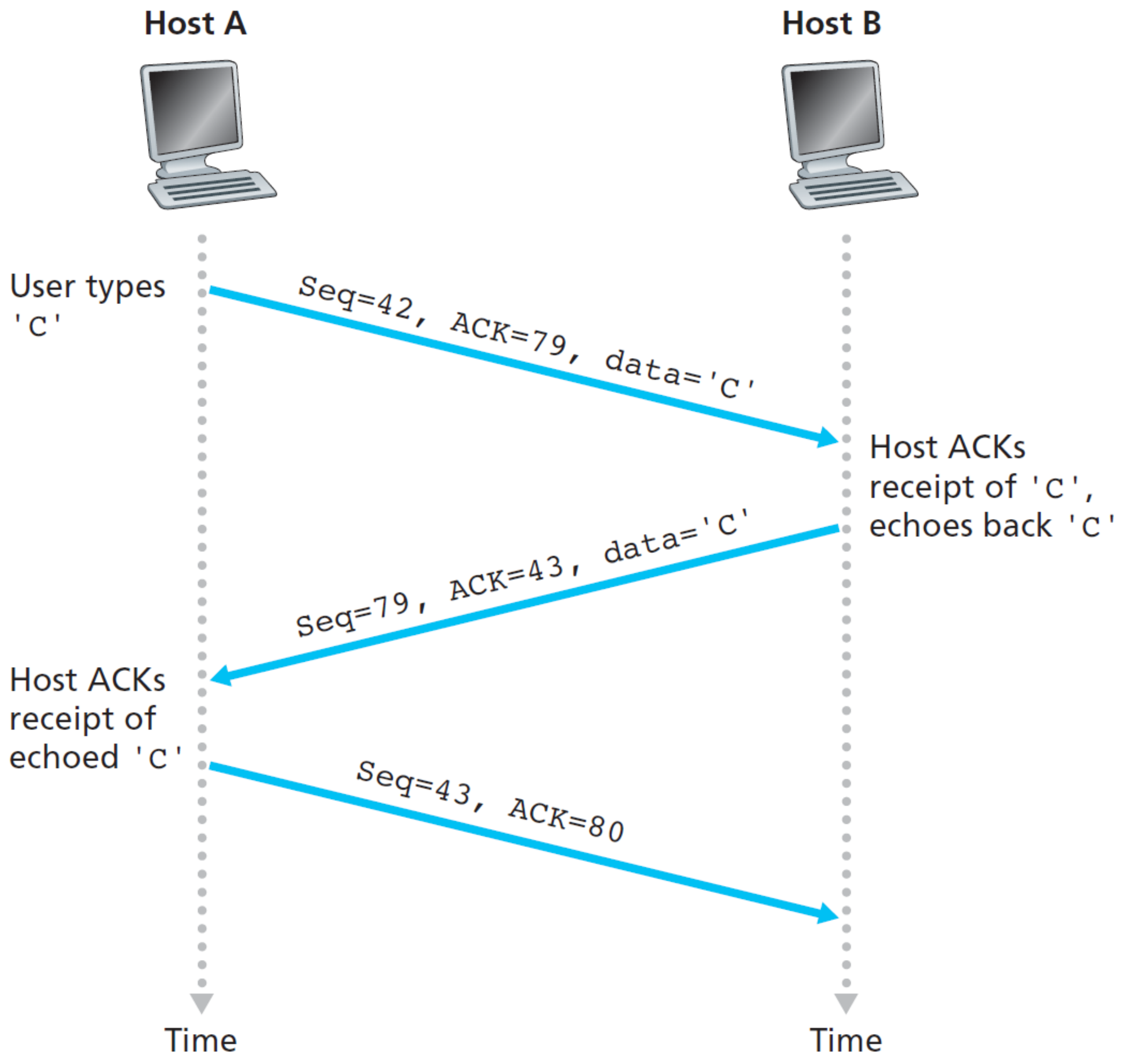
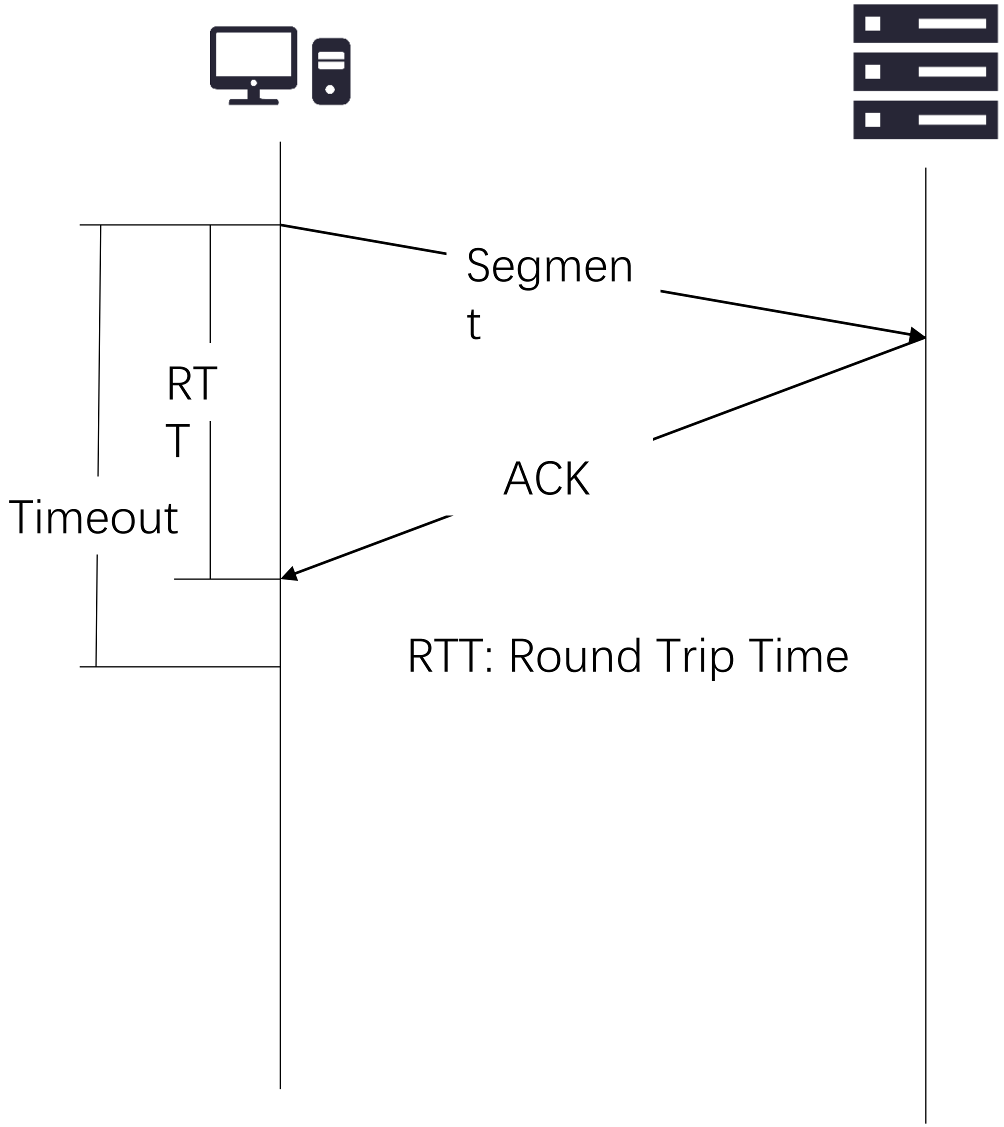


Figure 3.31 ♦ Sequence and acknowledgment numbers for a simple Telnet application over TCP

TCP: Timeout Interval

adapting to current network situations

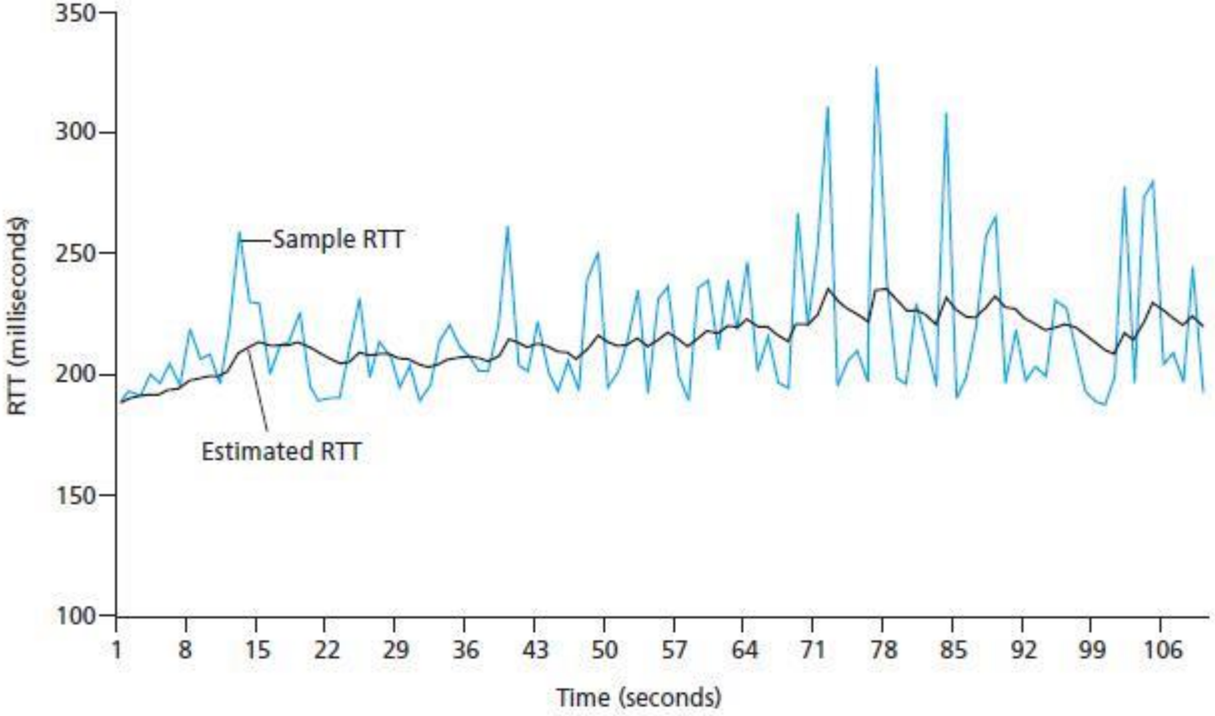
- Timer is a simple concept, but setting the timeout interval is considered as difficult in real life protocol.
- Timeout interval depends on network congestion, which fluctuates.
- To set: Timeout ? RTT
  - Too long? slow reaction to segment loss
  - Too short? unnecessary retransmissions
- Two problems
  - Estimate RTT
  - Define Timeout-RTT



RTT Estimation and Timeout

Q: how to estimate RTT?

- 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



RTT Estimation and Timeout

Principle: puts more weight on recent samples than on old samples

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

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

$E(0)=S(0)$
$E(1)=(1-\alpha)E(0)+\alpha S(1)=(1-\alpha)S(0)+\alpha S(1)$
$E(2)=(1-\alpha)E(1)+\alpha S(2)=(1-\alpha)^2S(0)+(1-\alpha)\alpha S(1)+ \alpha S(2)$
$E(3)=(1-\alpha)E(2)+\alpha S(3)=(1-\alpha)^3S(0)+(1-\alpha)^2\alpha S(1)+(1-\alpha)\alpha S(2)+\alpha S(3)$

… …

RTT Estimation and Timeout

Setting the timeout

- EstimatedRTT plus “safety margin”
  - large variation in EstimatedRTT -> larger safety margin
- first estimate of how much SampleRTT deviates from EstimatedRTT:

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

(typically, β = 0.25)


TimeoutInterval = EstimatedRTT + 4*DevRTT
```

TCP: Reliable Data Transfer

- TCP creates rdt service on top of IP's unreliable service
- Pipelined segments
- Cumulative ACKs
- Cache out-of-order segments
- TCP uses single retransmission timer

- Retransmissions are triggered by:
  - timeout events
  - Three duplicate ACKs
- Initially consider simplified TCP sender:
  - ignore duplicate ACKs
  - ignore flow control, congestion control
  - Data on one direction

A Simplified Rdt

Timer: associated with the oldest unacknowledged segment.

```
NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber

loop (forever) {
    switch(event)

        event: data received from application above
            create TCP segment with sequence number NextSeqNum
            if (timer currently not running)
                start timer
            pass segment to IP
            NextSeqNum=NextSeqNum+length(data)
            break;

        event: timer timeout
            retransmit not-yet-acknowledged segment with
                smallest sequence number
            start timer
            break;

        event: ACK received, with ACK field value of y
            if (y > SendBase) {
                SendBase=y
                if (there are currently any not-yet-acknowledged segments)
                    start timer
                }
            break;

} /* end of loop forever */
```

Subtle cases

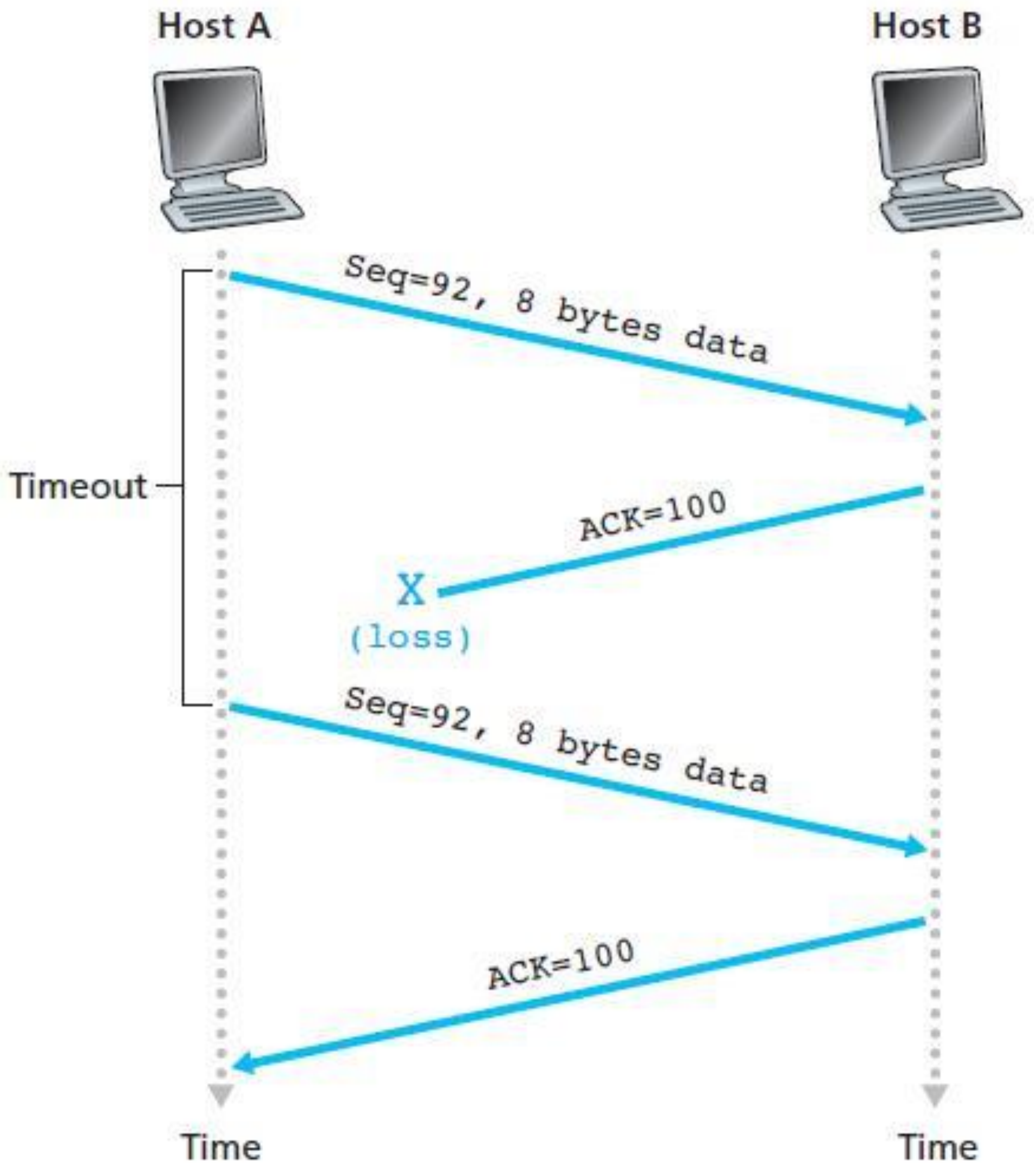


Figure 3.34 ♦ Retransmission due to a lost acknowledgment

Subtle cases

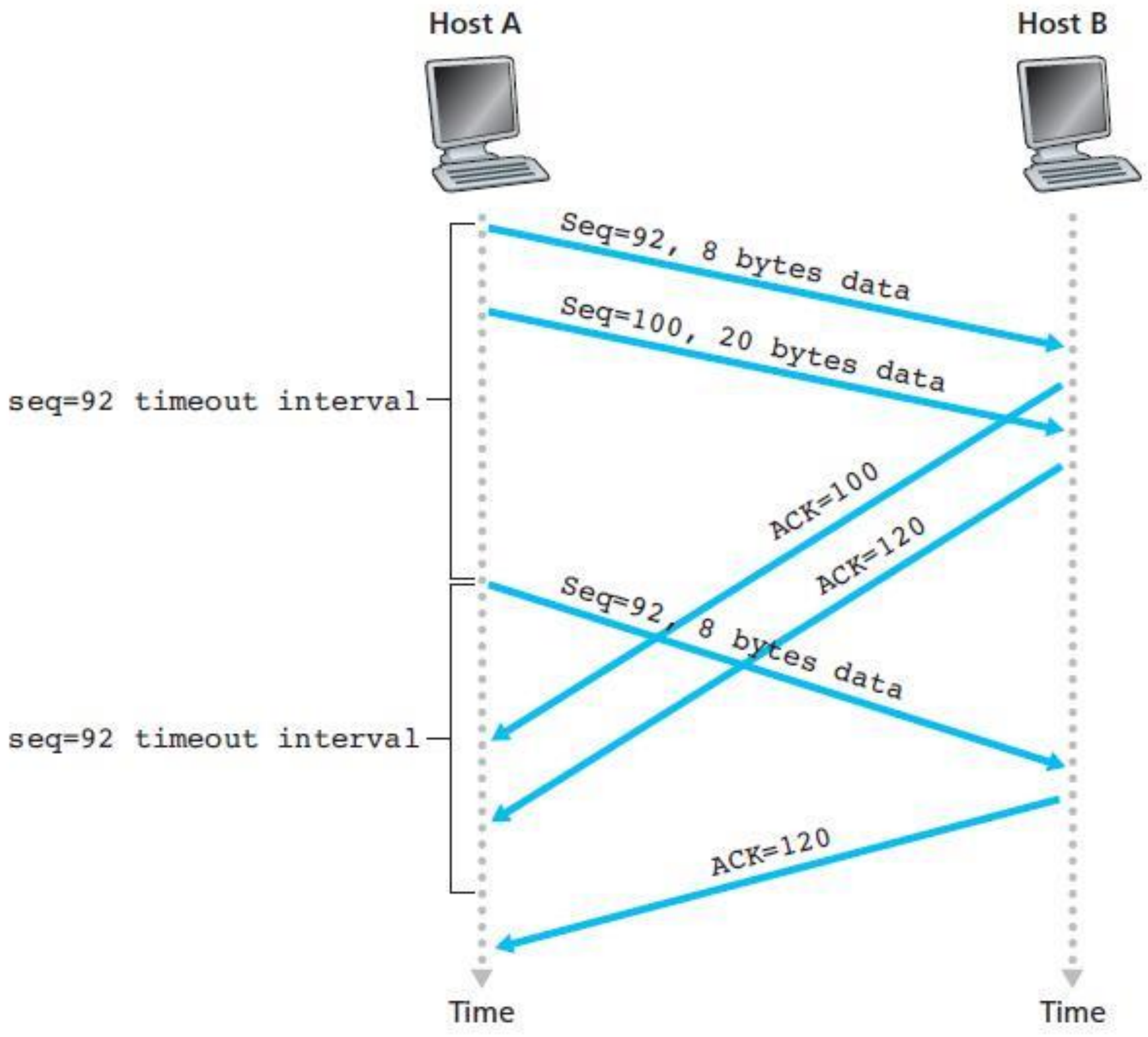


Figure 3.35 ♦ Segment 100 not retransmitted

Subtle cases

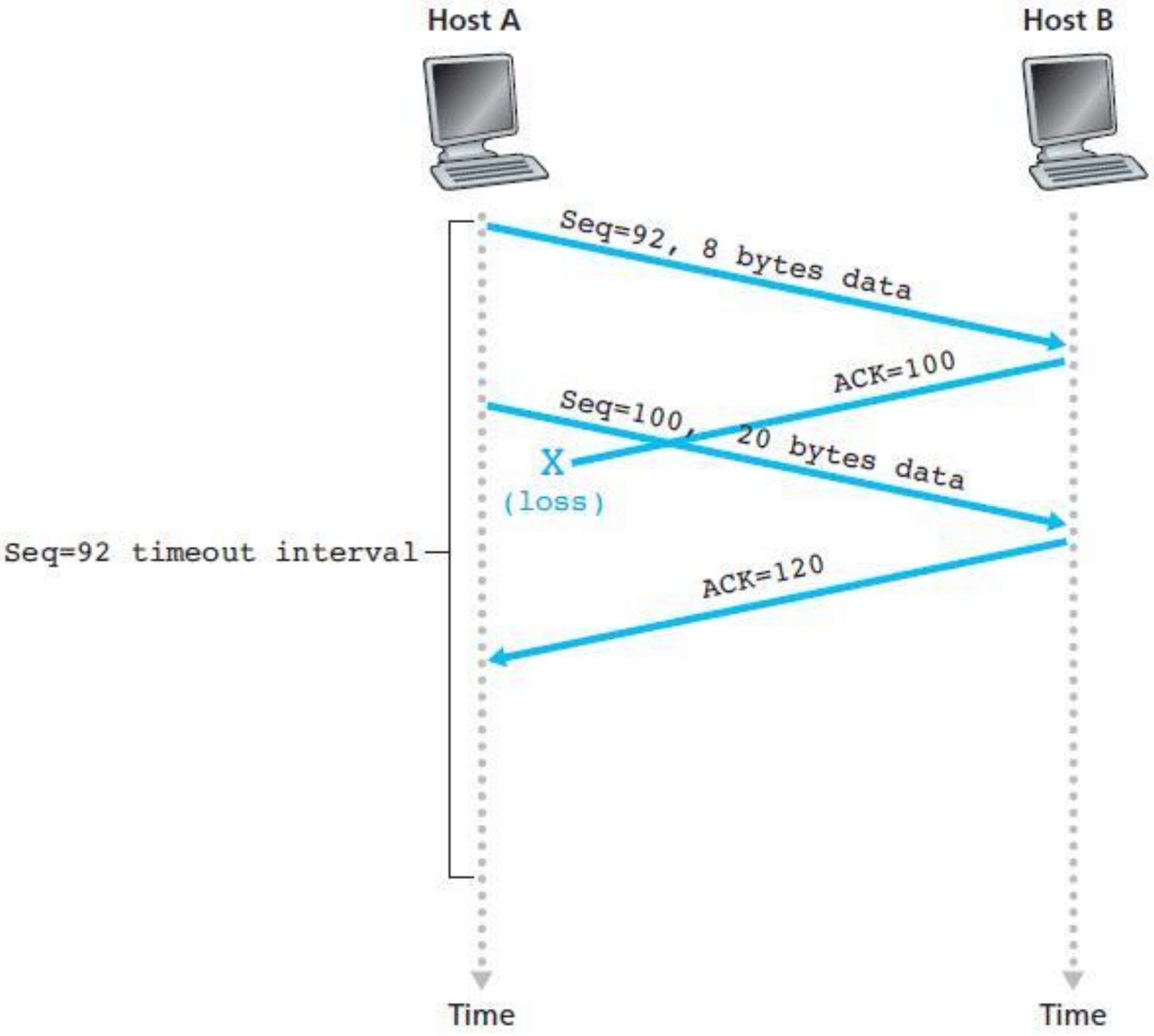


Figure 3.36 ♦ A cumulative acknowledgment avoids retransmission of the first segment

Improvement: Double the Timeout Interval

in case of timeout event

- What timeout event means? Why special?
- Reset timer:
    - Timeout event: the next timeout interval to twice the previous value
    - data received from application above: deriving it from the last EstimatedRTT and DevRTT
    - ACK received: deriving it from the last EstimatedRTT and DevRTT
- Double the Timeout Interval in case of timeout provides a limited form of congestion control!

Improvement: Fast Retransmit

handling lost of segments

- If we can know for sure that a segment has been lost, retransmission should be start immediately!!! For sake of end-to-end delay!
- The sender can detect packet loss well before the timeout event occurs by noting so-called duplicate ACKs.

Event	TCP Receiver Action
Arrival of in-order segment with expected sequence number. All data up to expected sequence number already acknowledged.	Delayed ACK. Wait up to 500 msec for arrival of another in-order segment. If next in-order segment does not arrive in this interval, send an ACK.
Arrival of in-order segment with expected sequence number. One other in-order segment waiting for ACK transmission.	Immediately send single cumulative ACK, ACKing both in-order segments.
Arrival of out-of-order segment with higher-than-expected sequence number. Gap detected.	Immediately send duplicate ACK, indicating sequence number of next expected byte (which is the lower end of the gap).
Arrival of segment that partially or completely fills in gap in received data.	Immediately send ACK, provided that segment starts at the lower end of gap.

Table 3.2 ♦ TCP ACK Generation Recommendation [RFC 5681]

Fast Retransmit

```
event: ACK received, with ACK field value of y
          if (y > SendBase) {
                SendBase=y
                if (there are currently any not yet
                             acknowledged segments)
                   start timer
                }
          else { /* a duplicate ACK for already ACKed
                segment */
             increment number of duplicate ACKs
                received for y
             if (number of duplicate ACKS received
                for y==3)
                /* TCP fast retransmit */
                resend segment with sequence number
             }
          break;
```

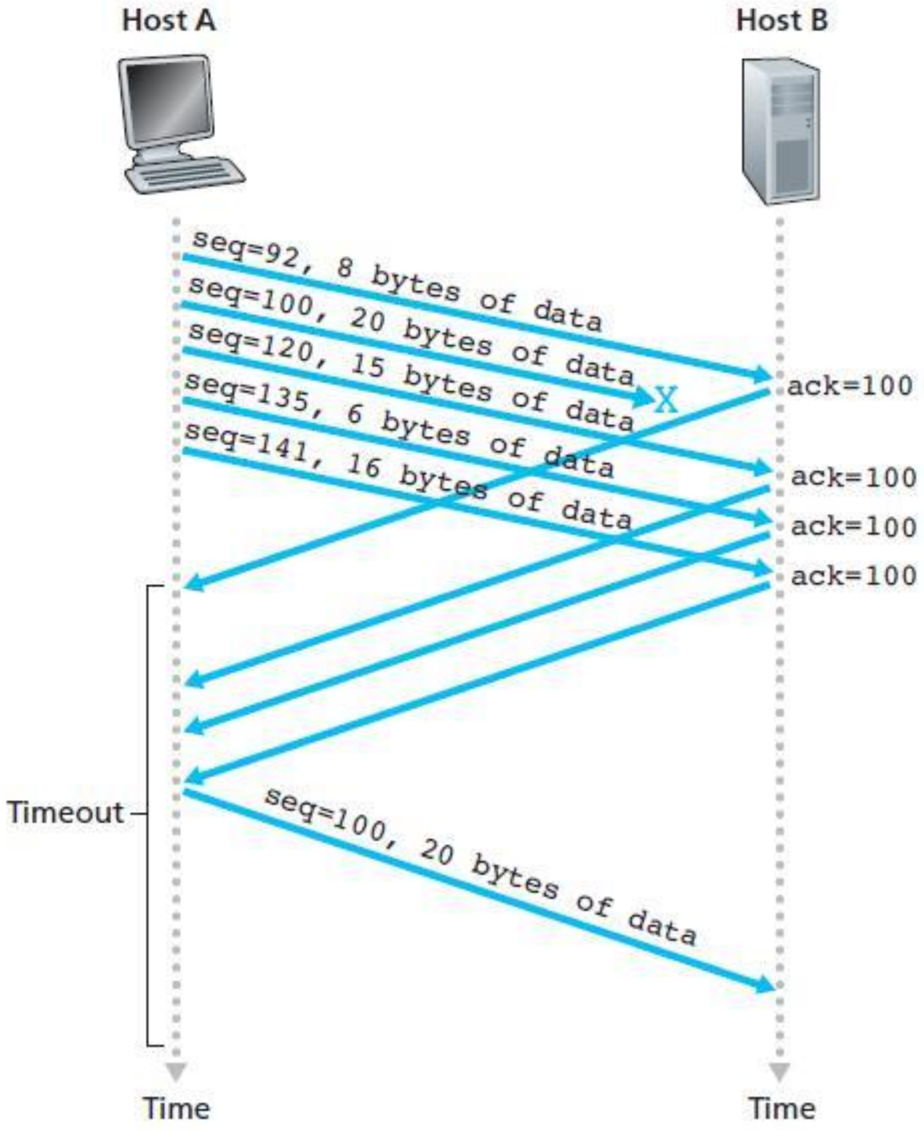
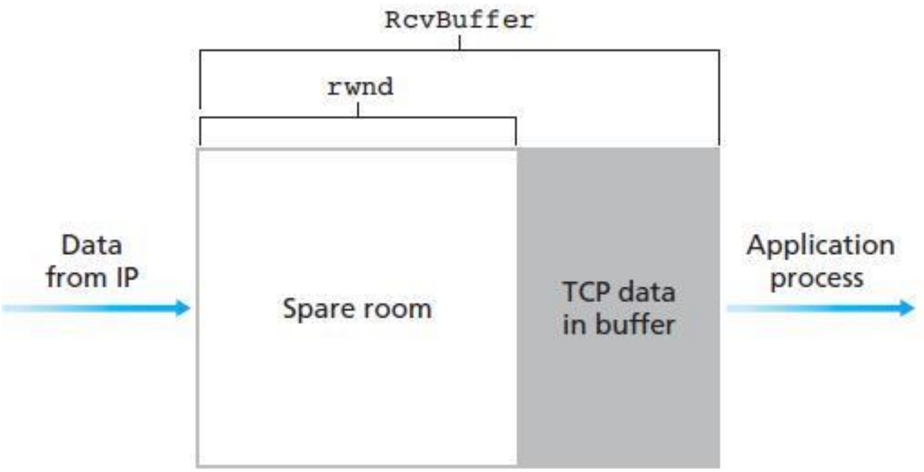


Figure 3.37 ♦ Fast retransmit: retransmitting the missing segment before the segment's timer expires

Flow Control

one form of limiting pipelined sending

- Receive side of TCP connection has a receive buffer:



- **app process may be slow at reading from buffer**

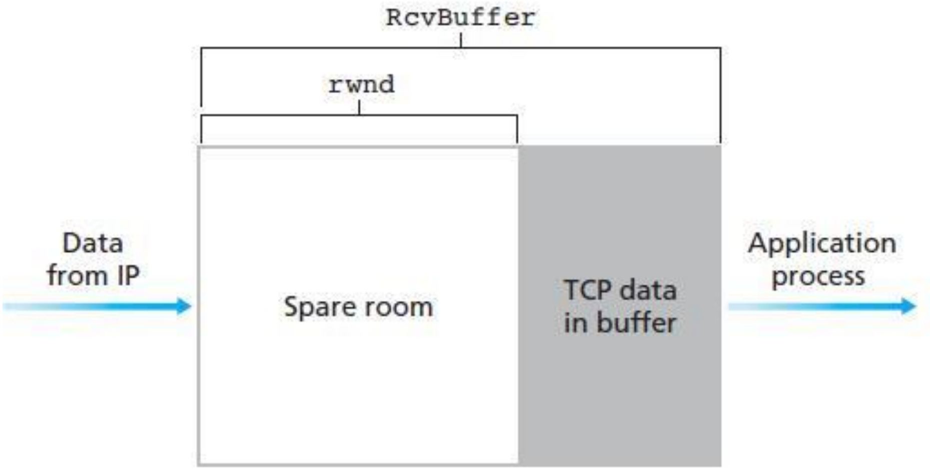
flow control

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

- speed-matching service: matching the send rate to the receiving app's drain rate

TCP Flow control: how it works

No cache of out-of-order segments in this case



- Sender limits unACKed data to RcvWindow
  - guarantees receive buffer doesn't overflow

`LastByteSent – LastByteAcked < RcvWindow`

- Receiver advertises spare room by including value of RcvWindow in segments

`Spare room: RcvWindow`

`= RcvBuffer-[LastByteRcvd - LastByteRead]`

`What happen sender receive RcvWindow = 0?`

TCP Connection Management

Establishment: to initialize TCP variables: #seq; buffers, flow control info

Three way handshake:

Step 1: client host sends TCP SYN segment to server
- specifies initial #seq
- no APP data

Step 2: server host receives SYN, replies with SYNACK segment
- server allocates buffers
- specifies server initial #seq.

Step 3: client receives SYNACK, replies with ACK segment, which may contain App data

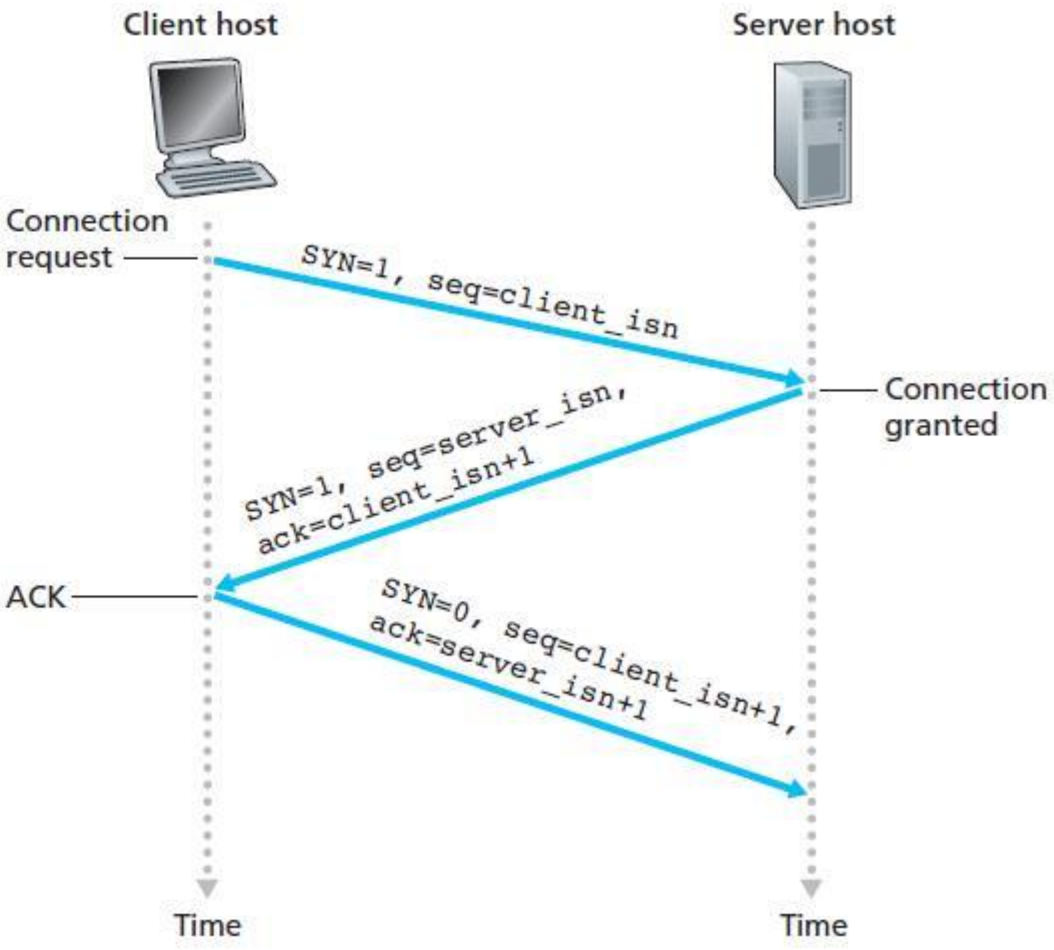
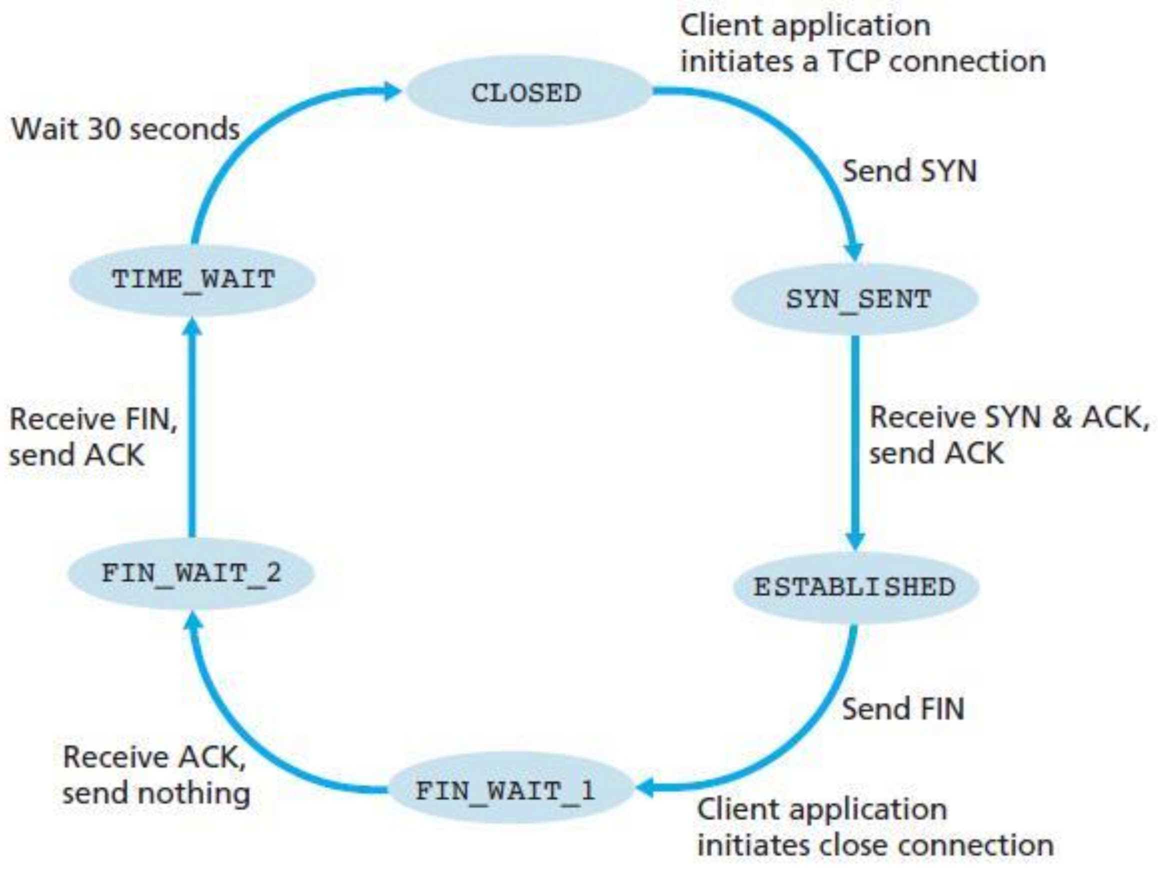


Figure 3.39 ♦ TCP three-way handshake: segment exchange

TCP Connection Management

Teardown: can be triggered by either sid



gure 3.41 ♦ A typical sequence of TCP states visited by a client TCP

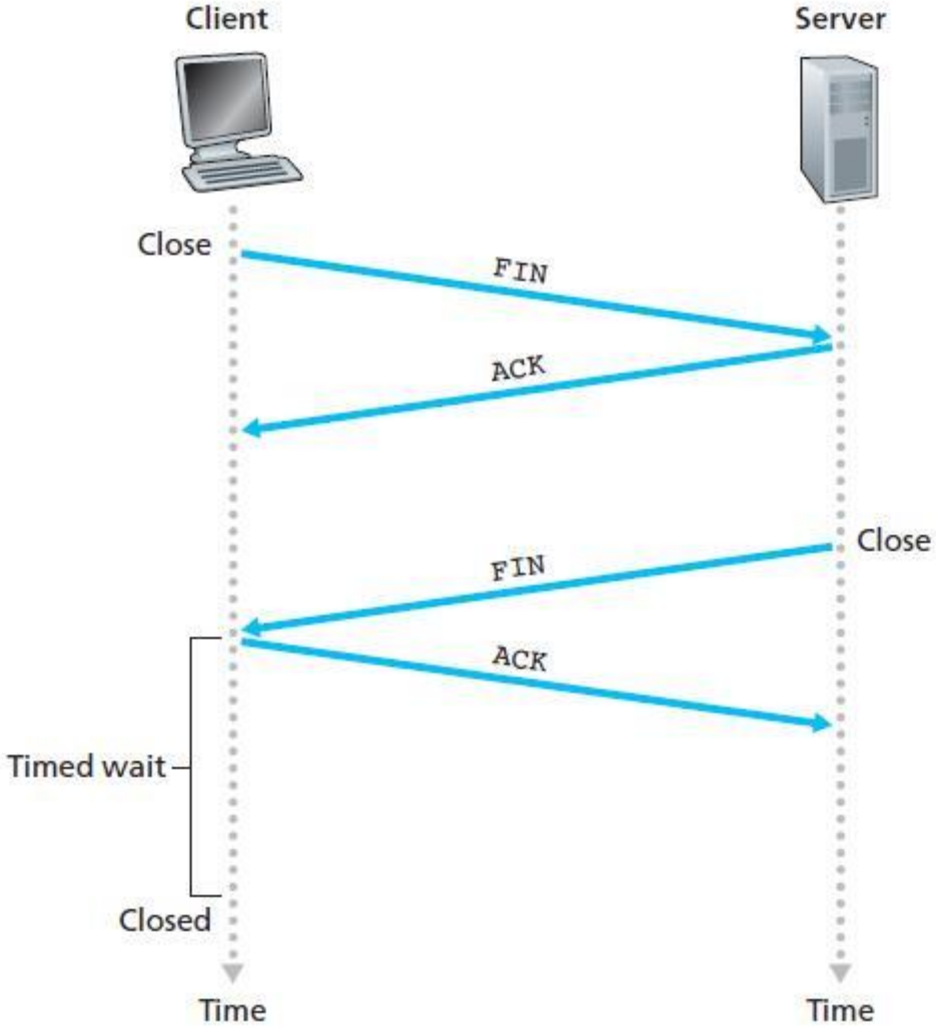
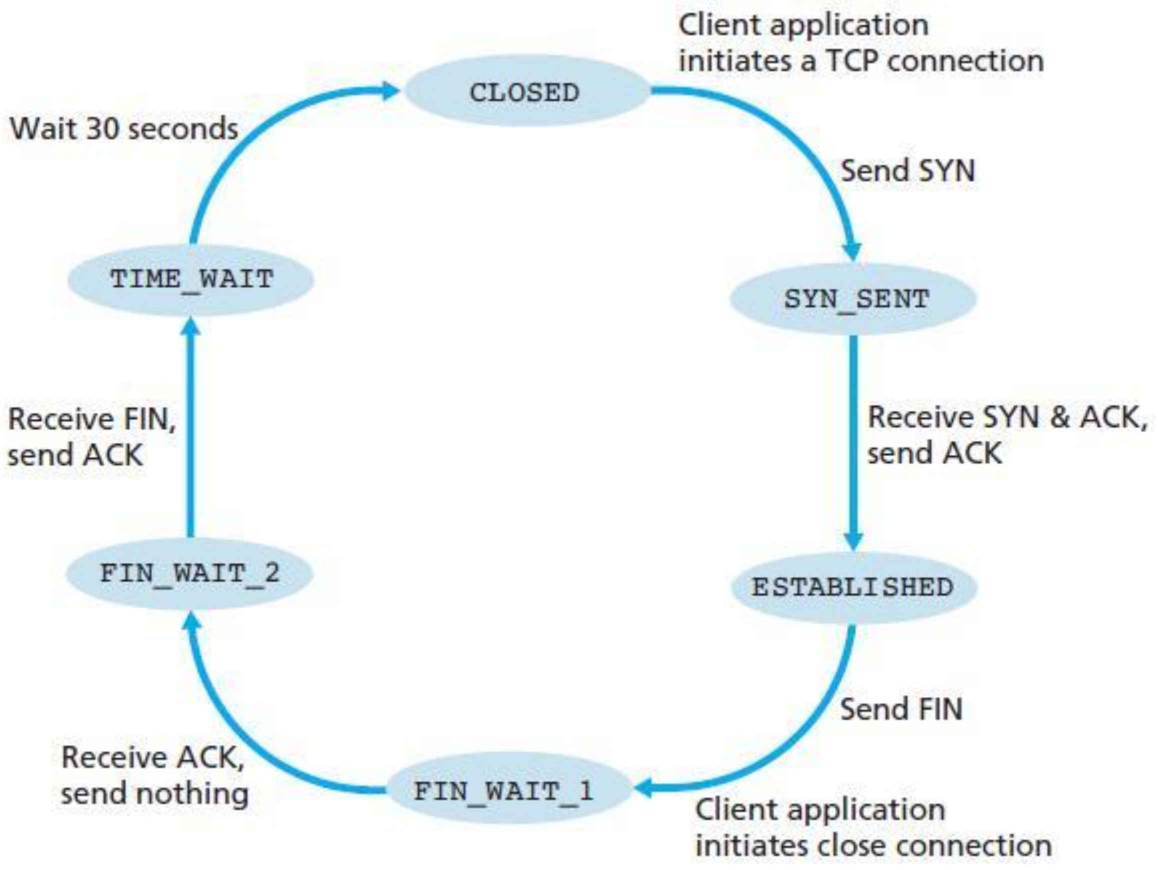


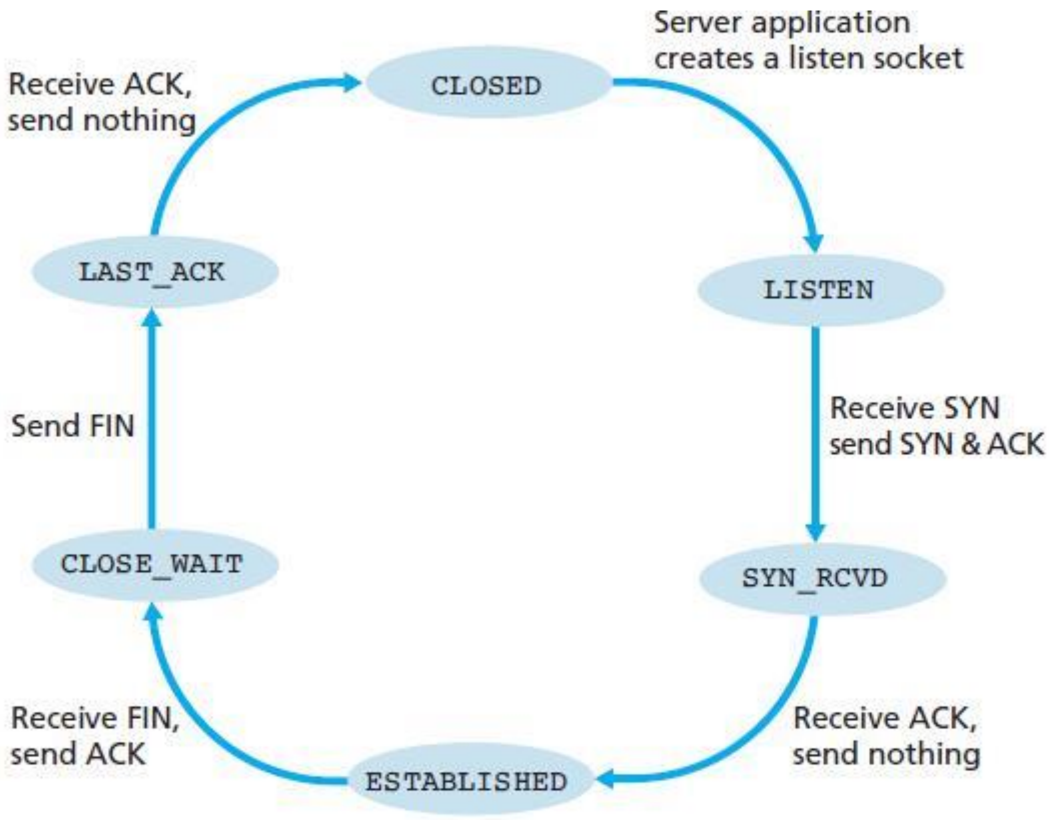
Figure 3.40 ♦ Closing a TCP connection

TCP Connection Management

Teardown: can be triggered by either side



gure 3.41 ♦ A typical sequence of TCP states visited by a client TCP



ıre 3.42 ♦ A typical sequence of TCP states visited by a server-side TCP