CS 330: Network Applications & Protocols

Transport Layer

Galin Zhelezov
Department of Physical Sciences
York College of Pennsylvania

Overview of Transport Layer

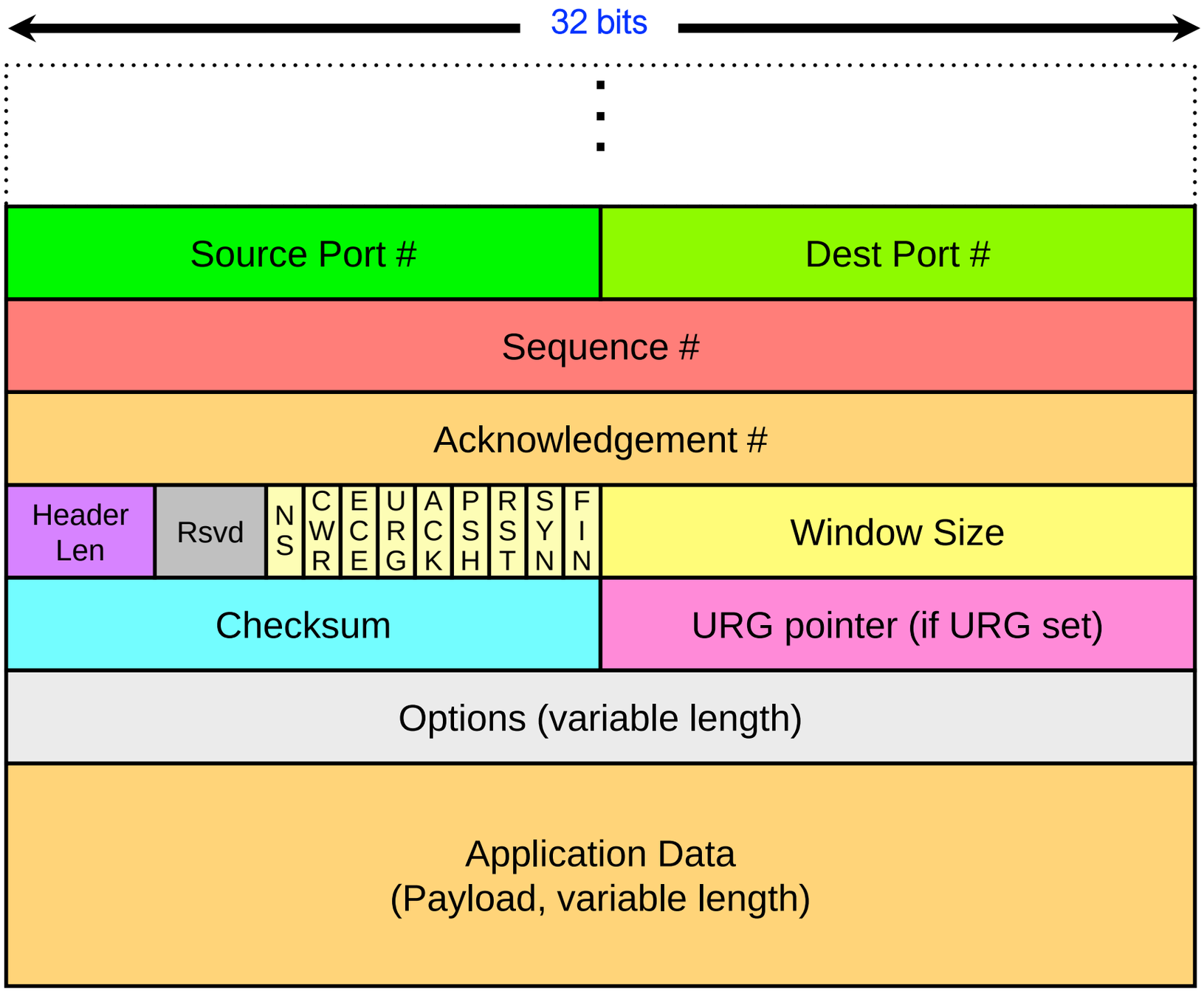
- **Transport-layer Services**
- **Multiplexing and Demultiplexing**
- **Connectionless Transport: UDP**
- **Principles of Reliable Data Transfer**
- **Connection-oriented Transport: TCP**
  - Segment Structure
  - Reliable Data Transfer
  - Flow Control
  - Connection Management
- **Principles of Congestion Control**
- **TCP Congestion Control**

TCP: Overview

- **A point-to-point protocol - one sender, one receiver**
- **Provides a reliable, in-order byte stream**
- **A pipelined protocol -- allows multiple in-flight packets**
  - TCP congestion and flow control set window size
- **Full duplex data - bi-directional data flow on same connection**
- **Connection-oriented:**
  - Handshaking (the exchange of control messages) initializes sender and receiver state before data exchange starts
- **Flow controlled:**
  - Sender will not overwhelm receiver

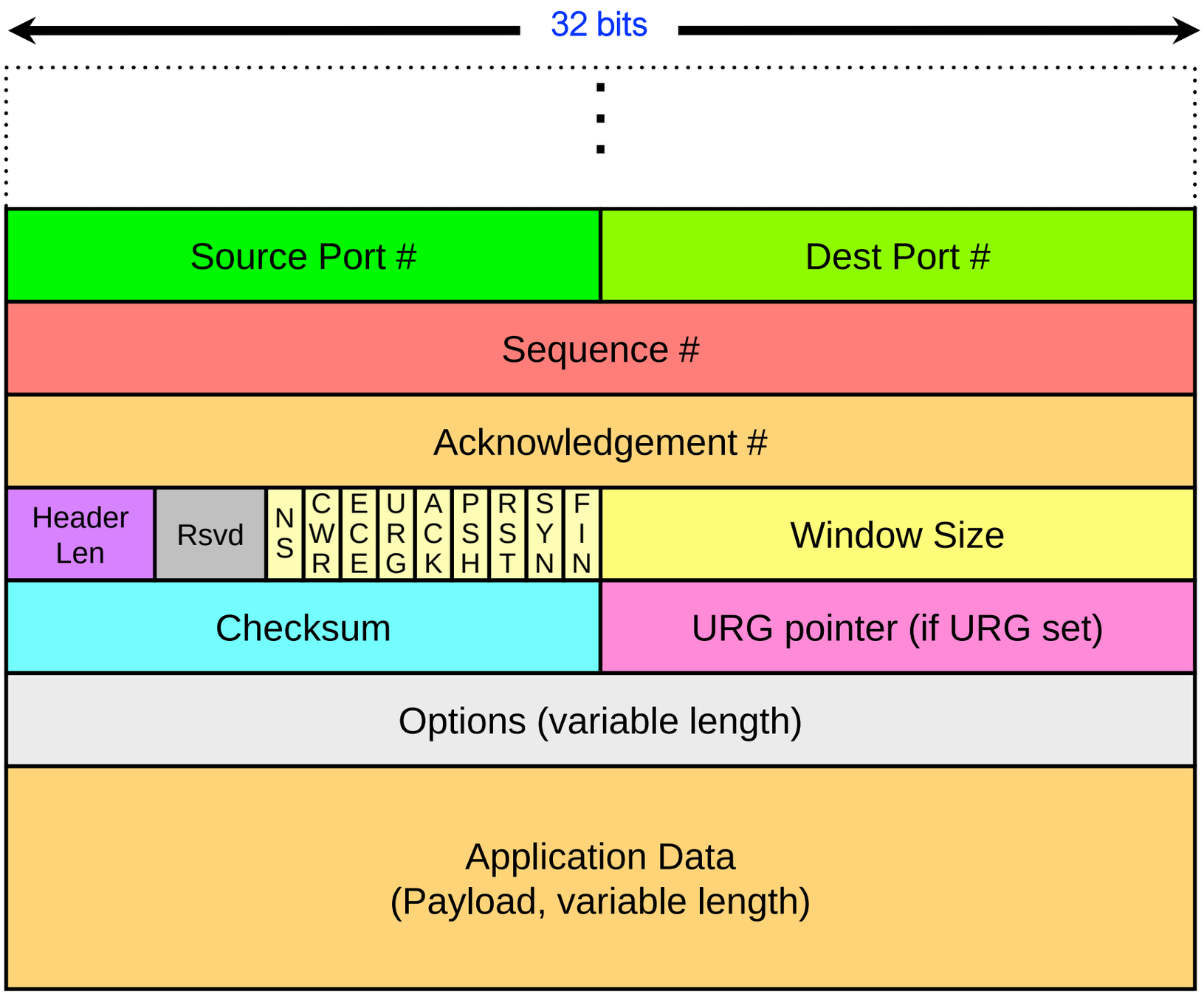
TCP Segment Structure

- **Source Port # - the port on the sender**
- **Dest Port # - the port on the receiver**
- **Sequence # - 32-bit number that represents the byte stream 'number' of the first byte in this segment's data**
  - e.g. if Sequence # is 10 and there are 7 bytes of data in this packet, then this segment contains bytes 10-16 of the data stream
- **Acknowledgement # - 32-bit number that represents the next sequence number that the receiver is expecting**



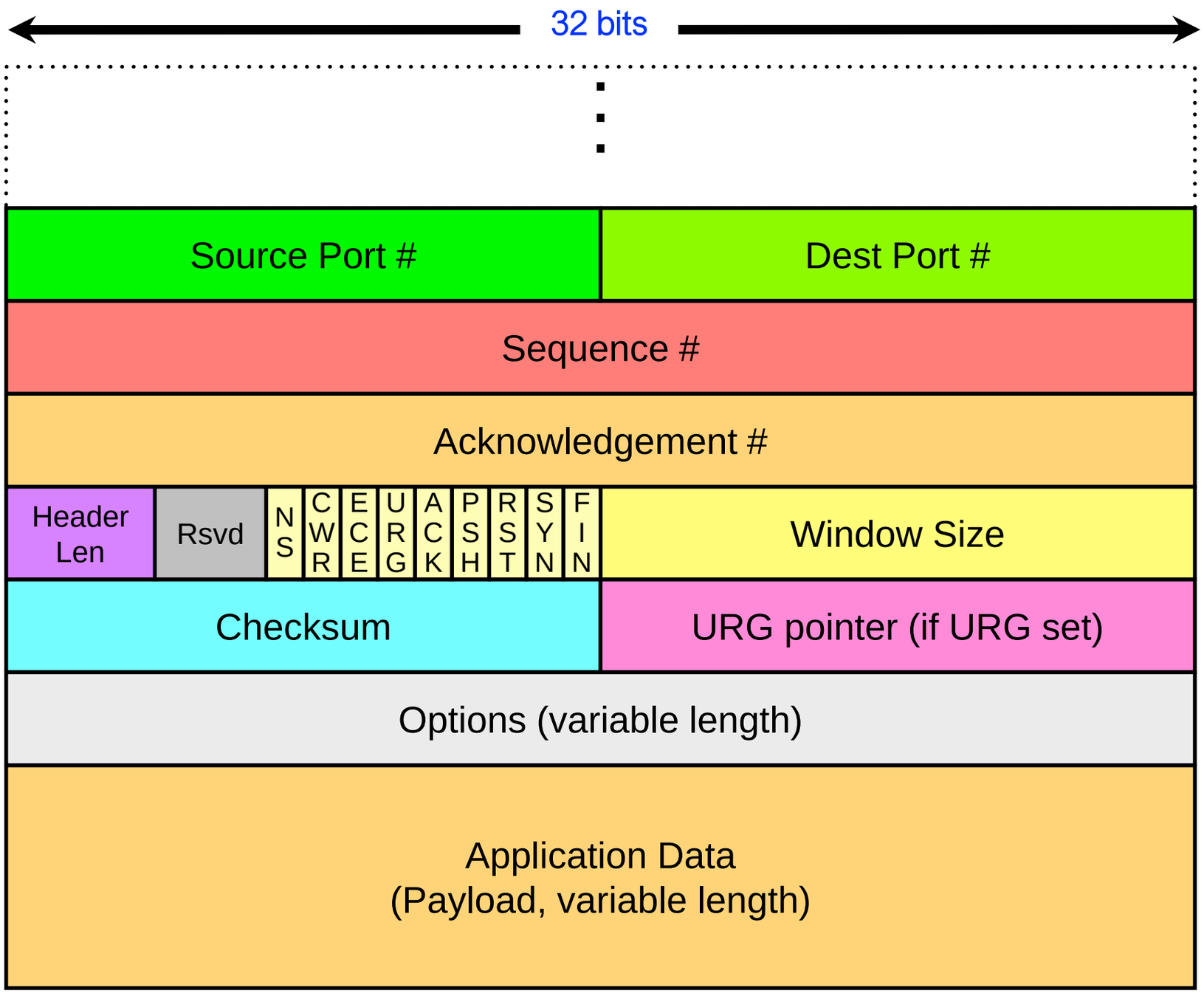
TCP Segment Structure (Cont.)

- **Header Length - 4-bit field that specifies the size of the TCP header in 32-bit words**
  - min = 5 ; max = 16
- **Reserved - 3-bits, not currently used**
- **Window Size - the size of the receive window that specifies the number of bytes that the receiver of this packet is currently willing to receive**
- **URG Pointer - specifies an offset of urgent data**
- **Checksum - 16-bit checksum computed over header and data (similar to UDP)**
- **Options - optional header fields**



TCP Segment Structure: Flags (1-bit each)

- **NS, CWR, & ECE - used in congestion mechanism**
- **URG - indicates that an URG pointer is present (not used much)**
- **ACK - indicates that the ACK # is significant**
- **PSH - request to push data to receiving application (not used much)**
- **RST - reset the connection**
- **SYN - synchronize sequence numbers. First packet from each end of connection should set this.**
- **FIN - indicates that there is no more data from the sender**



TCP Sequence Numbers/ACKs

Simple Telnet Scenario

Host A

Host B

User types ‘C’

Seq=42, ACK=79, data = ‘C’

Host ACKs receipt of ‘C’, Echoes back ‘C’

Seq=79, ACK=43, data = ‘C’

Host ACKs receipt of echoed ‘C’

Seq=43, ACK=80

Note that ACK messages can piggyback on data messages

TCP Round Trip Time & Timeout

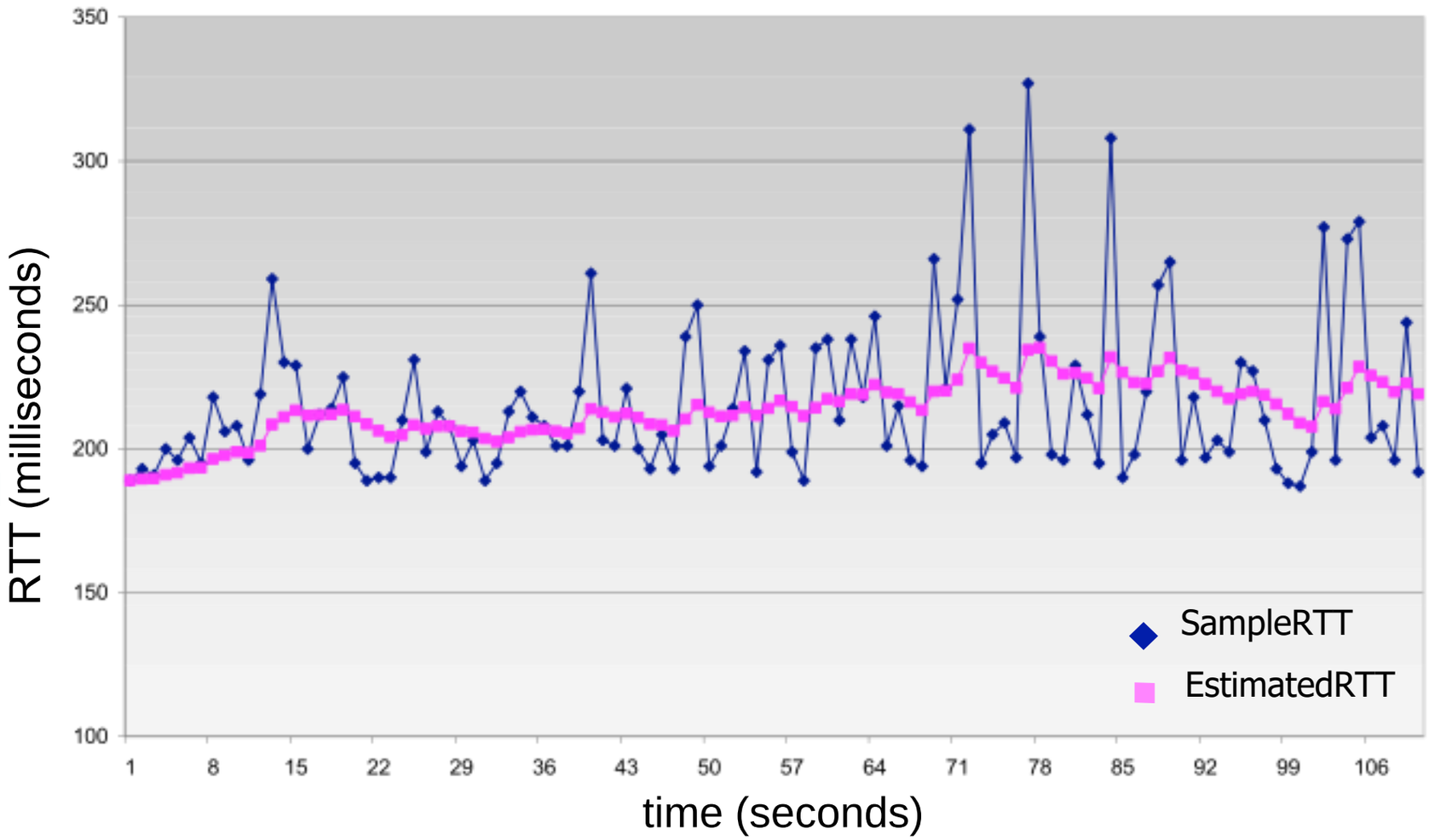
- **How should the TCP timeout value be set?**
  - Must be longer than RTT -- but RTT varies
  - Too short and timer will timeout prematurely causing unnecessary retransmissions of data
  - Too long and sender will have a slow reaction to segment loss

- **How can the RTT be estimated?**
  - Sample the RTT -- measure the time from segment transmission until ACK receipt (**`SampleRTT`**)
    - Ignore retransmissions
    - **`SampleRTT`** will vary, need to compute average over time to 'smooth' value

TCP Round Trip Time & Timeout

- **Compute `EstimatedRTT` from `SampleRTT`**
- **Exponentially weighted moving average**
- **Influence of past sample decreases exponentially fast**
- **Typical value: `α = 0.125`**
- **Use `EstimatedRTT` plus some safety-margin to determine timeout interval**

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



TCP round trip time, timeout

- **timeout interval: `EstimatedRTT` plus "safety margin"**
  - large variation in **`EstimatedRTT`** -> larger safety margin
- **estimate SampleRTT deviation from EstimatedRTT:**

$$\texttt{DevRTT} = (1-\beta)*\texttt{DevRTT} + \beta*|\texttt{SampleRTT}-\texttt{EstimatedRTT}|$$

(typically, $\beta = 0.25$)

$$\texttt{TimeoutInterval} = \texttt{EstimatedRTT} + 4*\texttt{DevRTT}$$

estimated RTT

"safety margin"

Overview of Transport Layer

- **Transport-layer Services**
- **Multiplexing and Demultiplexing**
- **Connectionless Transport: UDP**
- **Principles of Reliable Data Transfer**
- **Connection-oriented Transport: TCP**
  - Segment Structure
  - Reliable Data Transfer
  - Flow Control
  - Connection Management
- **Principles of Congestion Control**
- **TCP Congestion Control**

TCP Reliable Data Transfer

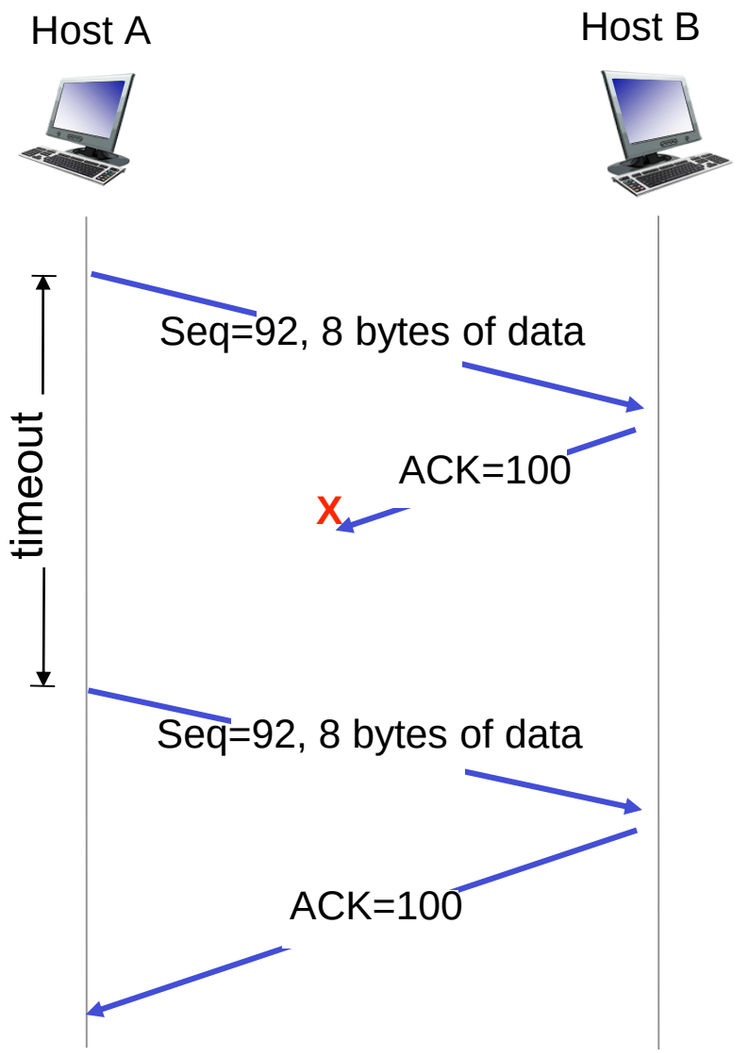
- **TCP creates reliable data transfer service on top of the IP protocol's unreliable service**
  - Send data as pipelined segments
  - Uses cumulative ACKs
  - Uses a single retransmission timer

- **Retransmissions are triggered by:**
  - Timeout events
  - Duplicate ACK messages

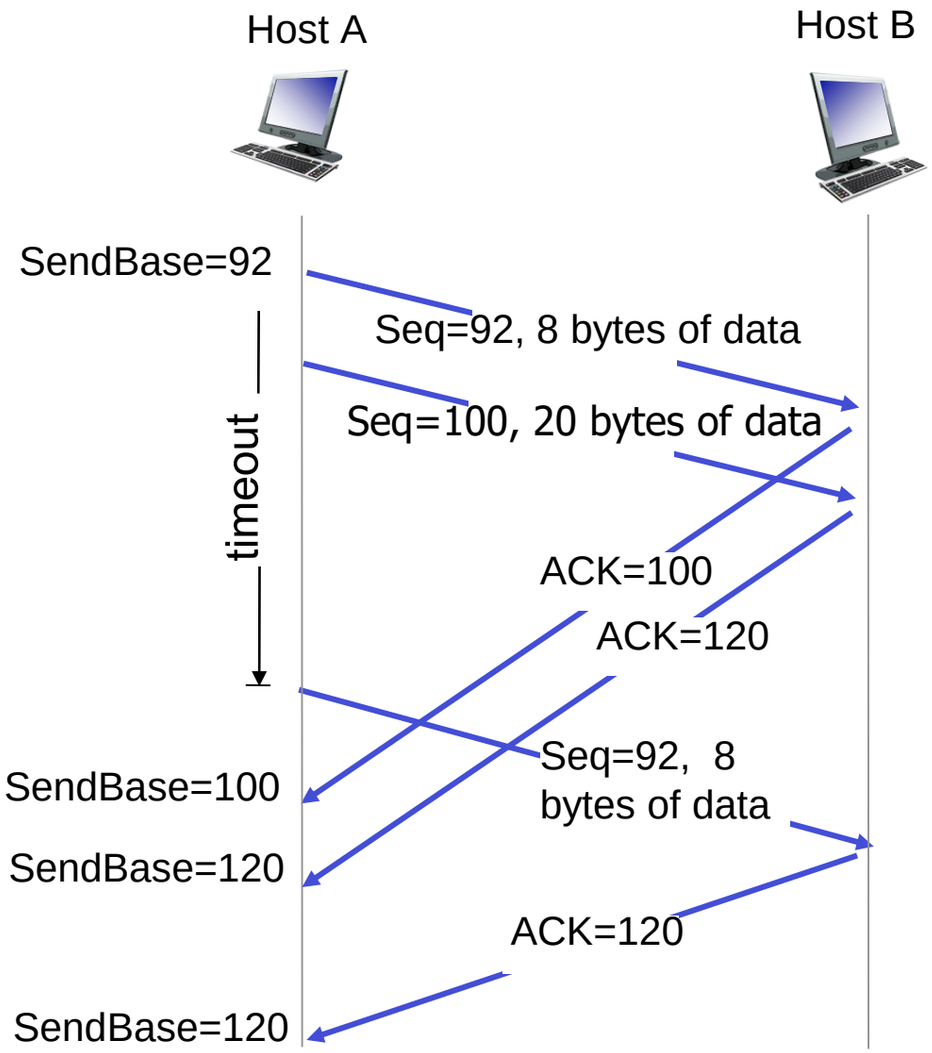
TCP Sender Events

- **Data received from application layer**
  - Create segment with sequence number
  - Sequence number is byte-stream number of the first data byte in segment
  - Start a timer if not already running
    - Timer is for oldest unacked segment
    - Use timeout interval computed from sampling RTT
- **If timeout occurs:**
  - Retransmit the segment that caused timeout
  - Restart the timer
- **When ACK is received:**
  - If ACK is for previously unacked segments
    - Update what is known to be ACKed
    - Restart timer if there are still unacked segments

TCP: Retransmission Scenarios

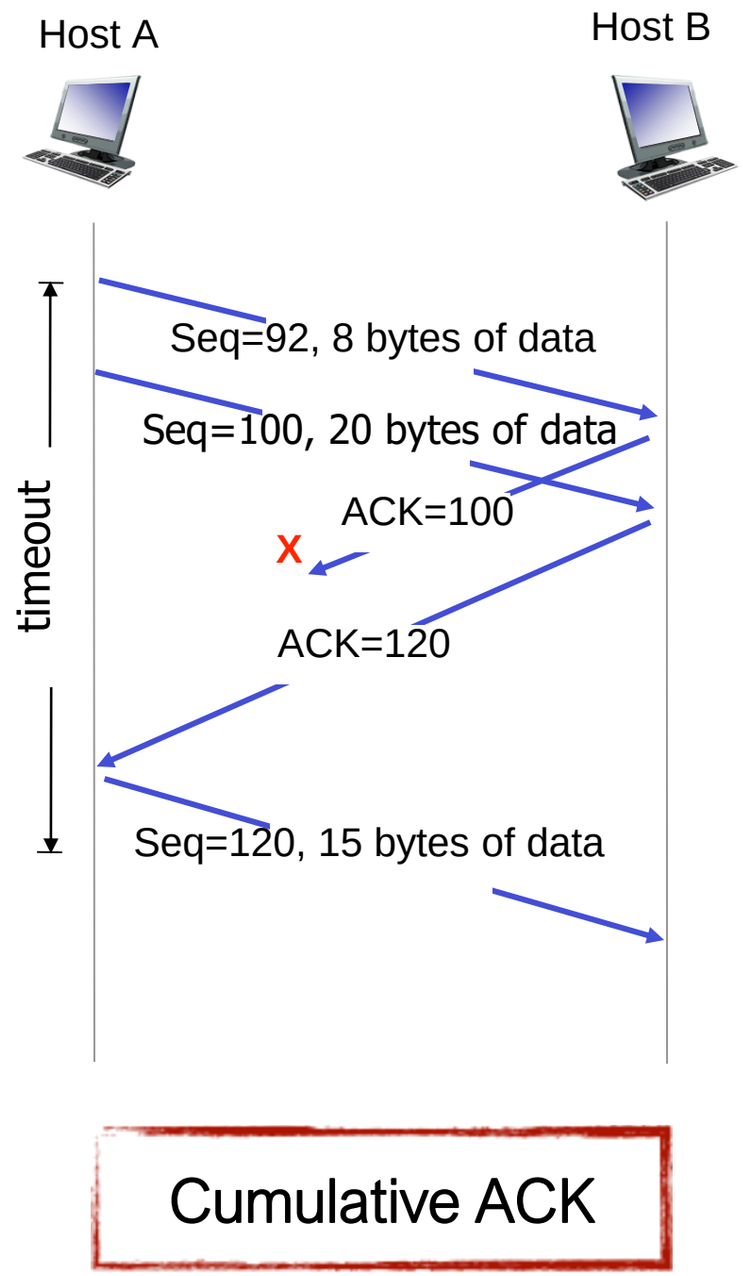


Lost ACK



Premature Timeout

TCP: Retransmission Scenarios (Cont.)



Cumulative ACK

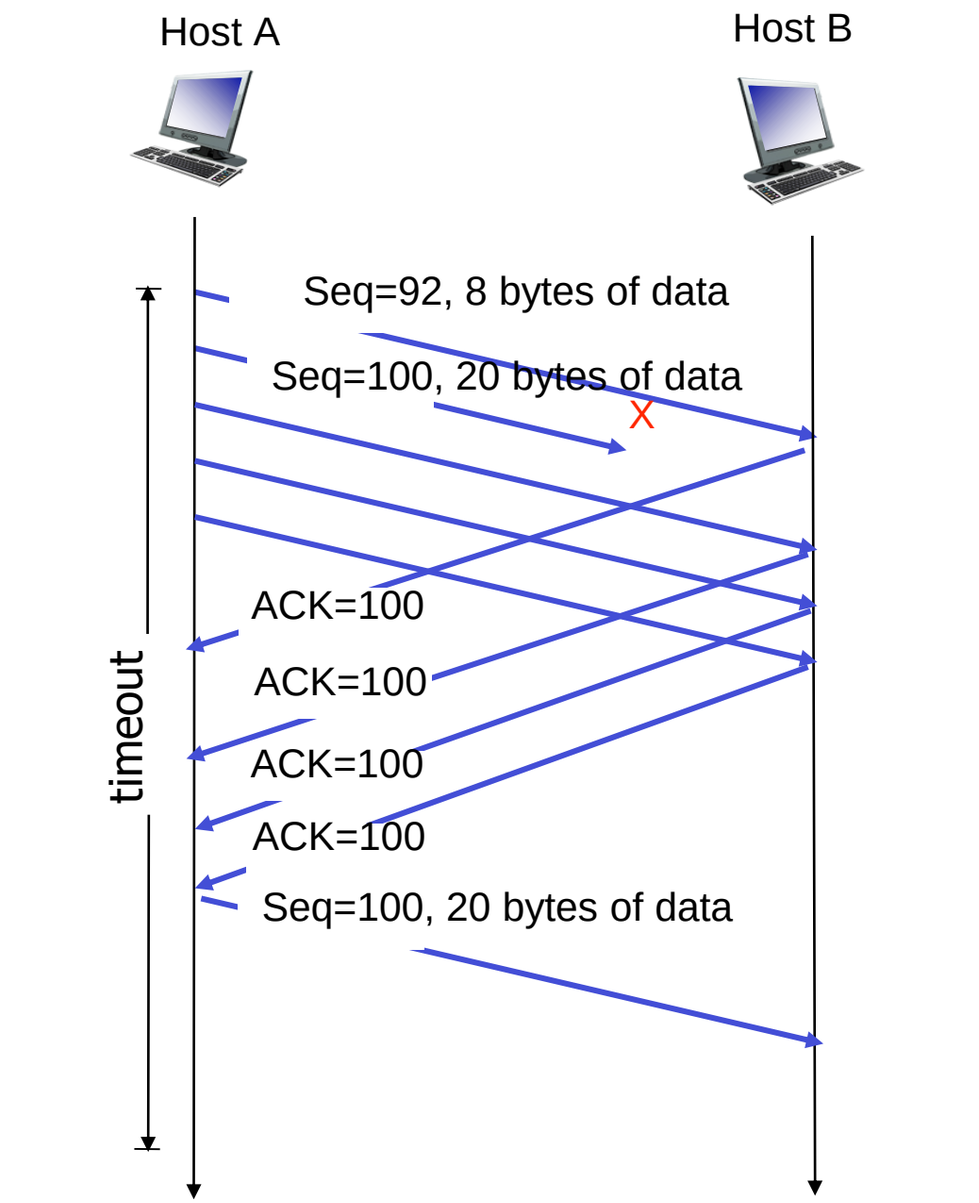
TCP ACK Generation

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 500 ms for next segment. If no segment arrives, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send a single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq. #. Gap detected	Immediately send *duplicate ACK*, indicating seq. # of next expected byte
Arrival of a segment that partially or completely fills gap	Immediate send ACK, provided that segment starts at lower end of gap

TCP Fast Retransmit

- **Time-out period for a segment is often relatively long**
  - Means there is a long delay before sender resends lost packet
- **Lost segments can usually be detected via duplicate ACKs before timeout occurs on that segment**
  - Sender often sends many segments back-to-back
  - If a segment is lost, there will likely be many duplicate ACKs indicating that the lost segment should be retransmitted
- **TCP fast retransmit protocol**
  - If sender receives 3 ACKs for same data (triple duplicate ACKs), resend unacked segment with the smallest sequence number
    - Very likely that the unacked is segment lost, so don't wait for timeout

TCP Fast Retransmit (Cont.)



Fast retransmit after sender receipt of triple duplicate ACKs

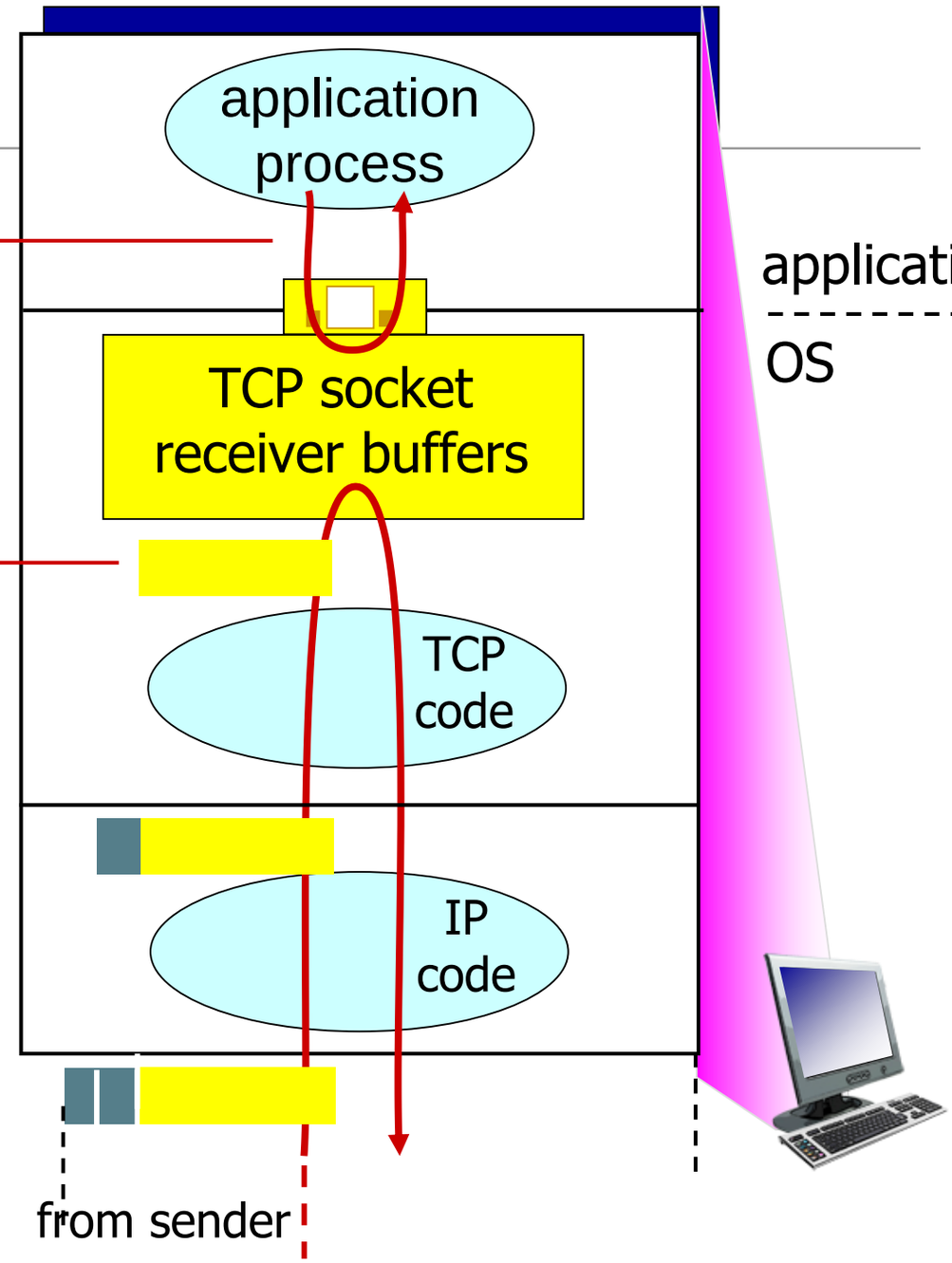
Overview of Transport Layer

- **Transport-layer Services**
- **Multiplexing and Demultiplexing**
- **Connectionless Transport: UDP**
- **Principles of Reliable Data Transfer**
- **Connection-oriented Transport: TCP**
  - Segment Structure
  - Reliable Data Transfer
  - Flow Control
  - Connection Management
- **Principles of Congestion Control**
- **TCP 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

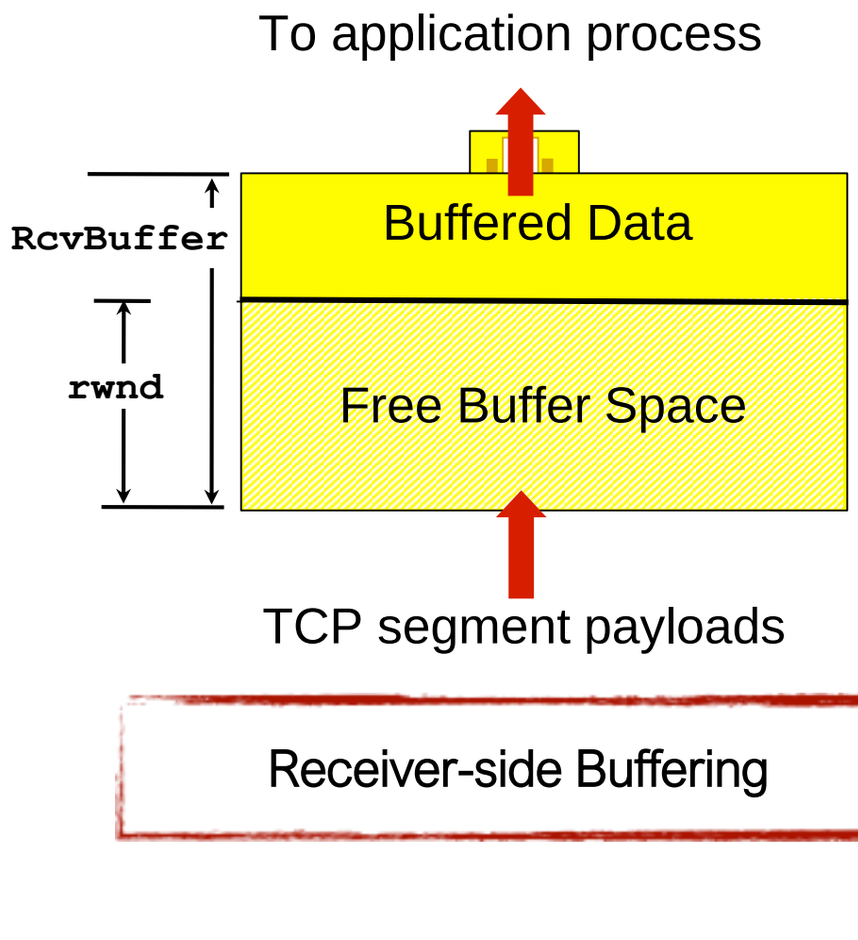
receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

TCP Flow Control

- **The receiver can control the sender, so the sender won't overflow the receiver's buffer by transmitting too much, too fast**
  - TCP socket between Transport and Application layer contains buffers to accumulate received data
  - Data buffer may fill faster than receiver's Application Layer app can empty the buffer
- **Receiver needs way to indicate to sender that its data buffers are too full and the sender should send data at a slower rate**

TCP Flow Control (Cont.)

- **Receiver "advertises" how much buffer space it has available by including a value `rwnd`, in the Window Size field of the TCP header**
  - The `rwnd` value indicates how much free space is available in the buffer
  - Receive buffer size can typically be set via socket options (default is usually 4096 bytes)
  - Many operating systems will automatically adjust the size of the receive buffer
- **The sender limits amount of unacked ("in-flight") data to receiver based on the Window Size field**
  - Guarantees receive buffer will not overflow



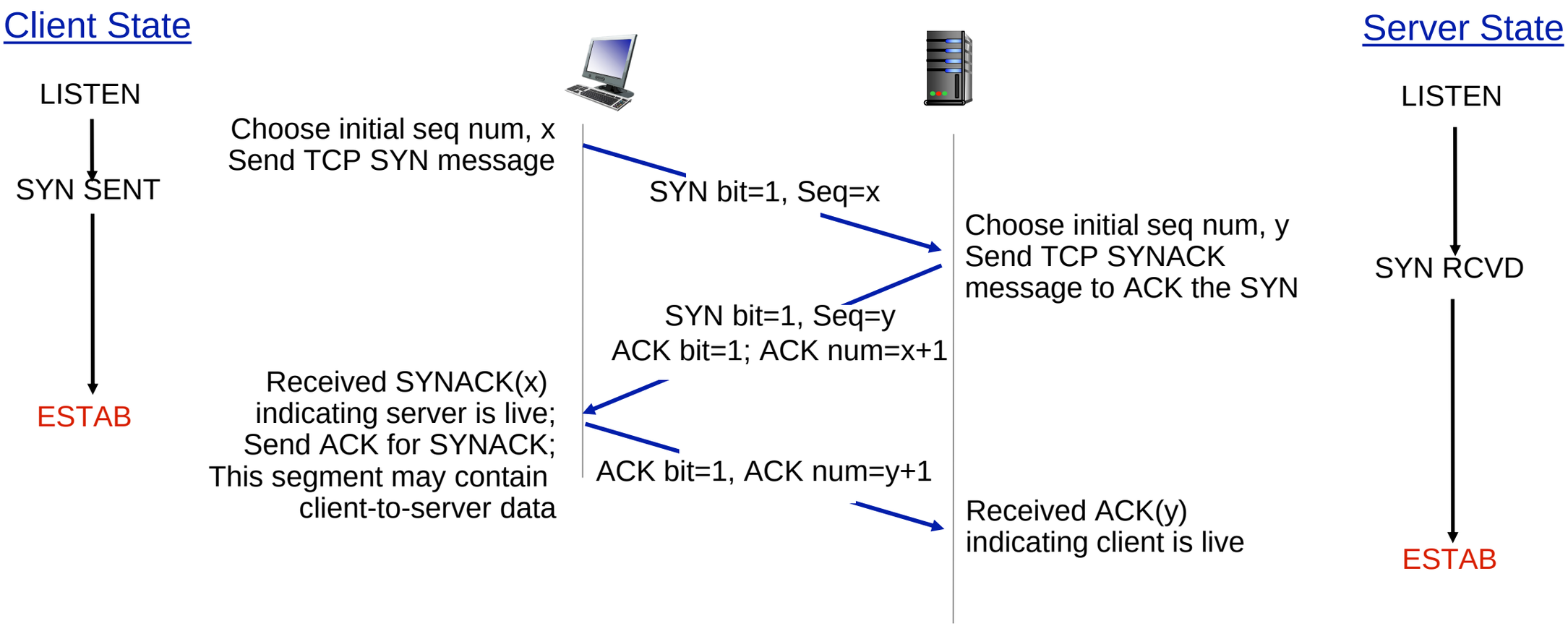
Overview of Transport Layer

- **Transport-layer Services**
- **Multiplexing and Demultiplexing**
- **Connectionless Transport: UDP**
- **Principles of Reliable Data Transfer**
- **Connection-oriented Transport: TCP**
  - Segment Structure
  - Reliable Data Transfer
  - Flow Control
  - Connection Management
- **Principles of Congestion Control**
- **TCP Congestion Control**

Connection Management

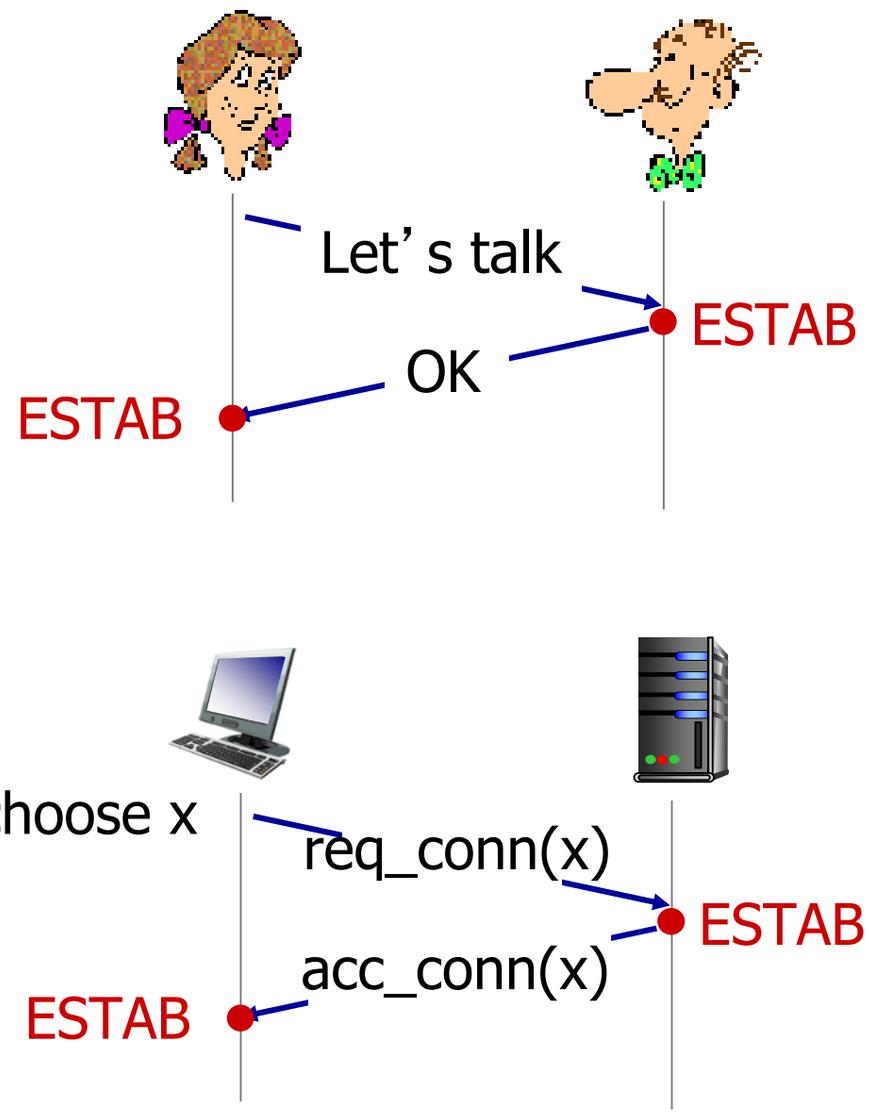
- **Before exchanging data, the sender and receiver perform a 3-way-handshake:**
  - Agree to establish connection (each knowing the other is willing to establish connection)
  - Agree on connection parameters

TCP 3-way Handshake



Agreeing to establish a connection

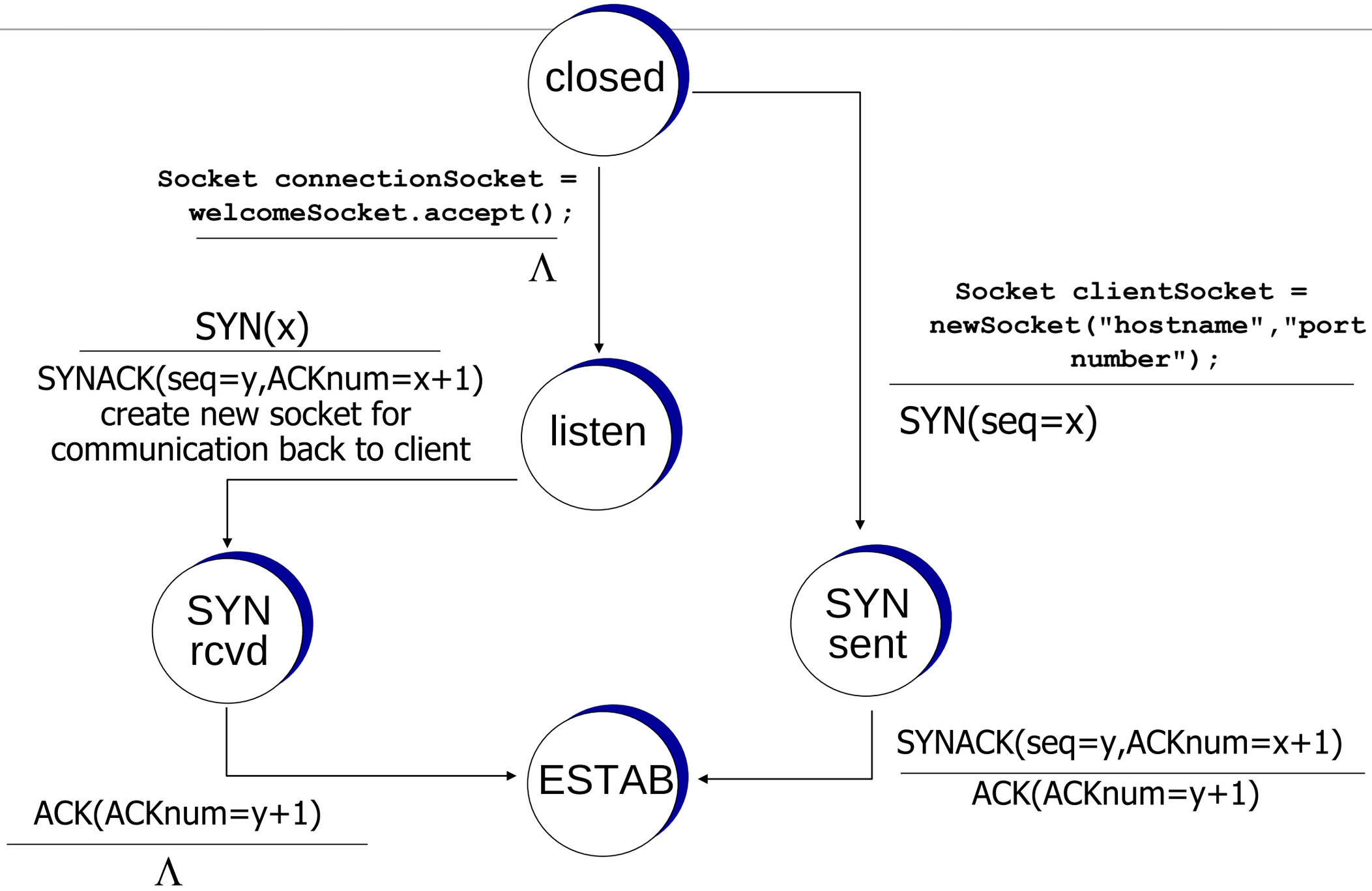
2-way handshake:



Q: **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**

TCP 3-way handshake: FSM



TCP: Closing a Connection

- **Client and server each close their side of the 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 (Cont.)