TCP

Prof. Anja Feldmann, Ph.D.

(Based on slide deck of Computer Networking, 7th ed., Jim Kurose and Keith Ross.)

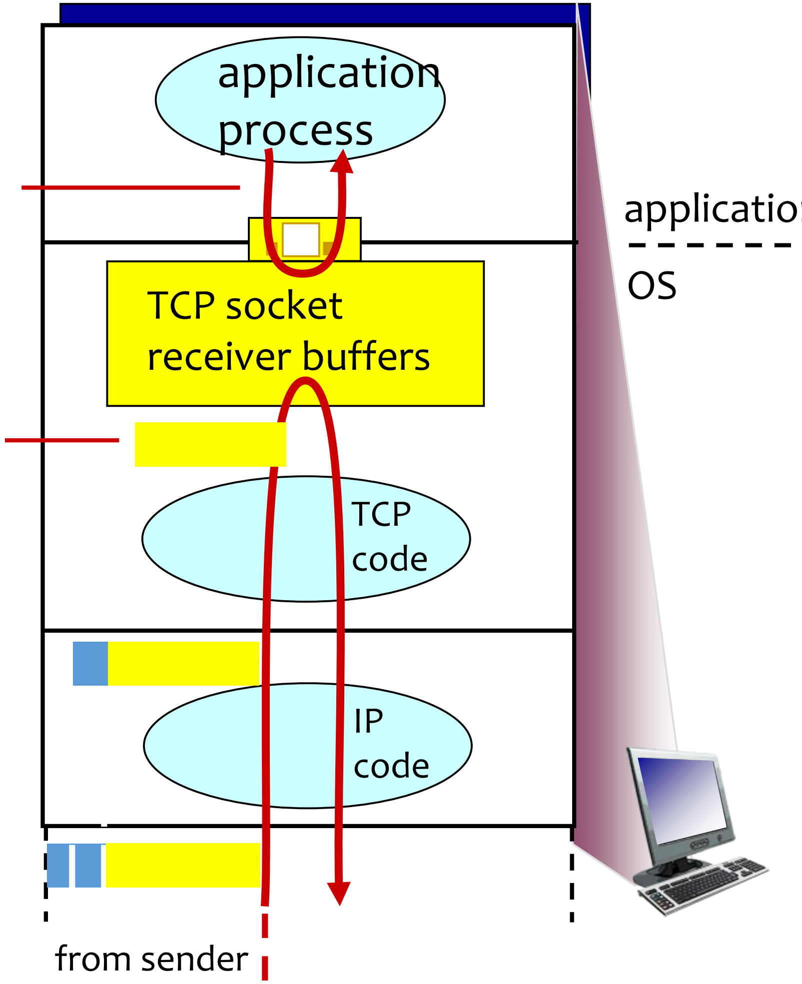
Outline

- *Connection-oriented* transport: TCP
  - Quick refresher on TCP *Segment structure*
    - Sequence numbers & Acknowledgements
  - Reliable data transfer
  - Flow control
  - Connection management
- Up next: Congestion control

TCP Flow Control

application may remove data from TCP socket buffers ….

… slower than TCP receiver is delivering (sender is sending)



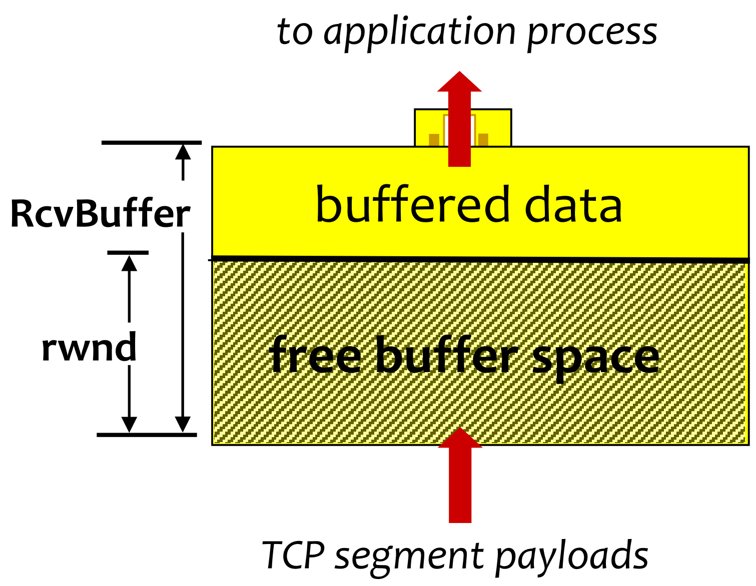
receiver protocol stack

Flow control

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

TCP Flow Control

- 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 auto-adjust *RcvBuffer*
- Sender limits amount of unacked ("in-flight") data to receiver's **rwnd** value
- Guarantees receive buffer will not overflow

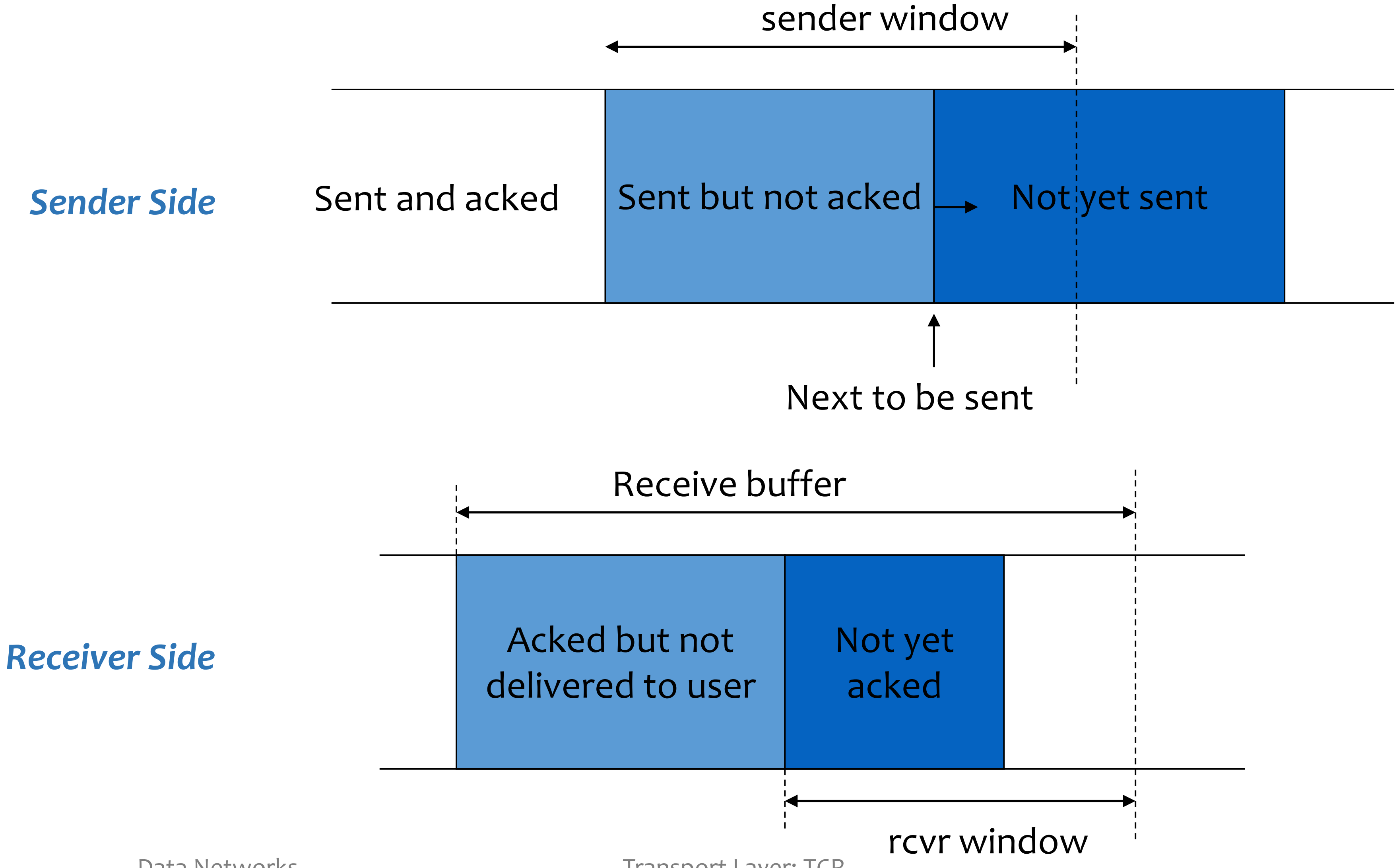


Receiver-side buffering

TCP Flow Control

- TCP is a sliding window protocol
  - For window size n, can send up to n bytes without receiving an acknowledgement
  - When the data is acknowledged, the window slides forward
- Original TCP always sent entire window
  - *Congestion control* now limits this via congestion window determined by the sender! (*network limited*)
  - If not, data rate is *receiver limited*
- Silly window syndrome
  - Too many small packets in flight
  - Limit the # of smaller packets than ***MSS*** to one per RTT

Window Flow Control



Ideal Window Size

Ideal size = delay * bandwidth (bw)

- ***Bandwidth-delay product*** (***RTT * bottleneck bitrate***)

- Window size < delay*bw ⇨ wasted bandwidth

- Window size > delay*bw ⇨
  - Queuing at intermediate routers ⇨ increased RTT
  - Eventually packet loss

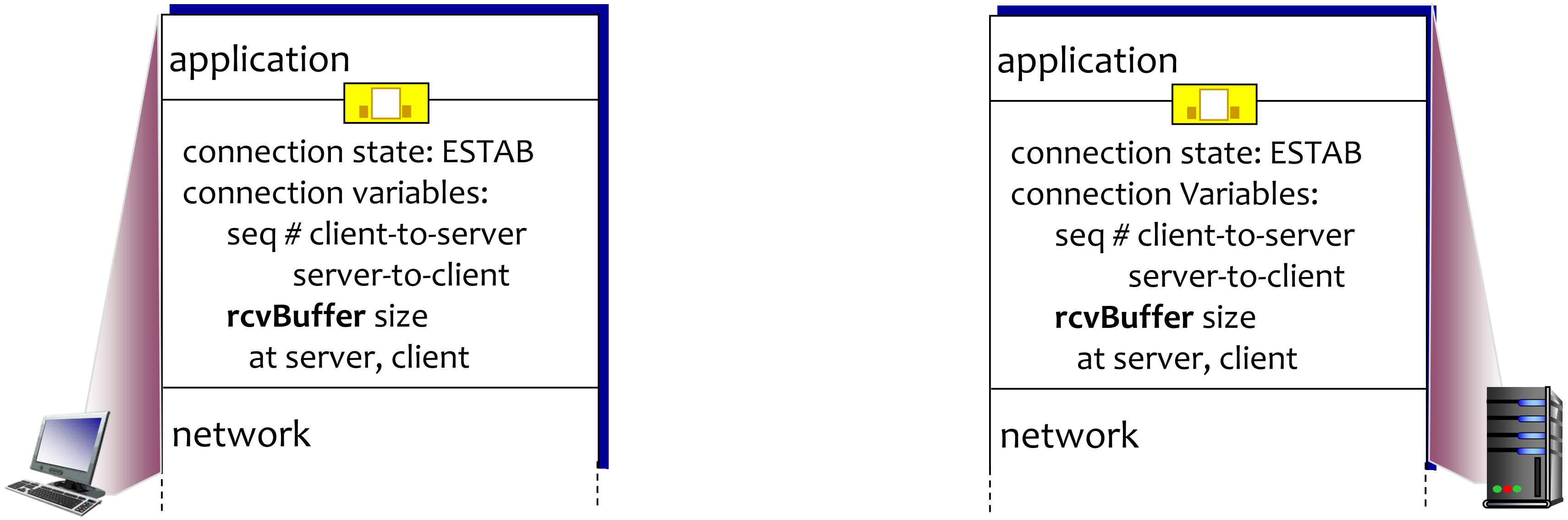
Outline

- *Connection-oriented* transport: TCP
  - Quick refresher on TCP *Segment structure*
    - Sequence numbers & Acknowledgements
  - Reliable data transfer
  - Flow control
  - Connection management
- Congestion control
  - Principles
  - Mechanism

Connection Management

Before exchanging data, sender/receiver "handshake":

- Agree to establish connection
  (each knowing the other willing to establish connection)
- Agree on connection parameters

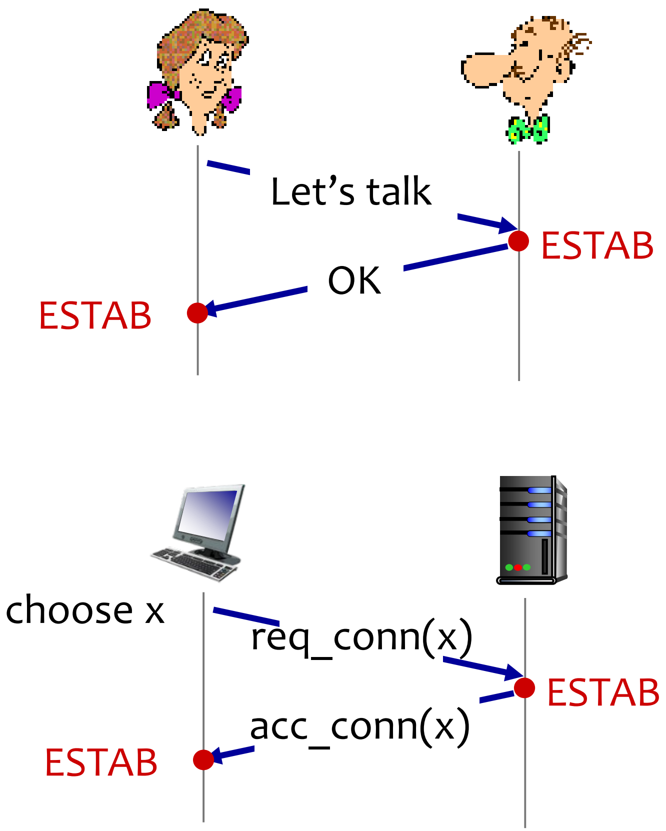



```
Socket clientSocket =
  newSocket("hostname",
             "port number");
```

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

Agreeing to establish a connection

2-way handshake:

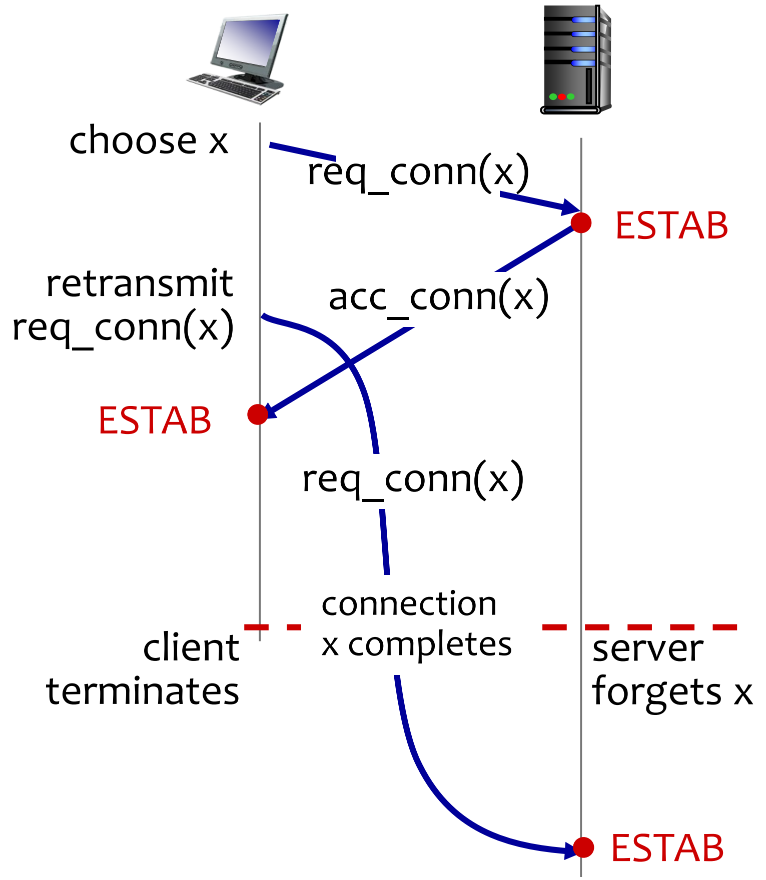


Will 2-way handshake always work in network?

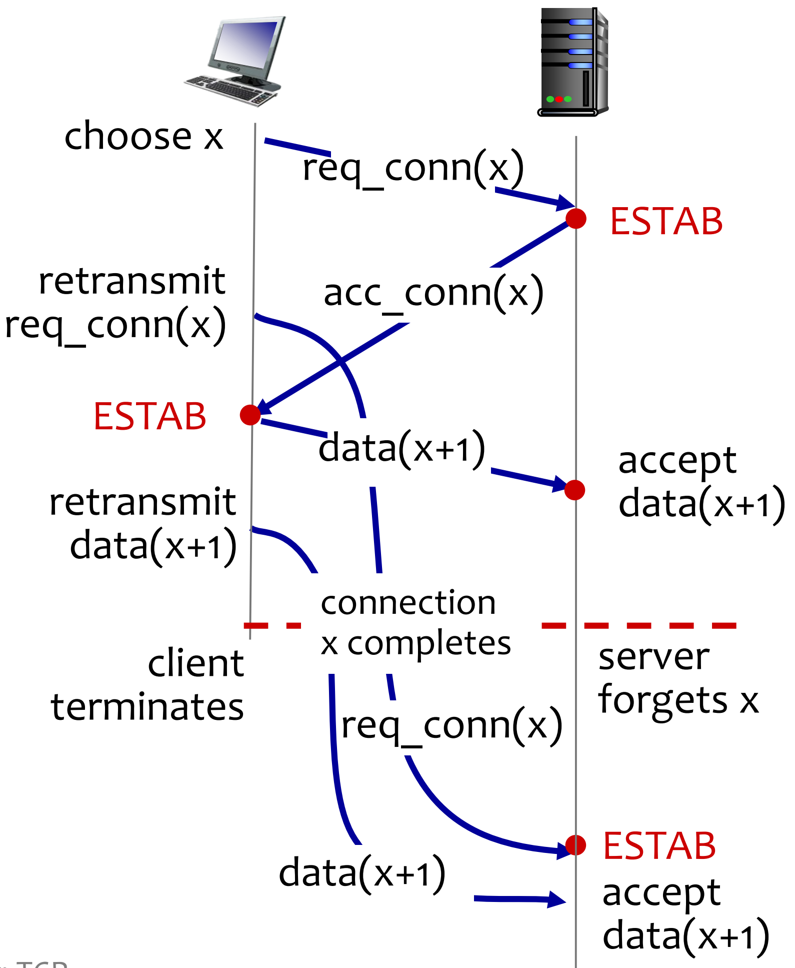
- Variable delays
- Retransmitted messages (e.g., req_conn(x)) due to message loss
- Message reordering
- Can't "see" other side

Agreeing to establish a connection

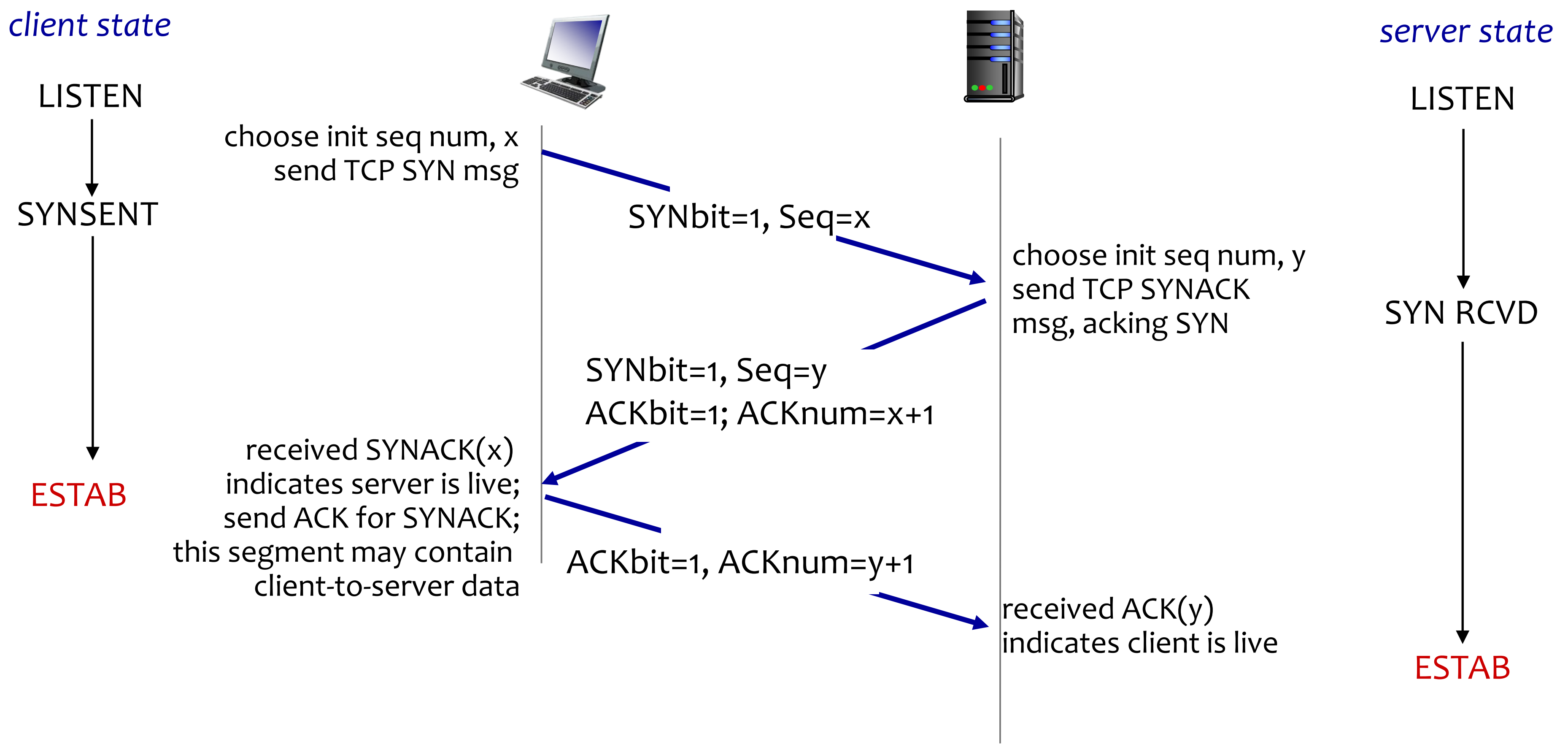
2-way handshake failure scenarios:



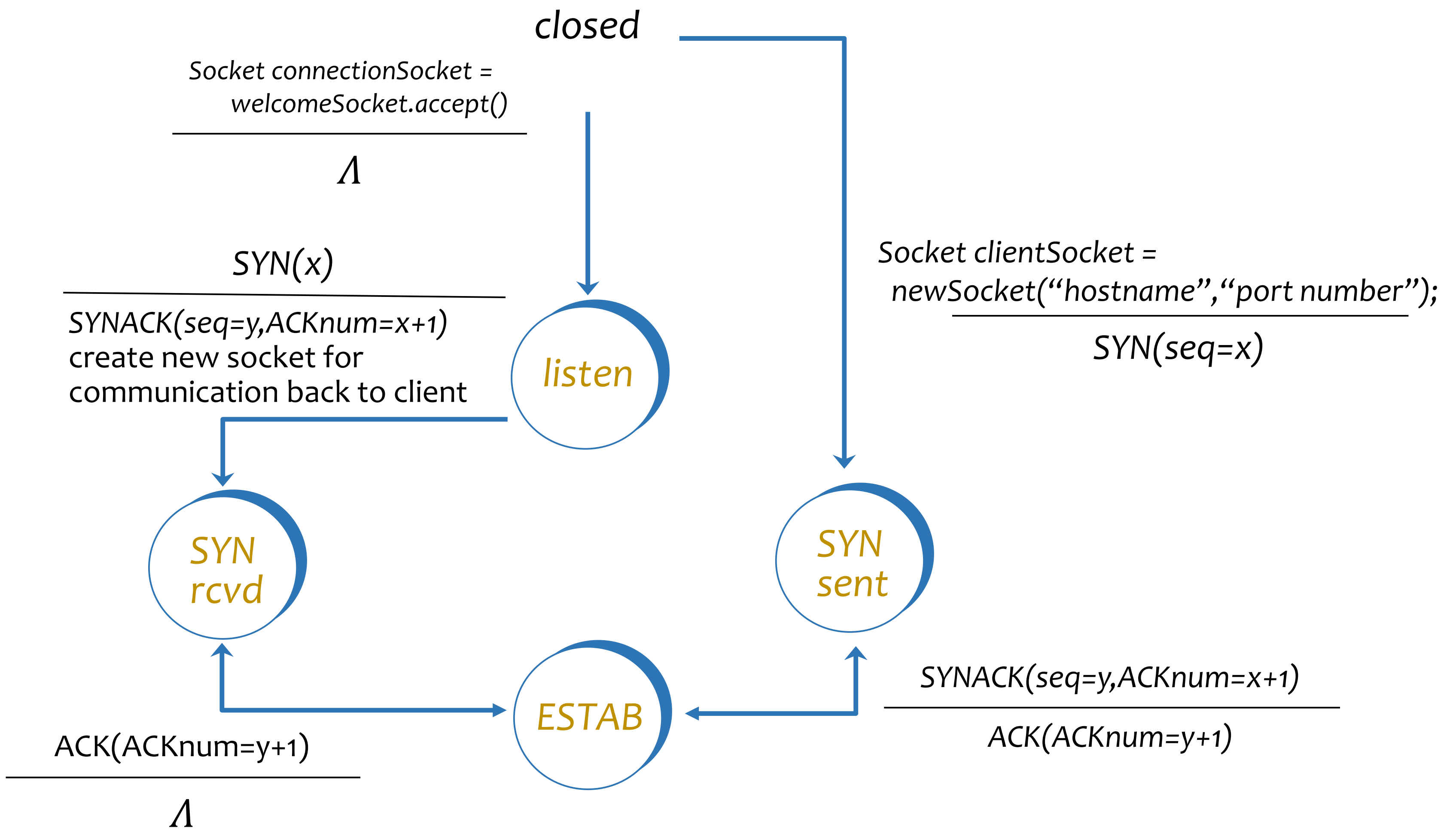
half open connection!
(no client!)



TCP 3-way Handshake



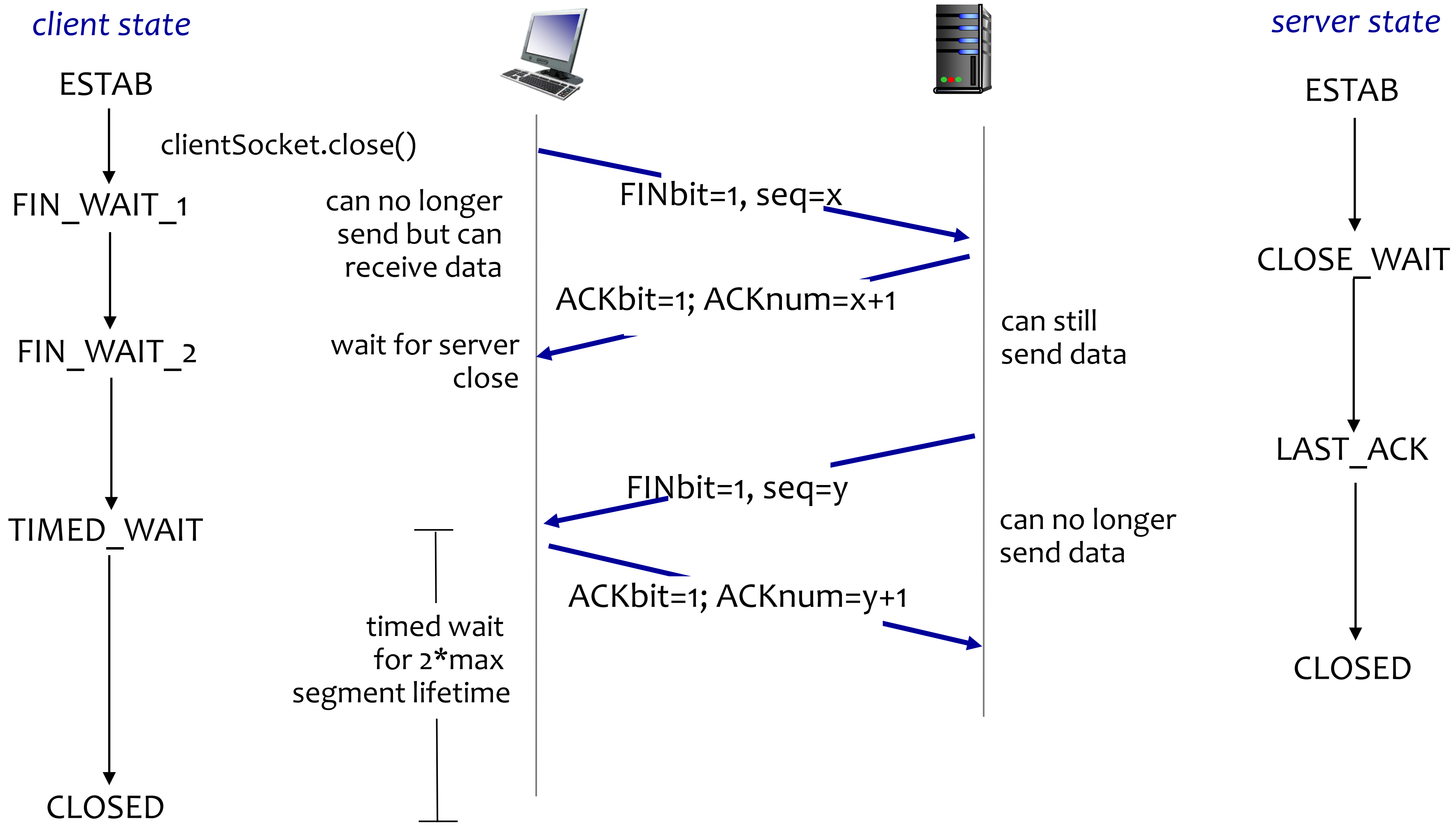
TCP 3-way Handshake: Finite State Machine



TCP: Closing a connection

- 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
- Error handling via RST!

TCP: Closing a connection



Outline

- *Connection-oriented* transport: TCP
  - Quick refresher on TCP *Segment structure*
    - Sequence numbers & Acknowledgements
  - Reliable data transfer
  - Flow control
  - Connection management
- Up next: Congestion control