BITS Pilani
Pilani Campus

Computer Networks (CS F303)

Virendra Singh Shekhawat
Department of Computer Science and Information Systems

innovate achieve lead

BITS Pilani
Pilani Campus

Second Semester 2020-2021
Module-3 <Transport layer>
Lecture: 14

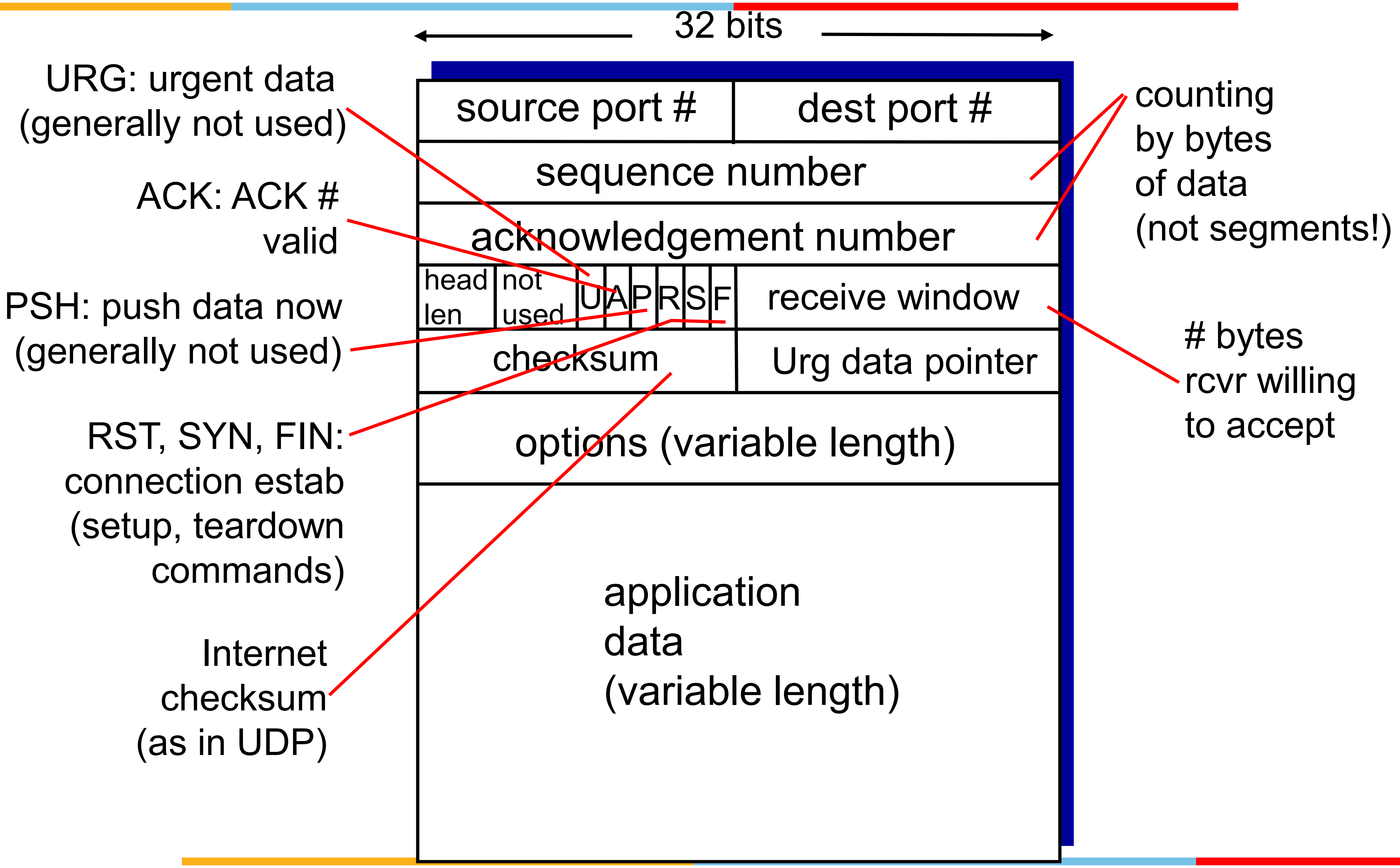
Topics

- Transport Layer
- TCP Protocol
  - Connection Establishment
  - TCP Segment Structure
  - Reliable data transfer
  - Flow control
  - Congestion control

TCP [RFCs: 793,1122,1223,2018,2581]

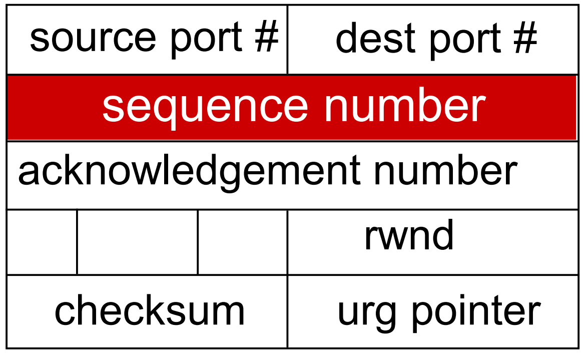
- Point to Point protocol
  - One sender and one receiver
- Reliable in-order byte stream
  - No message boundaries
- Pipelined
  - Window size is set by congestion and flow control
- Full duplex data
  - Bi-directional data flow in same connection
- Connection oriented
  - Handshaking (exchange of control msgs)
- Flow controlled
  - Sender do not overwhelm receiver

TCP Segment Structure

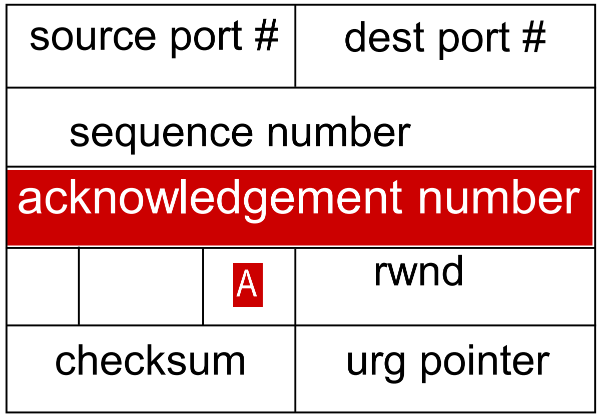

32 bits
URG: urgent data (generally not used)
ACK: ACK # valid
PSH: push data now (generally not used)
RST, SYN, FIN: connection estab (setup, teardown commands)
Internet checksum (as in UDP)
source port #
dest port #
sequence number
acknowledgement number
head len
not used
U
A
P
R
S
F
receive window
checksum
Urg data pointer
options (variable length)
application data (variable length)
counting by bytes of data (not segments!)
# bytes rcvr willing to accept

TCP Sequence Numbers and ACKs

outgoing segment from sender



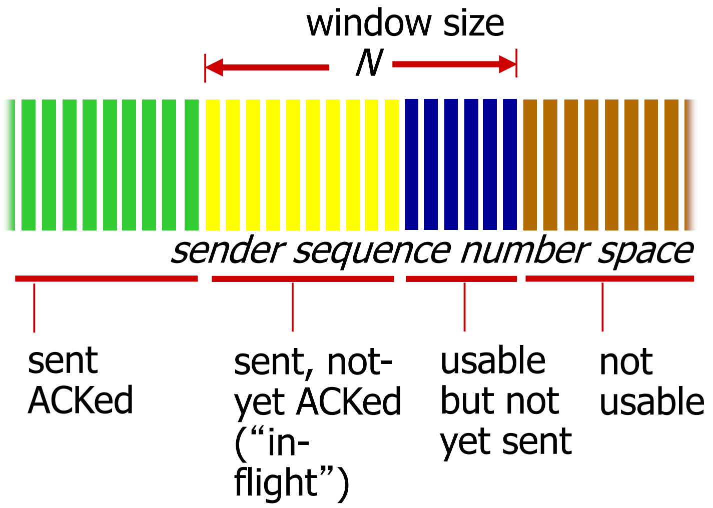
incoming segment to sender



TCP views data as stream of bytes

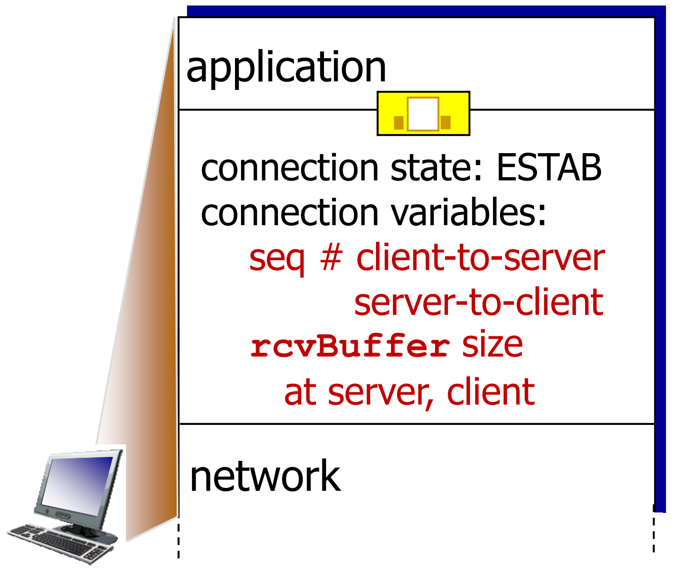
Sequence number reflects stream of transmitted bytes not segments

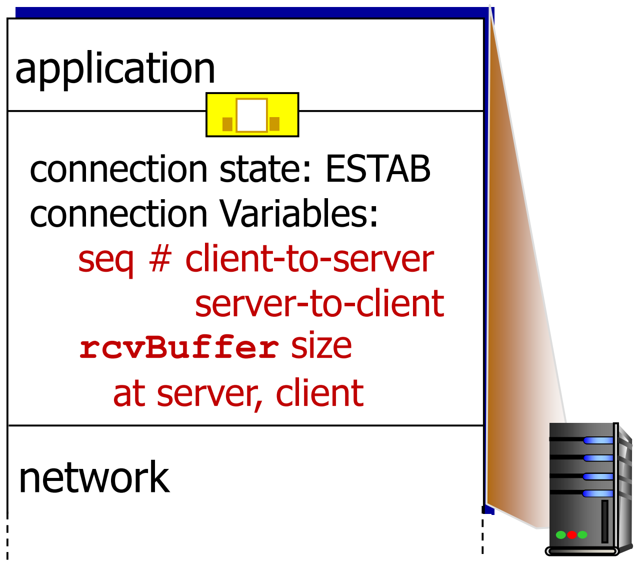
Sequence number of a segment – Byte stream number of the first byte in the segment


window size
N
sender sequence number space
sent ACKed
sent, not-yet ACKed ("in-flight")
usable but not yet sent
not usable

Connection Management

- Before exchanging data, sender/receiver do "handshake"
  - Agree on connection parameters



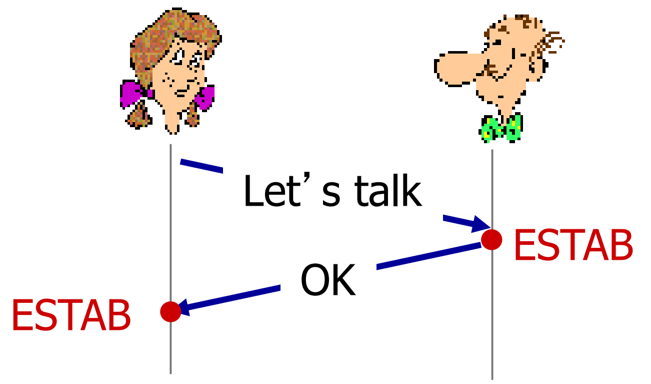


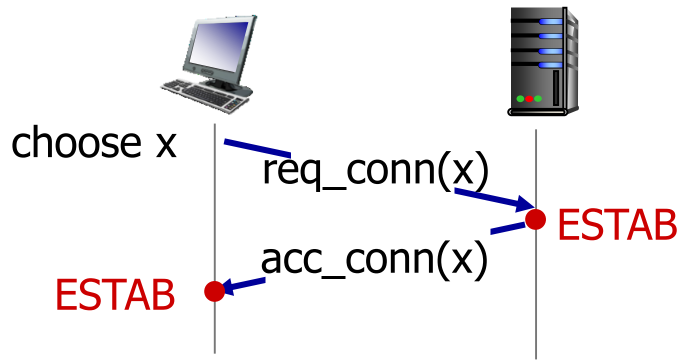
```
Socket clientSocket =
  newSocket("hostname","port
   number");
```

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

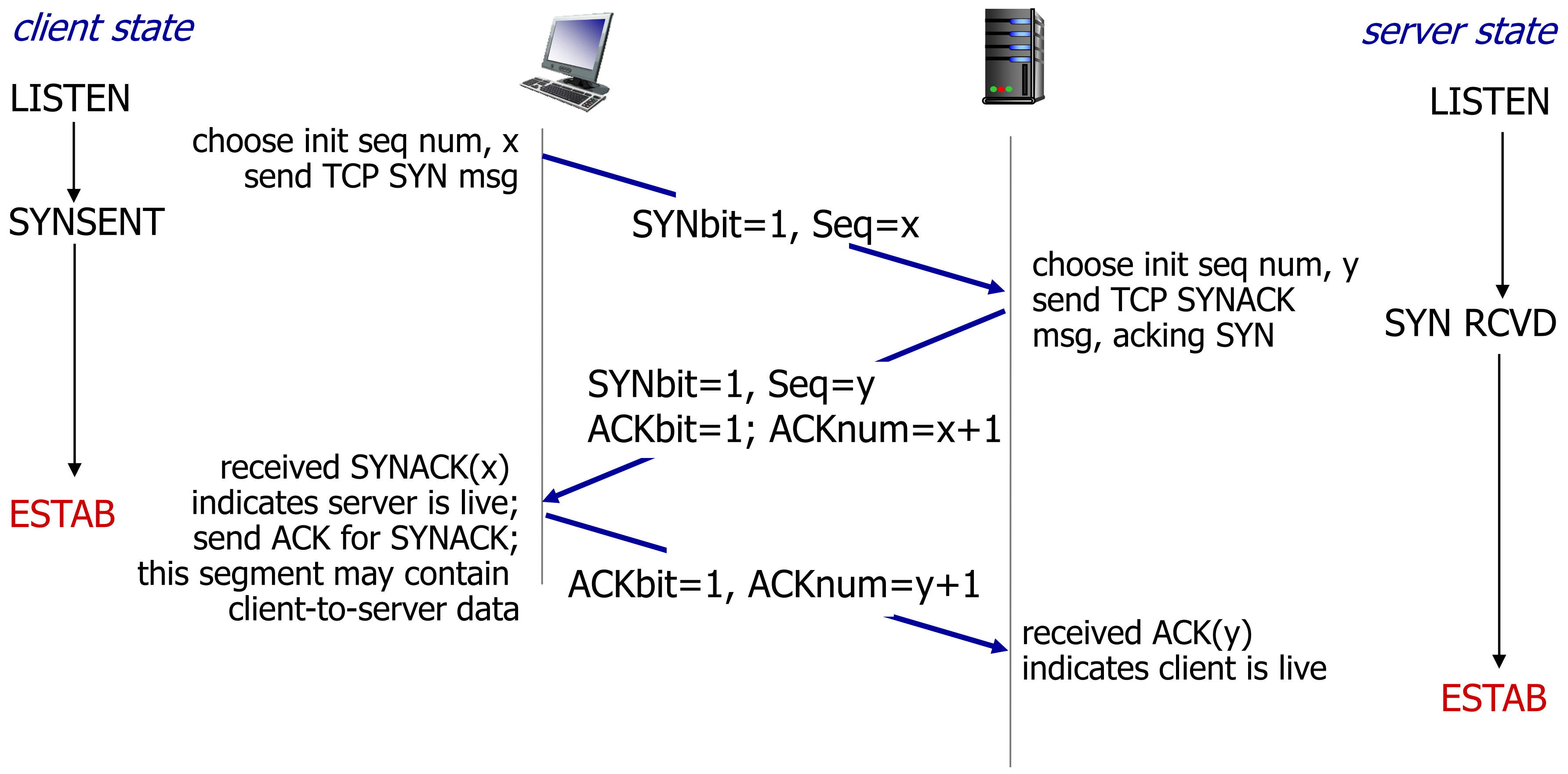
2-way Handshake

- Will 2-way handshake always work in network?

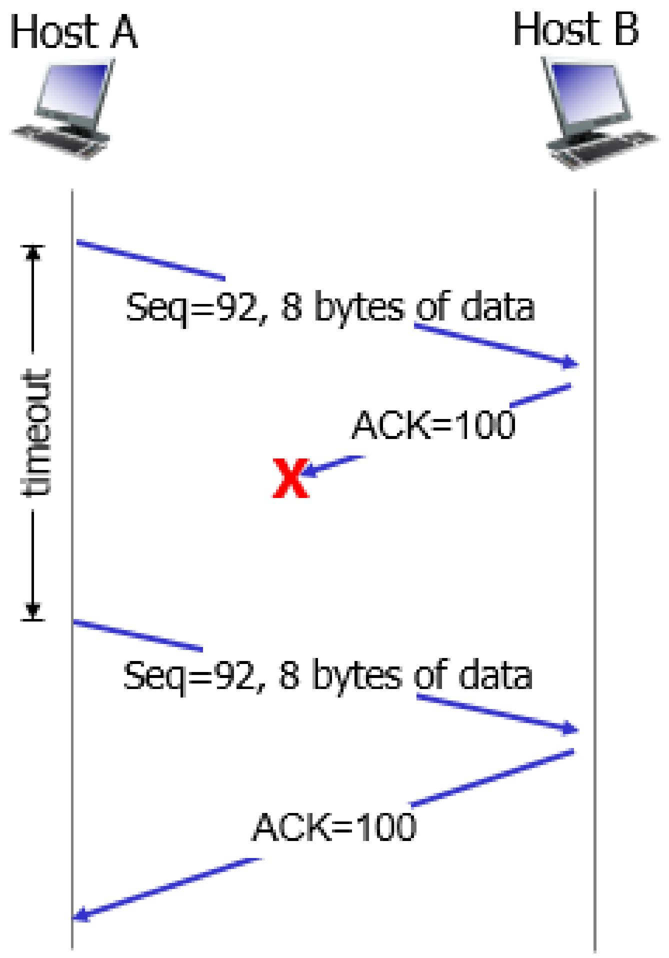




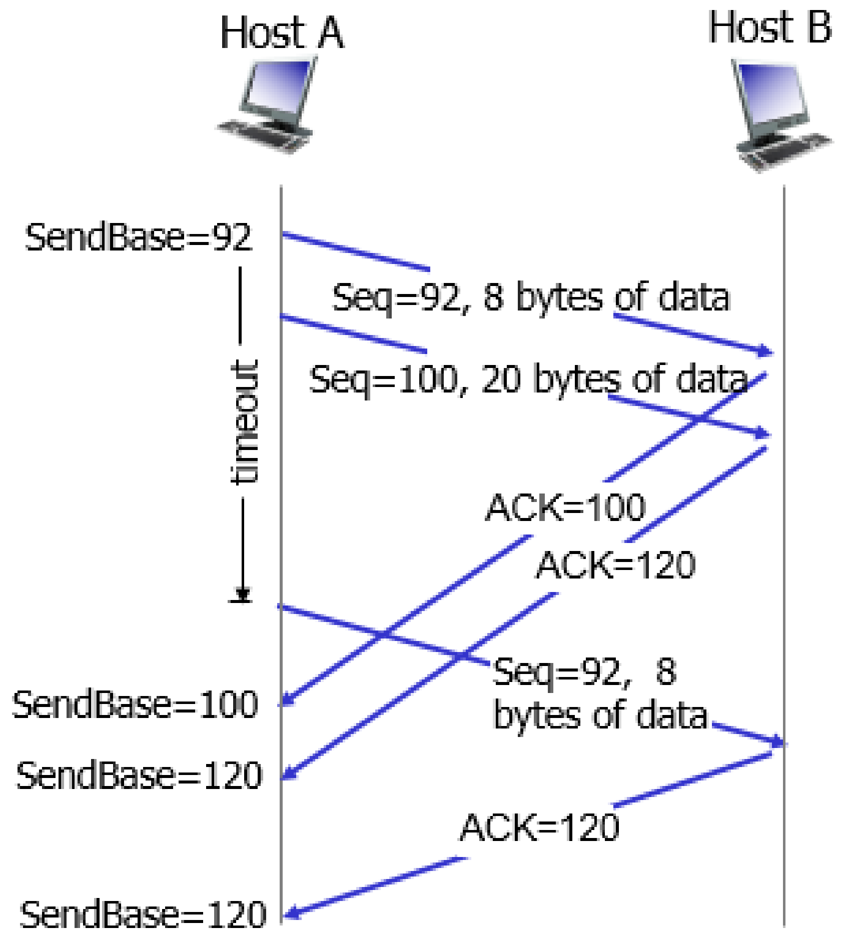
TCP 3-way Handshake



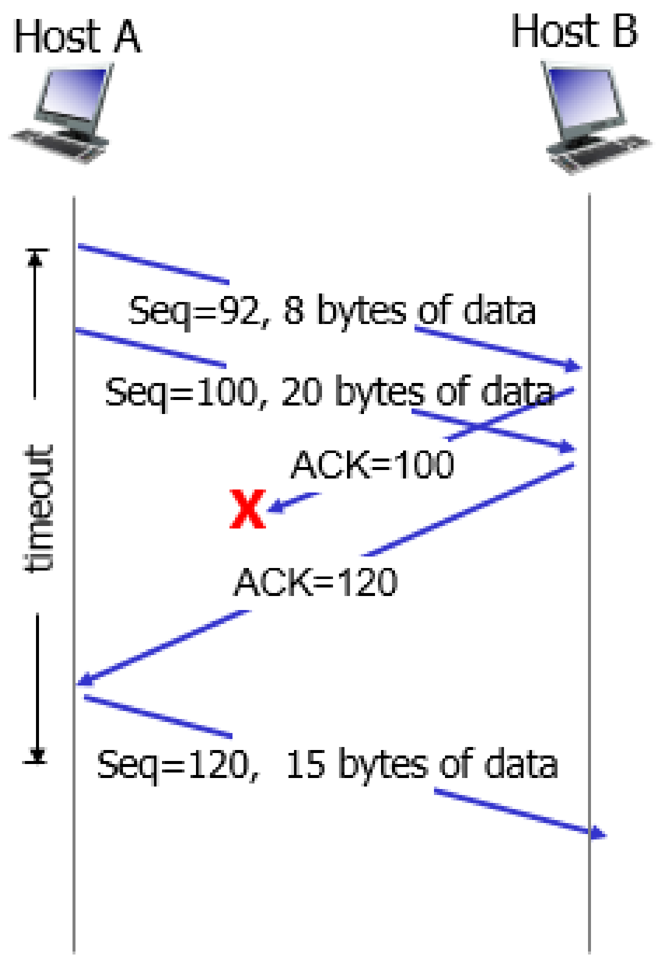
Lost ACK Scenario



Premature Timeout

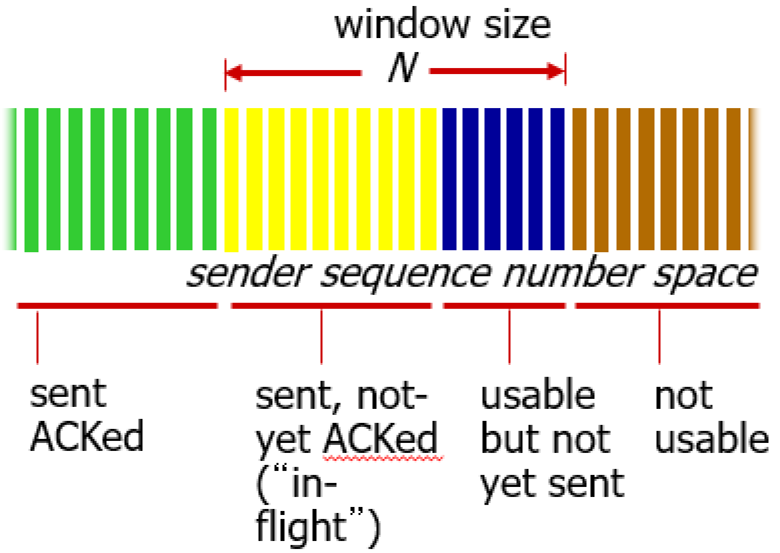


Cumulative ACK



TCP Sender Events and Actions

- Data received from app
  - Create segment with **seq #**
  - **seq #** is byte-stream number of first data byte in segment
  - Start timer if not already running
- Timeout
  - Retransmit segment that caused timeout
  - Restart timer
- ACK received
  - If ack acknowledges previously unacked segments
    - Update what is known to be ACKed (Shift window)
    - Start timer if there are still unACKed segments



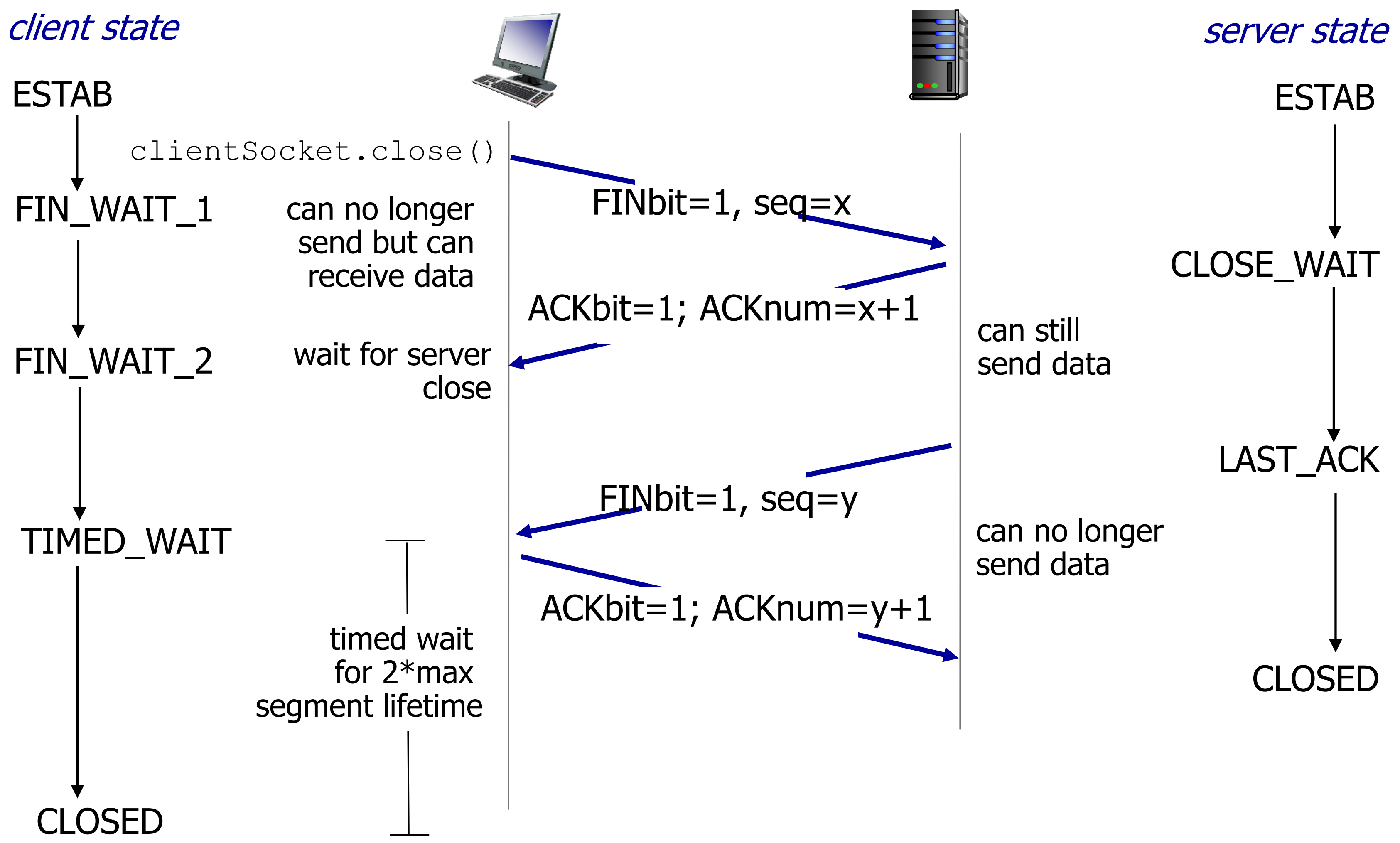
window size
N
sender sequence number space
sent ACKed
sent, not-yet ACKed ("in-flight")
usable but not yet sent
not usable

Is TCP GBN or SR…?

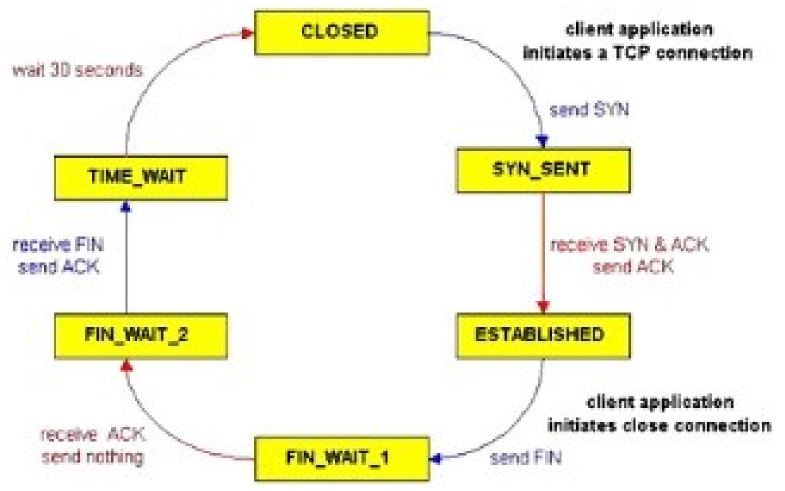
1. Is out of order segments are individually ACKed?
2. Are ACKs cumulative?
3. How many timers are maintained by sender?
4. Is TCP receiving out of order segments?

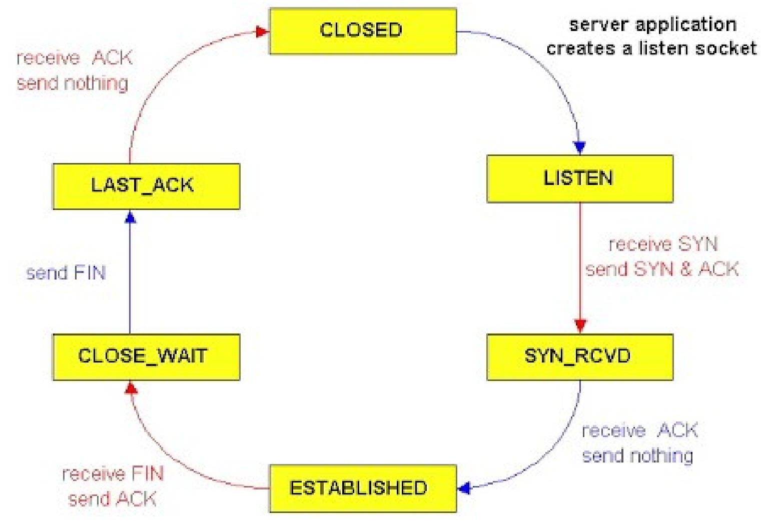
Example: TCP Reliable Data Transfer

TCP Connection Close



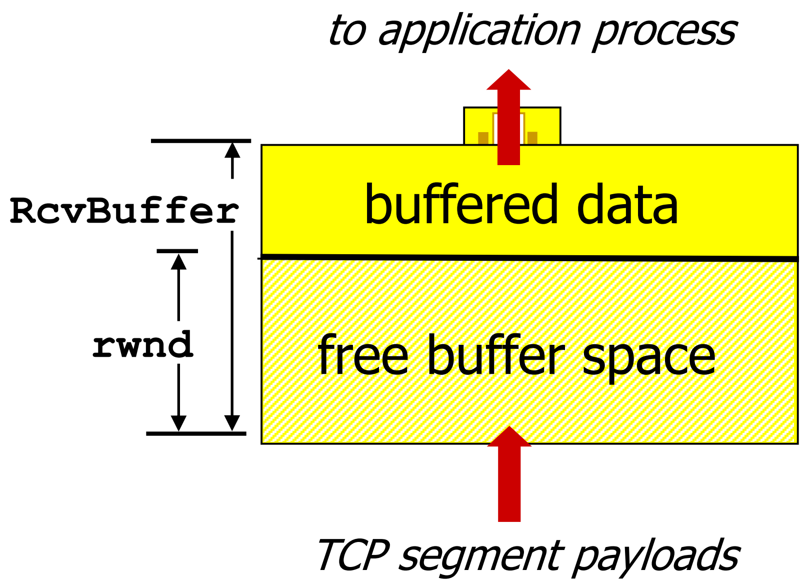
TCP Connection States





TCP Flow Control

- Receiver advertises free buffer space by including **`rwnd`** value in TCP header
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems auto adjust **`RcvBuffer`**
- Sender limits amount of unacked ("in-flight") data to receiver's **`rwnd`** value



TCP Timeout

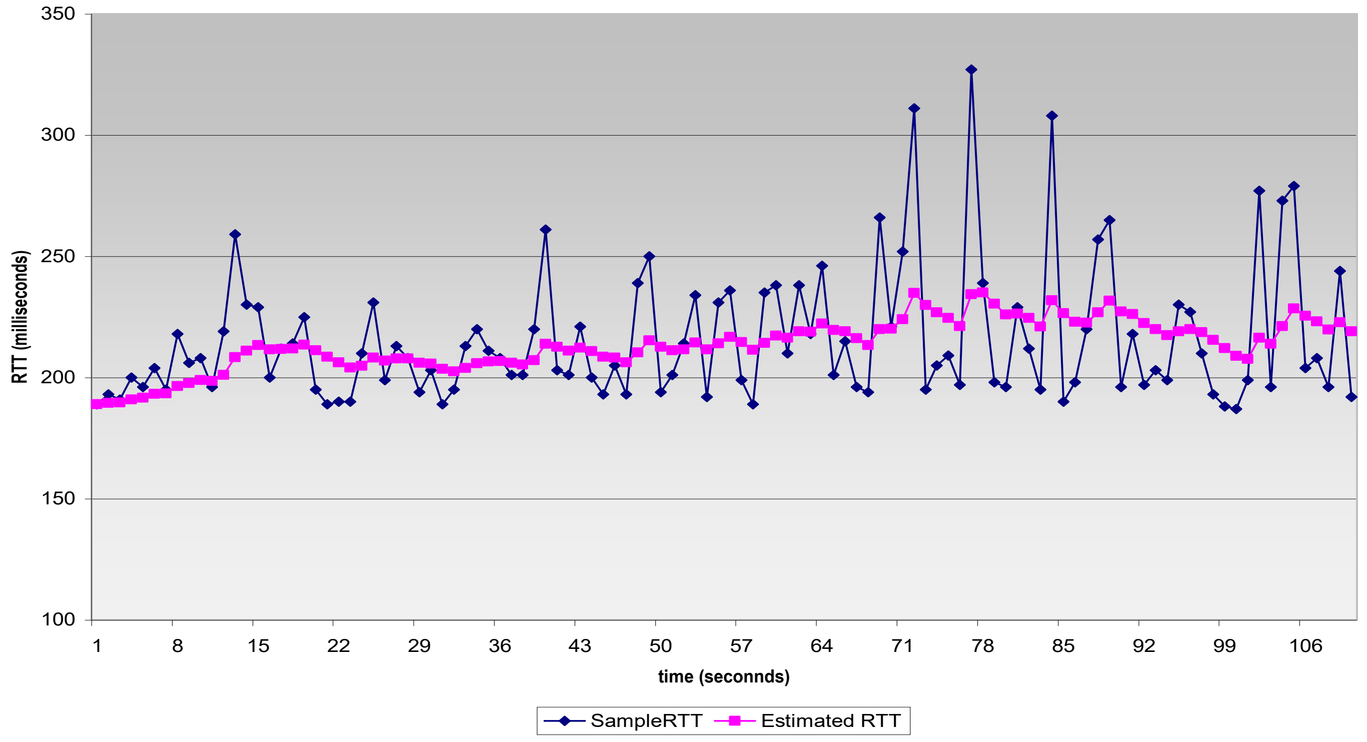
- How to set TCP Timeout value?
  - Must be longer than RTT
  - Too short vs. too long

- How to estimate RTT?
  - RTT: measured time from segment transmission until ACK receipt

RTT Estimation

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

- Influence of past sample decreases exponentially fast
- Typical value of α = 0.125



Timeout Interval

- Timeout Interval
  - **Estimated RTT** + “Safety margin”
  - Large variation in Estimated RTT → large safety margin

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

$$\texttt{TimeoutInterval} = \underbrace{\texttt{EstimatedRTT}}_{\text{estimated RTT}} + \underbrace{\texttt{4*DevRTT}}_{\text{“safety margin”}}$$

Example: Timeout Interval

- Consider three RTT samples (in ms): 150, 200 and 210 in that order. Assume initial estimated RTT= 200 ms, initial DevRTT = 50 ms, β = 0.25 and α = 0.125

Thank You