Transport Layer Protocols -3

A/PROF. DUY NGO

Learning Objectives

3.5 connection-oriented transport: TCP

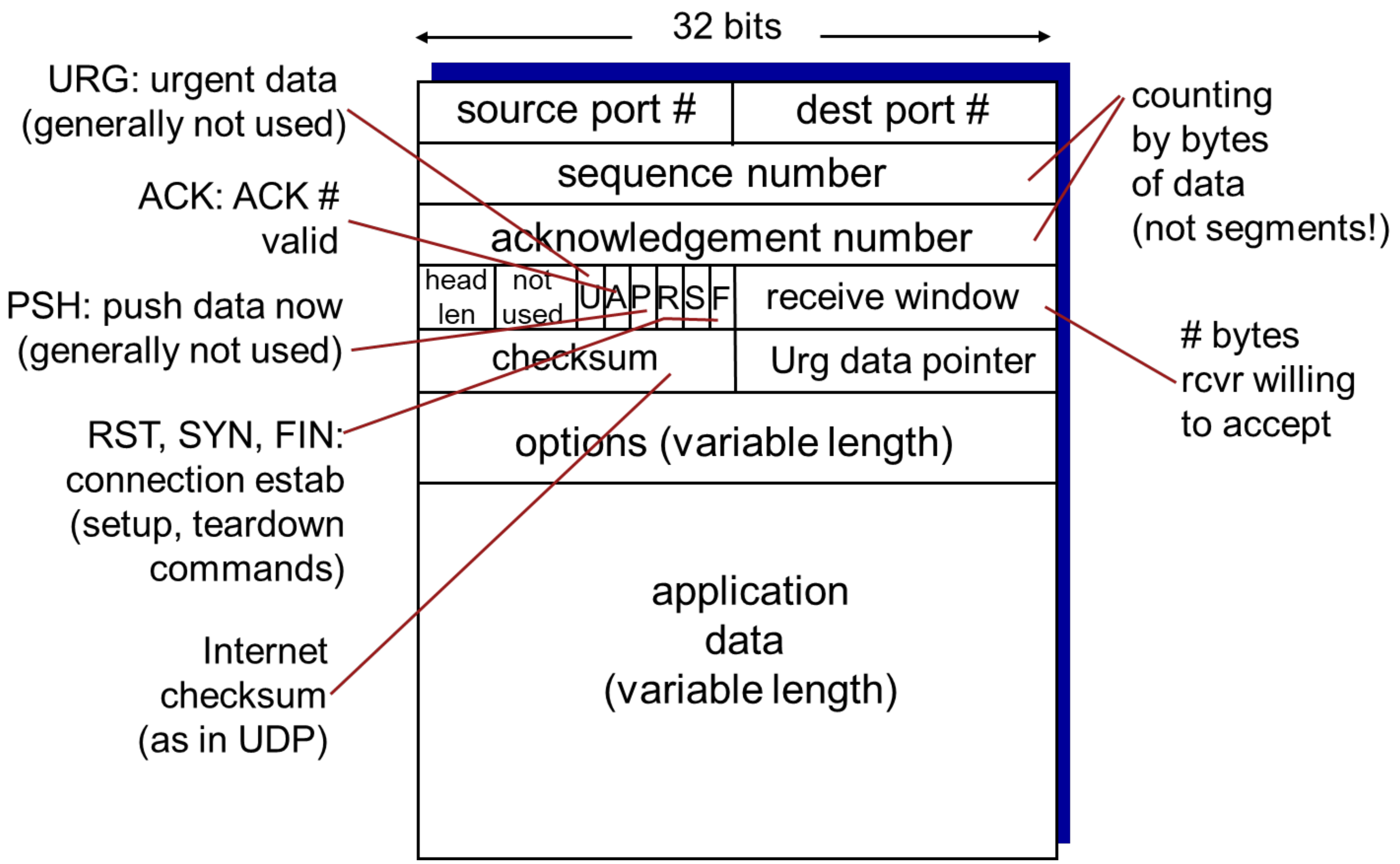
– segment structure
– reliable data transfer
– flow control
– connection management

COPYRIGHT @2017, 2013, 2010 PEARSONS EDUCATION INC.

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

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



TCP Sequence Numbers, ACKs (1 of 2)

sequence numbers:

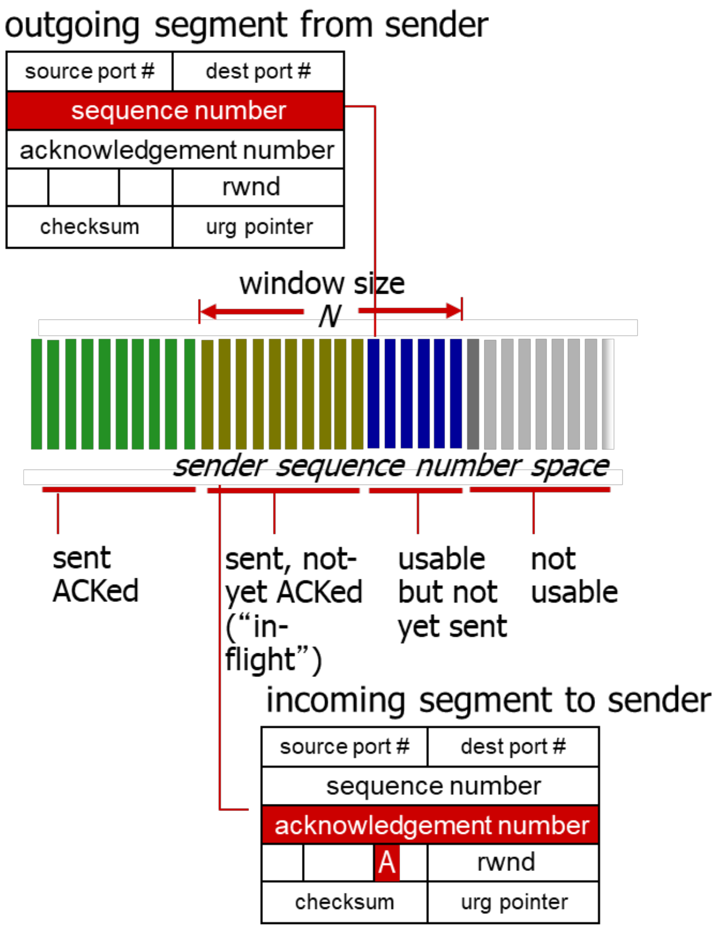
- byte stream "number" of first byte in segment's data

acknowledgements:

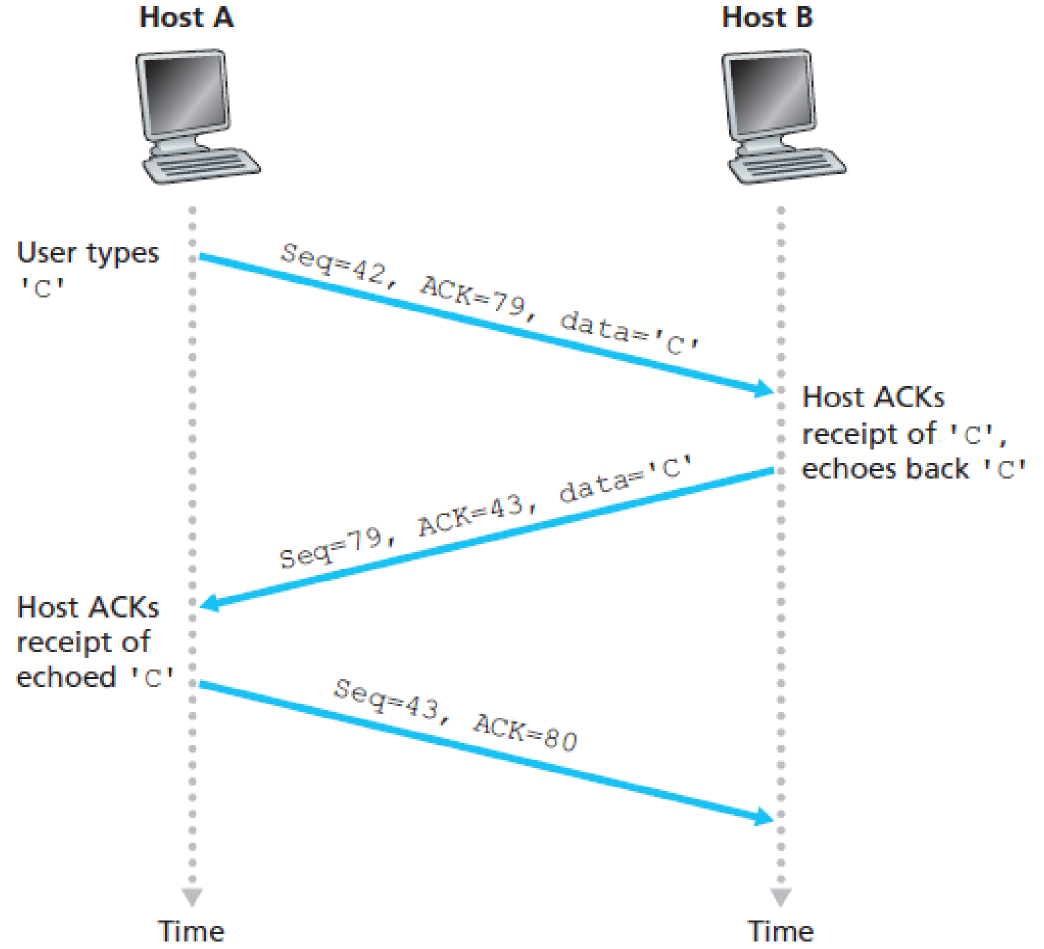
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

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



TCP Sequence Numbers, ACKs (2 of 2)



 COPYRIGHT @2017, 2013, 2010 PEARSONS EDUCATION INC.

TCP Round Trip Time, Timeout (1 of 3)

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

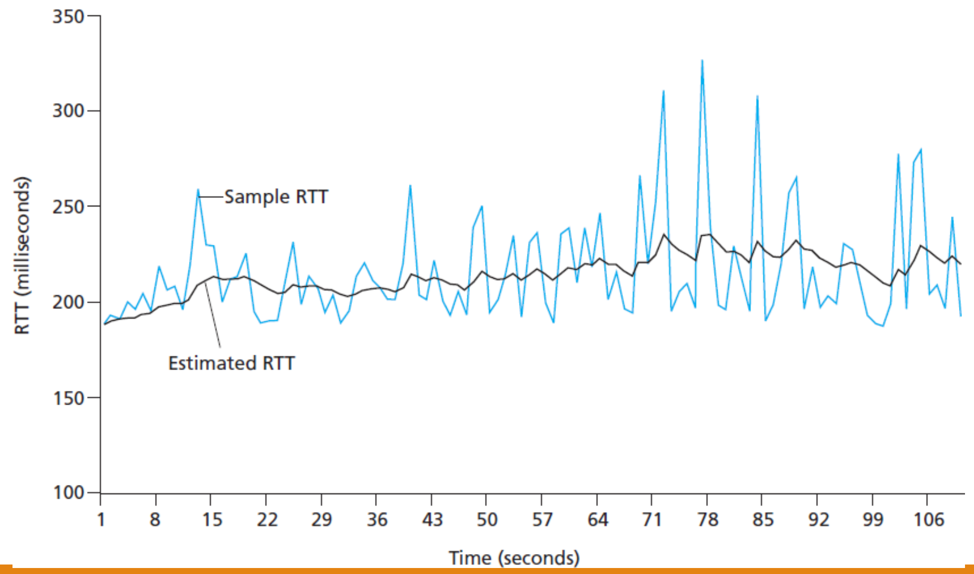
Q: how to estimate RTT?

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

TCP Round Trip Time, Timeout (2 of 3)

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

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



TCP Round Trip Time, Timeout (3 of 3)

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”

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

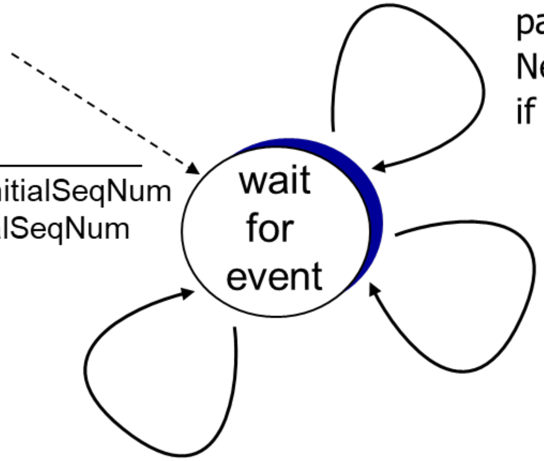
TCP Sender (Simplified)

data received from application above

create segment, seq. #: NextSeqNum
pass segment to IP (i.e., “send”)
NextSeqNum = NextSeqNum + length(data)
if (timer currently not running)
    start timer

Λ

NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum



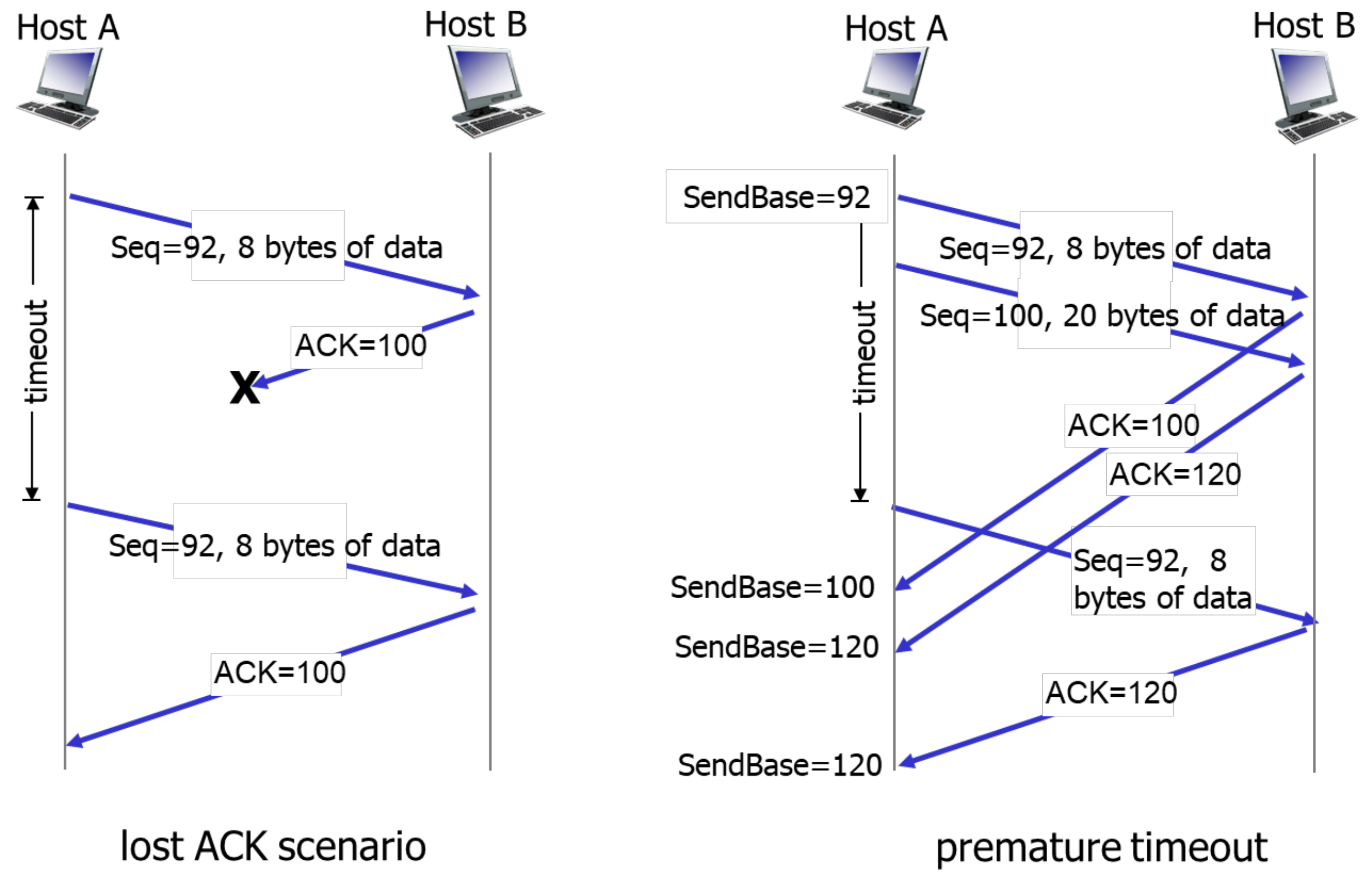
timeout

retransmit not-yet-acked segment
    with smallest seq. #
start timer

ACK received, with ACK field value y

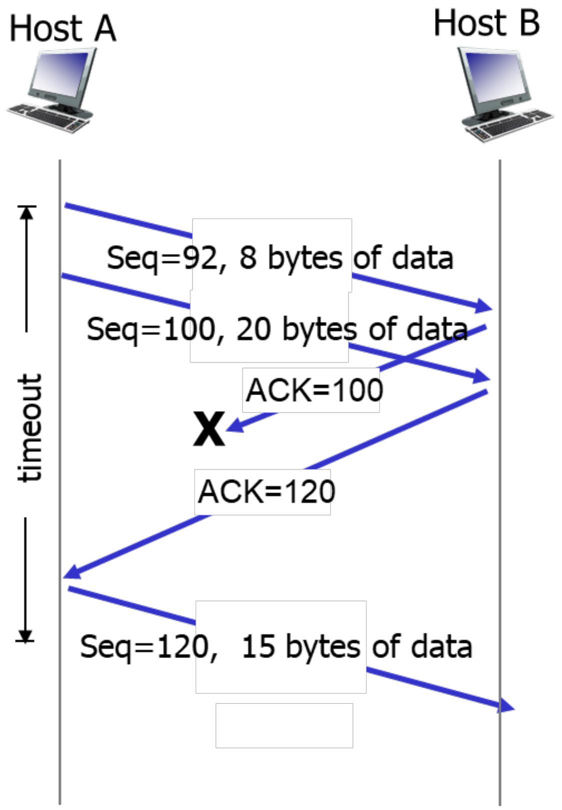
```
if (y > SendBase) {
   SendBase = y
   /* SendBase–1: last cumulatively ACKed byte */
   if (there are currently not-yet-acked segments)
       start timer
     else stop timer
  }
```

TCP: Retransmission Scenarios (1 of 2)



 COPYRIGHT @2017, 2013, 2010 PEARSONS EDUCATION INC.

TCP: Retransmission Scenarios (2 of 2)



cumulative ACK

 COPYRIGHT @2017, 2013, 2010 PEARSONS EDUCATION INC.

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 Sequence # . Gap detected	immediately send **duplicate ACK,** indicating Sequence # 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 (1 of 2)

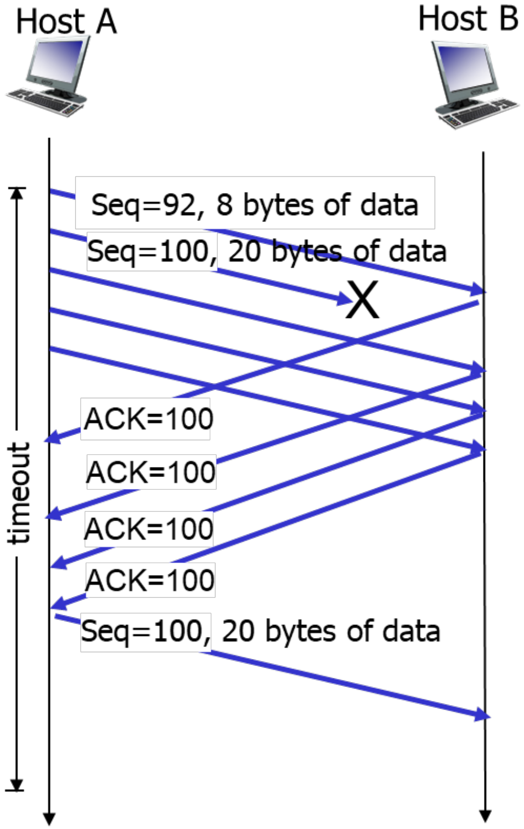
- time-out period often relatively long:
  - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs.

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 (2 of 2)

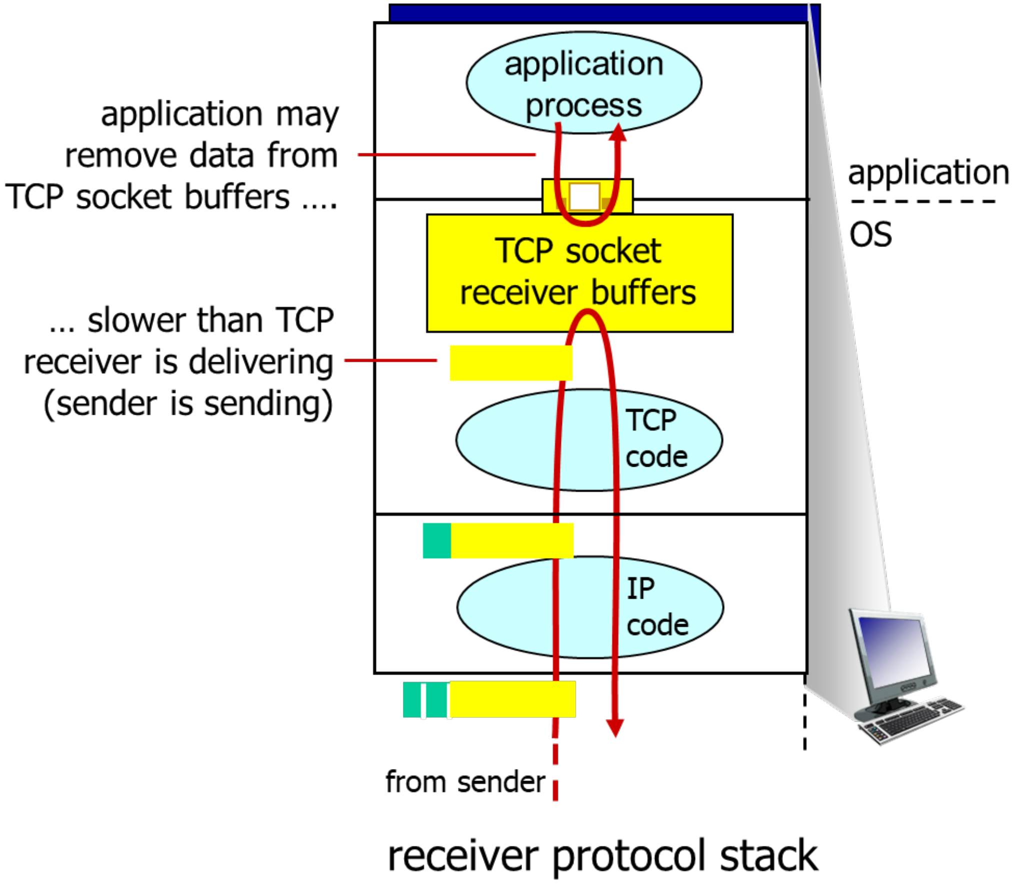


fast retransmit after sender receipt of triple duplicate ACK

TCP Flow Control (1 of 2)

flow control

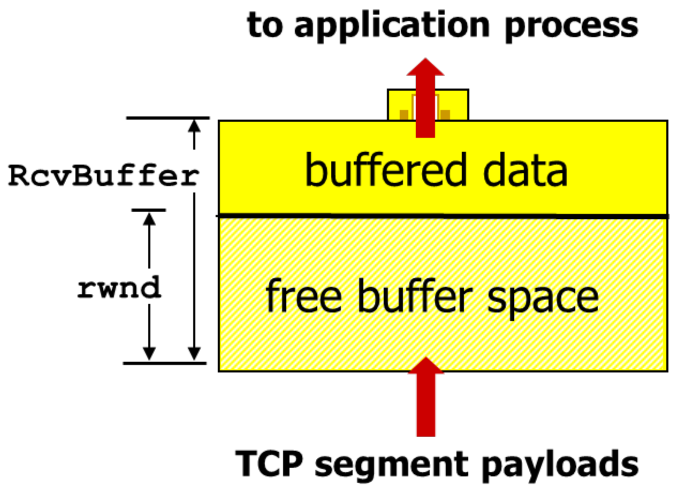
receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast



 COPYRIGHT @2017, 2013, 2010 PEARSONS EDUCATION INC.

TCP Flow Control (2 of 2)

- receiver "advertises" free buffer space by including **`rwnd`** value in TCP header of receiver-to-sender segments
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **`RcvBuffer`**
- sender limits amount of unacked ("in-flight") data to receiver's **`rwnd`** value
- guarantees receive buffer will not overflow



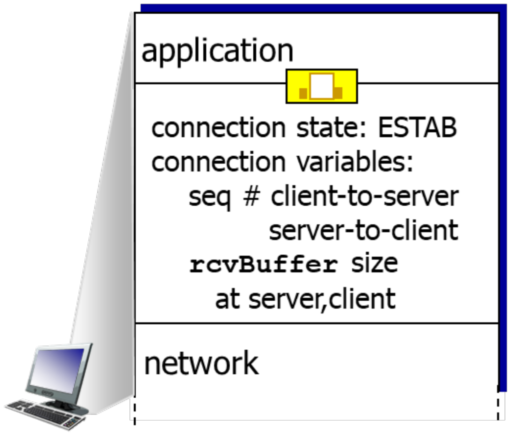
receiver-side buffering

Connection Management

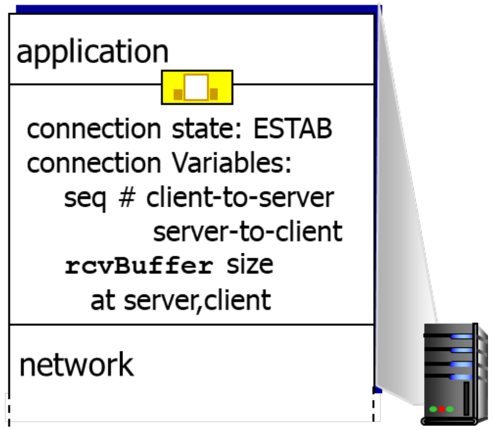
before exchanging data, sender/receiver “handshake”:

agree to establish connection (each knowing the other willing to establish connection)

agree on connection parameters




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

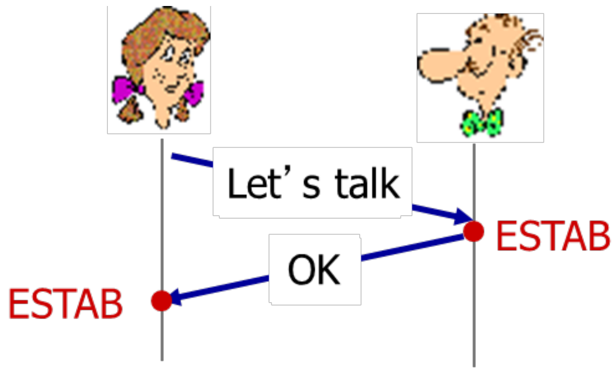


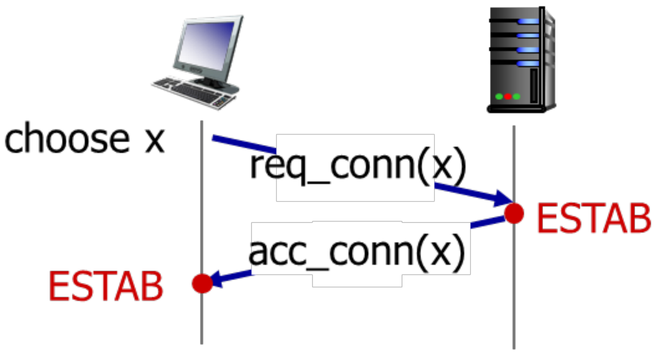

```
Socket connectionSocket =
  welcomeSocket.accept();
```

 COPYRIGHT @2017, 2013, 2010 PEARSONS EDUCATION INC.

Agreeing to Establish a Connection (1 of 2)

2-way handshake:



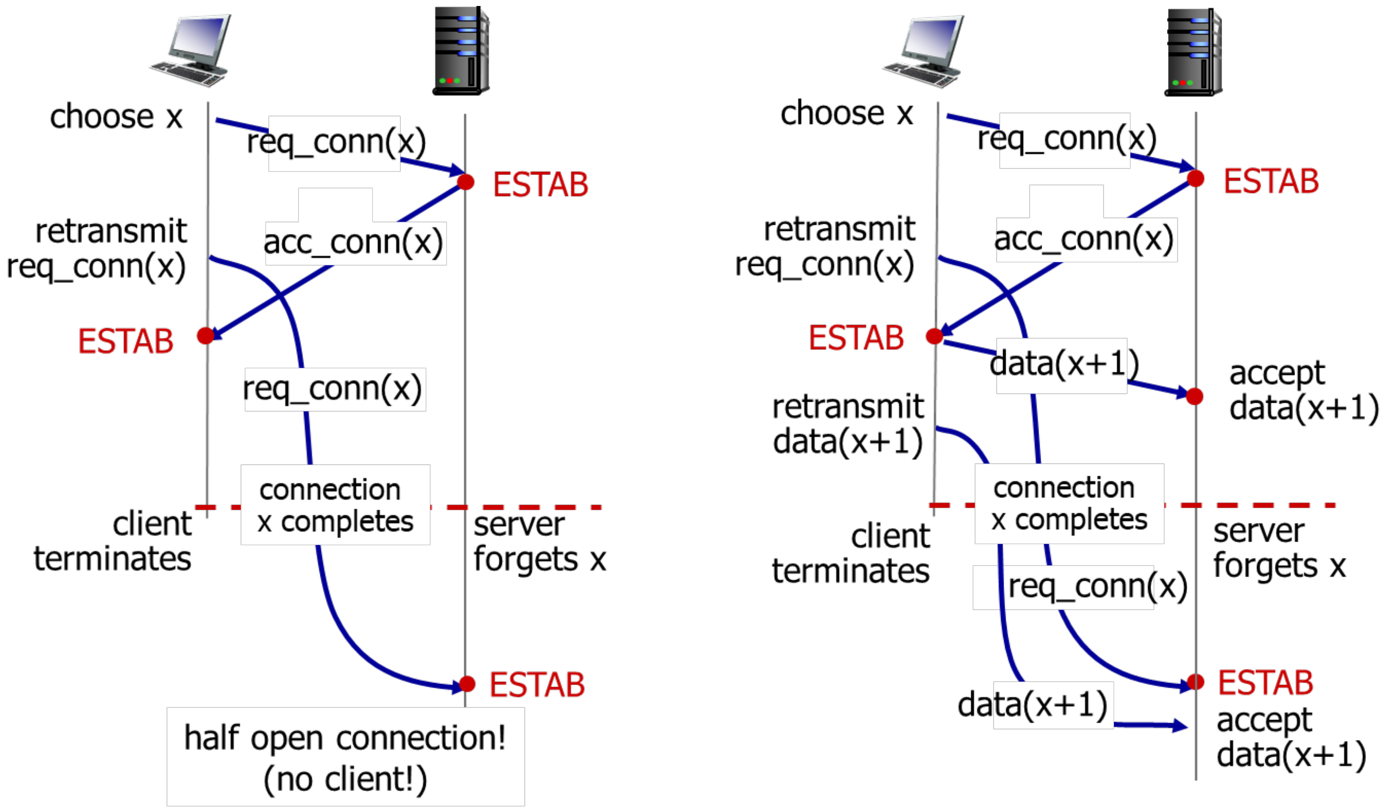


Q: will 2-way handshake always work in network?

- variable delays
- retransmitted messages (example. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

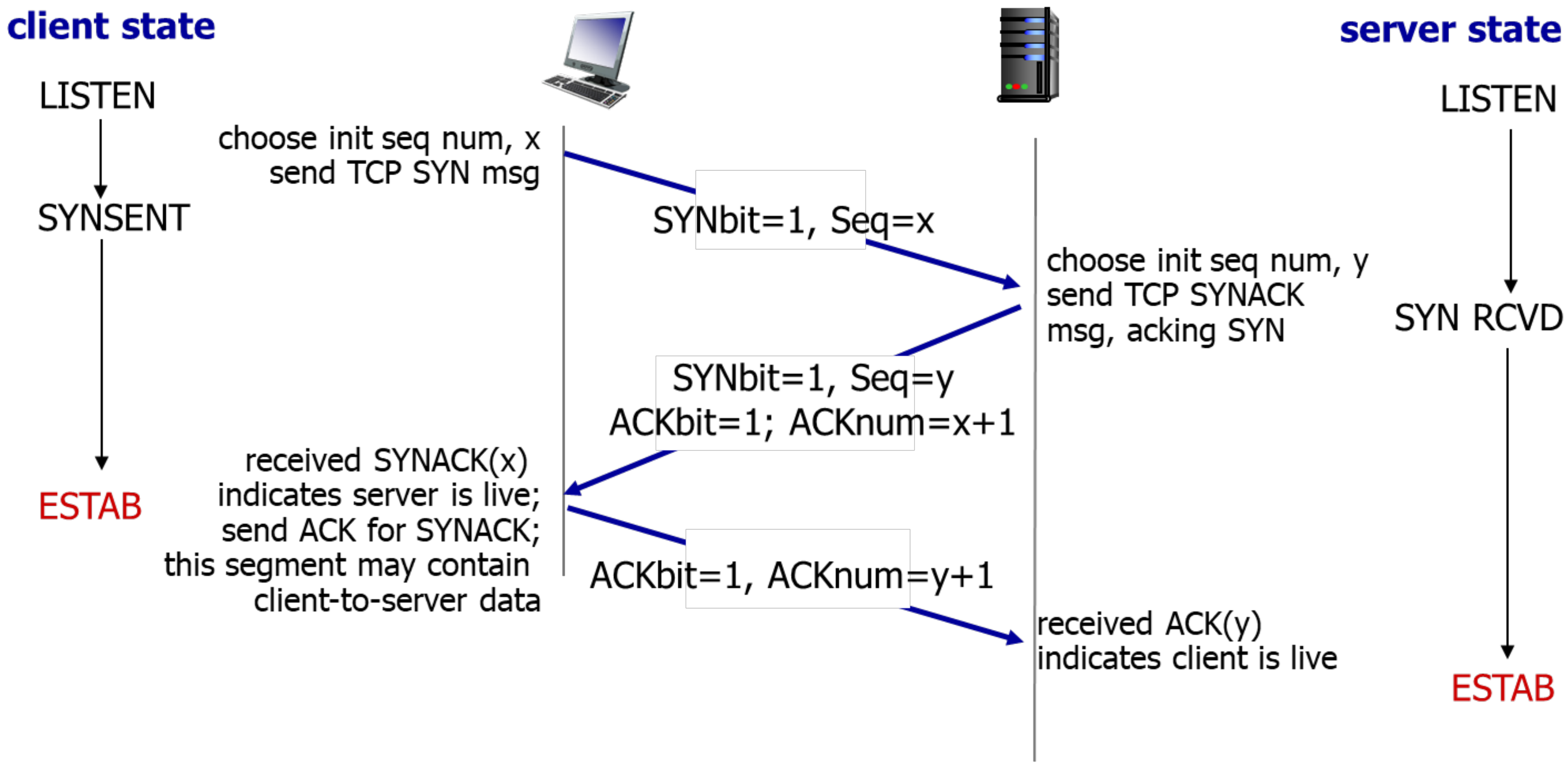
Agreeing to Establish a Connection (2 of 2)

2-way handshake failure scenarios:

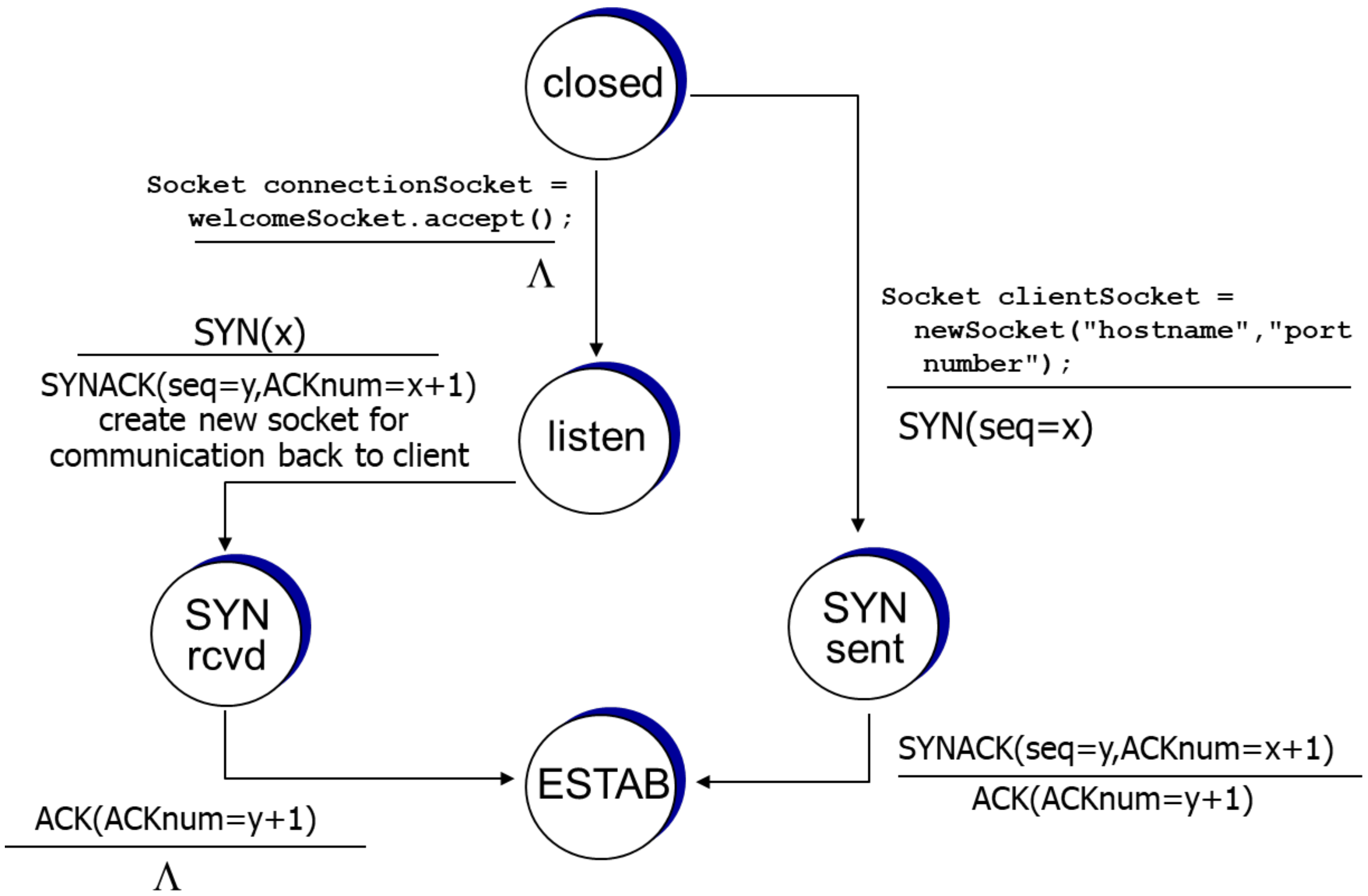


 COPYRIGHT @2017, 2013, 2010 PEARSONS EDUCATION INC.

TCP 3-Way Handshake



TCP 3-Way Handshake: FSM



TCP: Closing a Connection (1 of 2)

client, server each close their side of connection

- send TCP segment with FIN bit = 1

respond to received FIN with ACK

- on receiving FIN, ACK can be combined with own FIN

simultaneous FIN exchanges can be handled

TCP: Closing a Connection (2 of 2)