Computer Networks and Communication

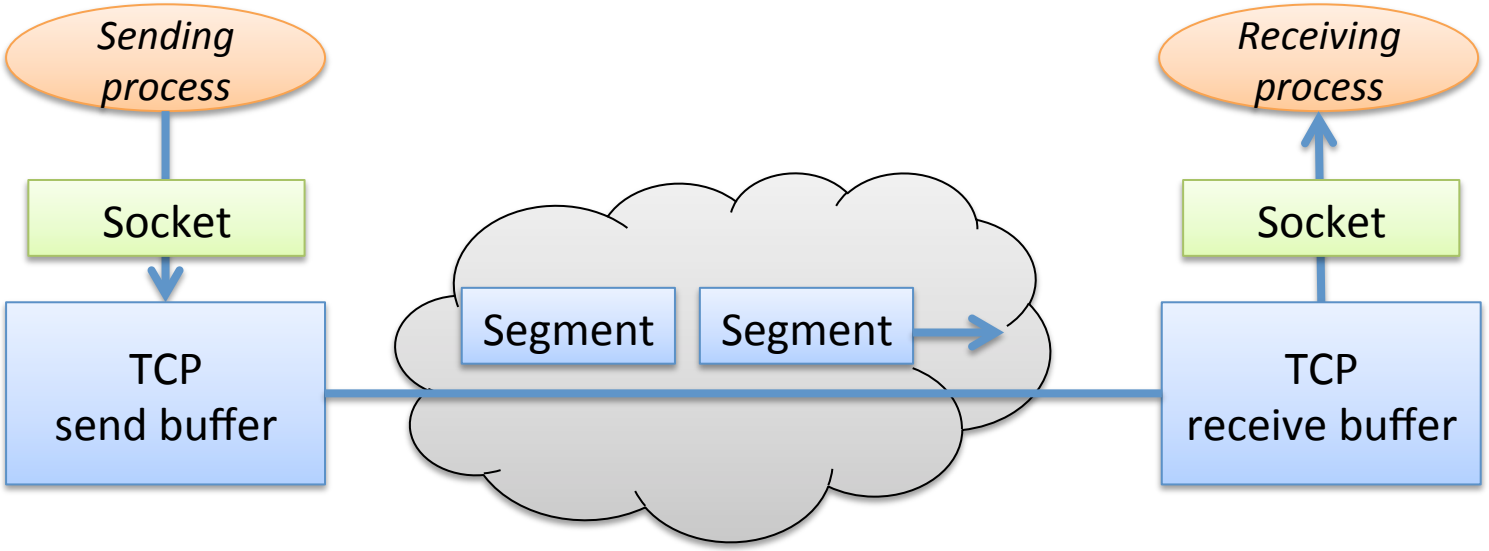
Lecture 7

TCP Protocol

TCP

- Point-to-Point
  - one sender, one receiver
- Reliable and in-order **byte stream**
  - No message boundaries
- Pipelined
  - Both sender and receiver have buffers
  - TCP congestion and flow control set the window size
- Full duplex service
  - Bidirectional data transfer
- Connection-oriented
  - Three-way handshaking
- Flow controlled
  - Sender will not **overflow** receiver

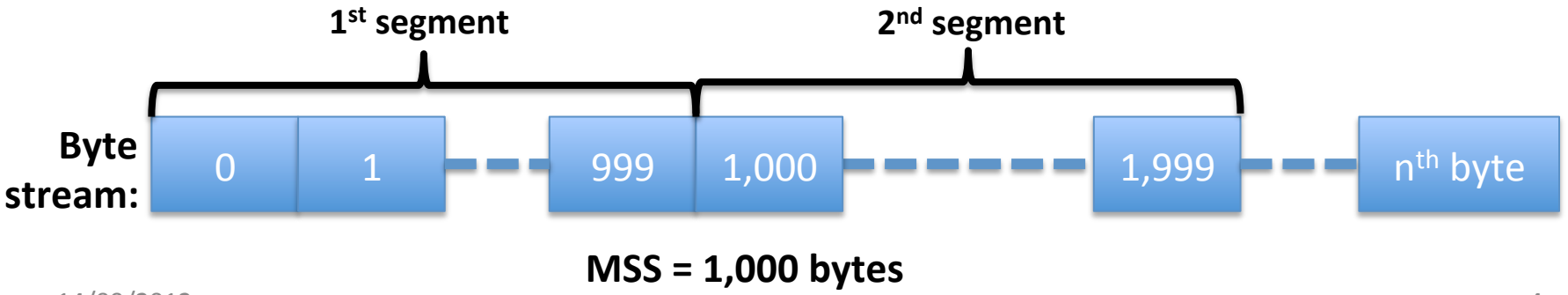
TCP (2)



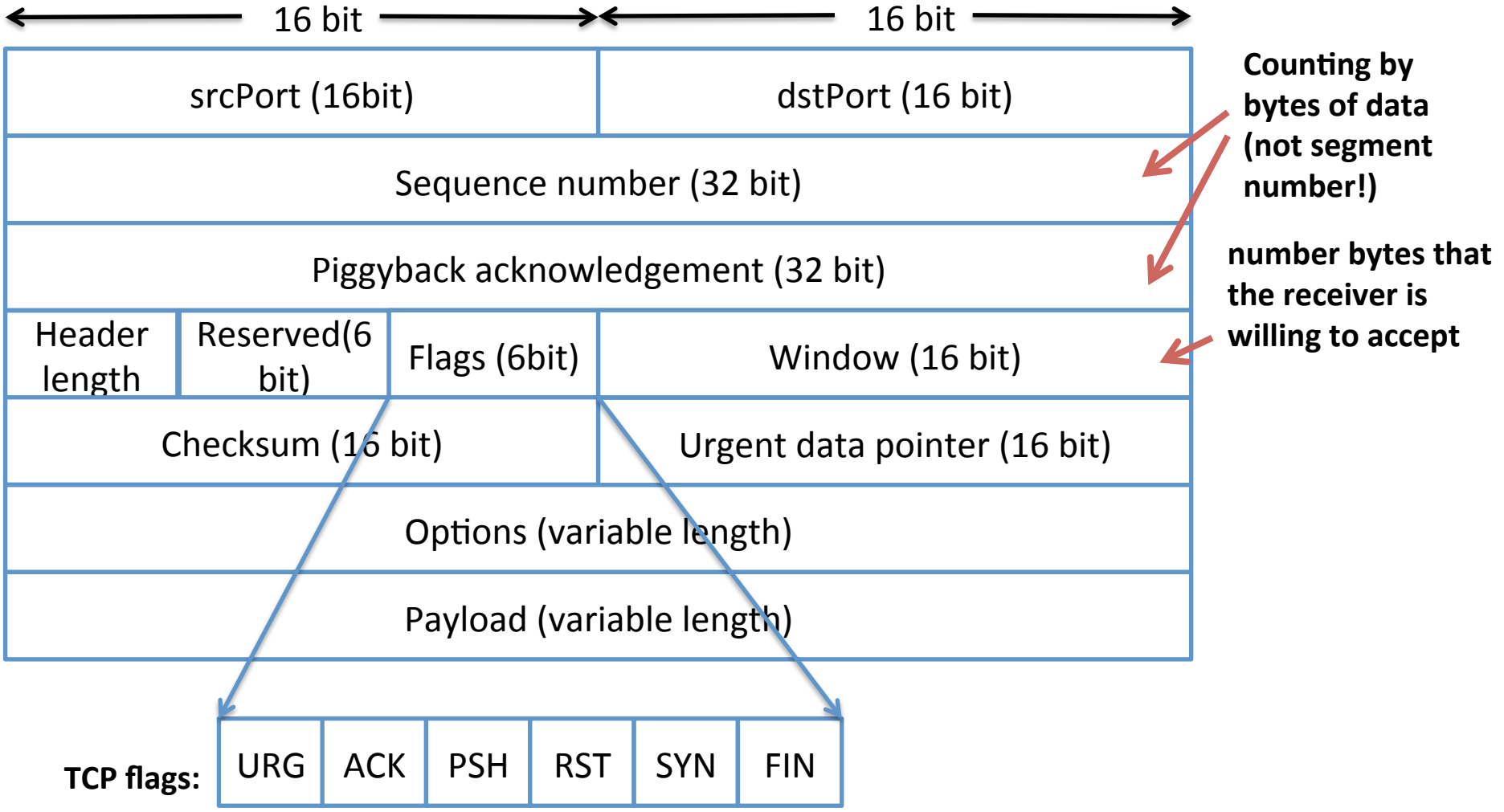
Max length of a segment is specified by Maximum Segment Size (MSS)

Important Characteristics of TCP

- TCP is a full-duplex protocol
  - Processes on both sides can send data to each other at the same time within the same connection
  - When host B wants to send an ACK to host A, it attaches the ACK into one of the packets which is sent to A
  - That is, the ACKs and data from B to A are sent in the same packet
- Seq# used in TCP is the byte-stream number of the first byte in the segment

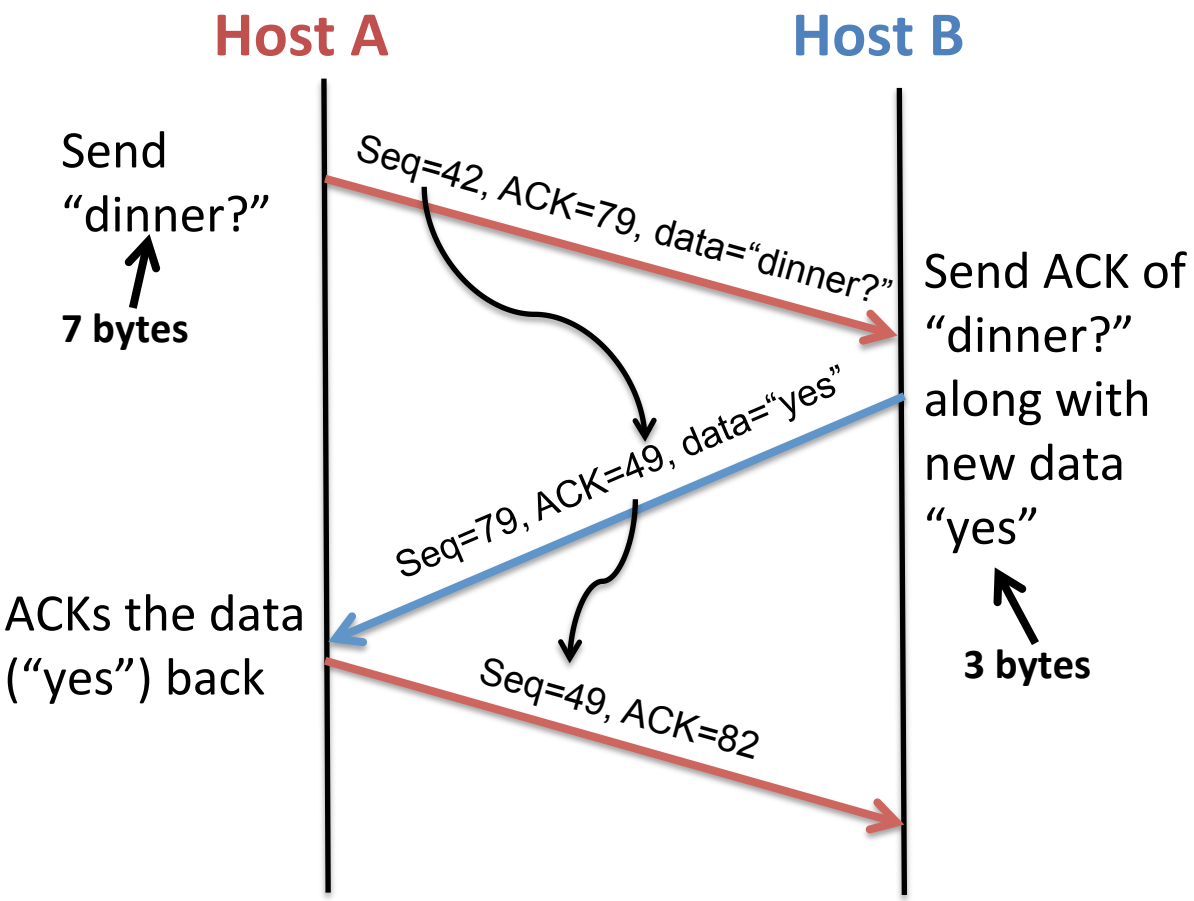


TCP Header

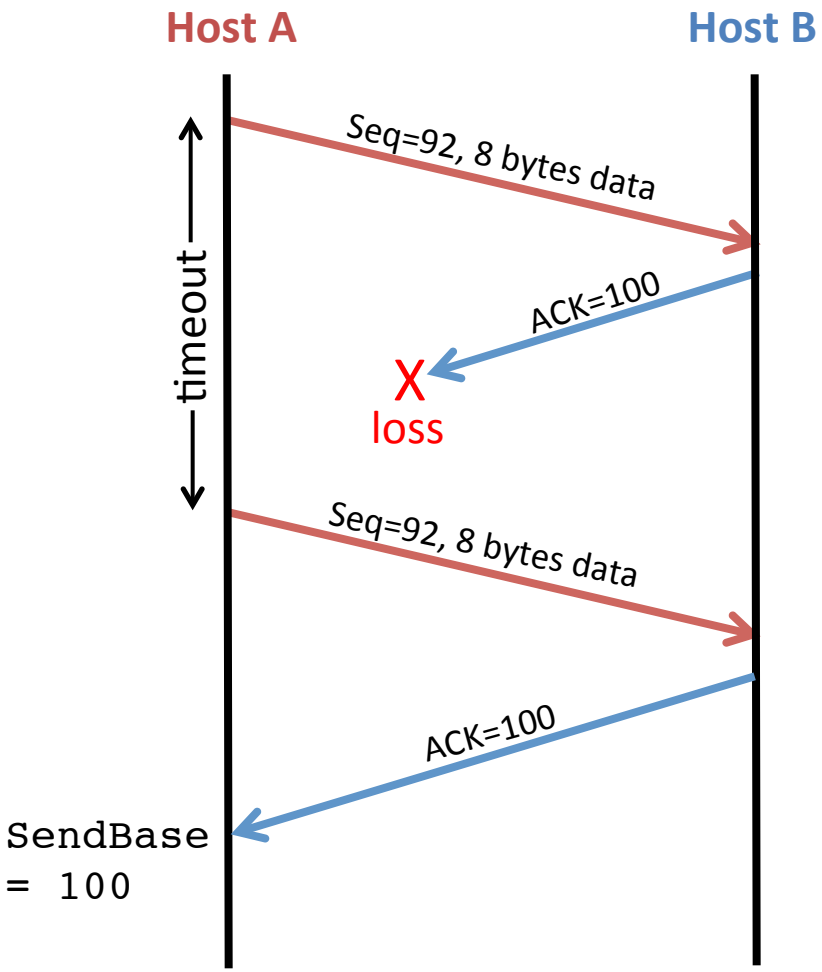

16 bit
16 bit
srcPort (16bit)
dstPort (16 bit)
Sequence number (32 bit)
Piggyback acknowledgement (32 bit)
Header length
Reserved(6 bit)
Flags (6bit)
Window (16 bit)
Checksum (16 bit)
Urgent data pointer (16 bit)
Options (variable length)
Payload (variable length)
Counting by bytes of data (not segment number!)
number bytes that the receiver is willing to accept
TCP flags:
URG
ACK
PSH
RST
SYN
FIN

TCP Sequence Numbers and ACKs

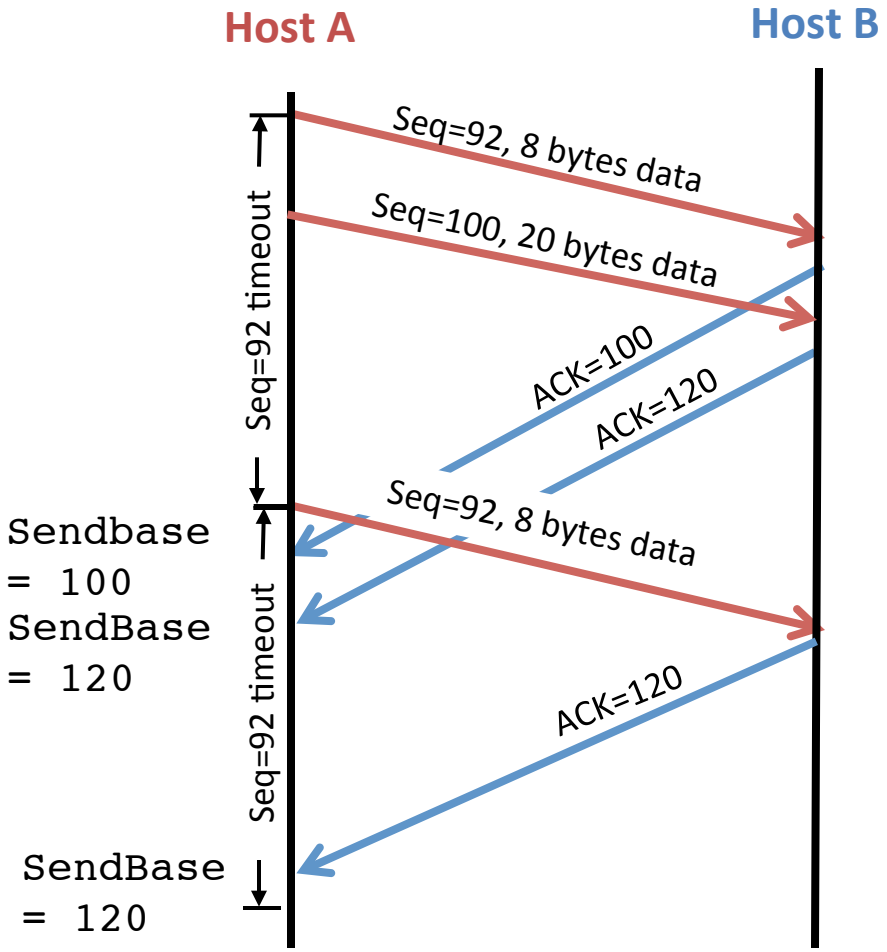
- **Seq#**: Byte stream number of the first byte in segment's data
- **ACKs**: Seq# of the next byte expected from the other side



TCP Retransmission Scenarios

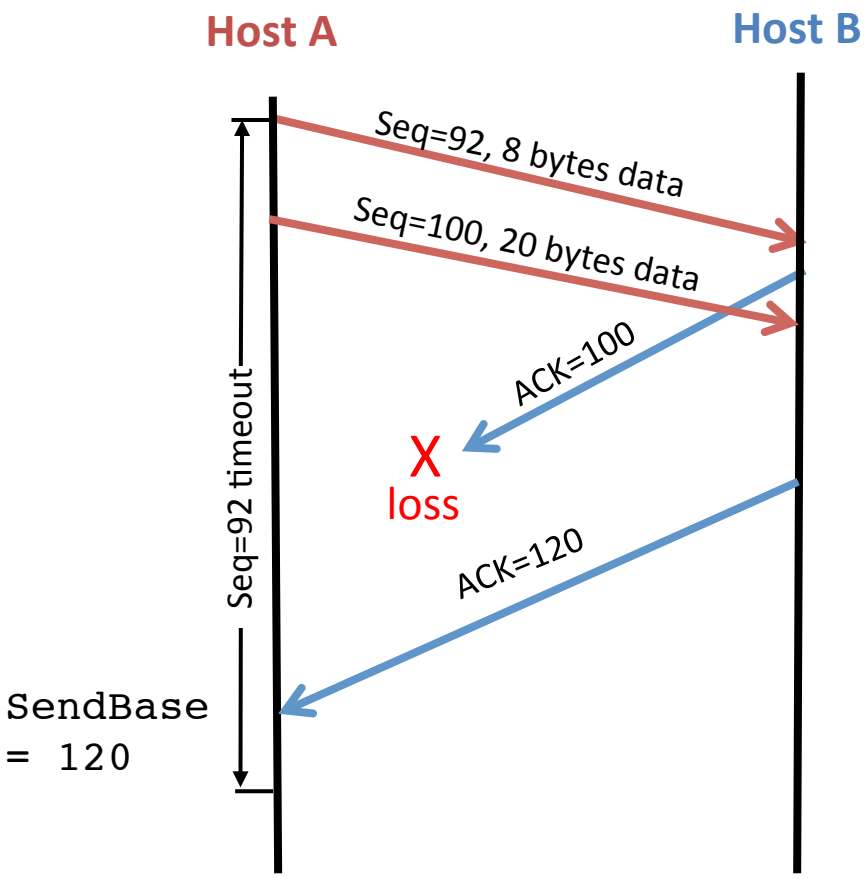


lost ACK scenario



premature timeout

TCP Retransmission Scenarios (2)

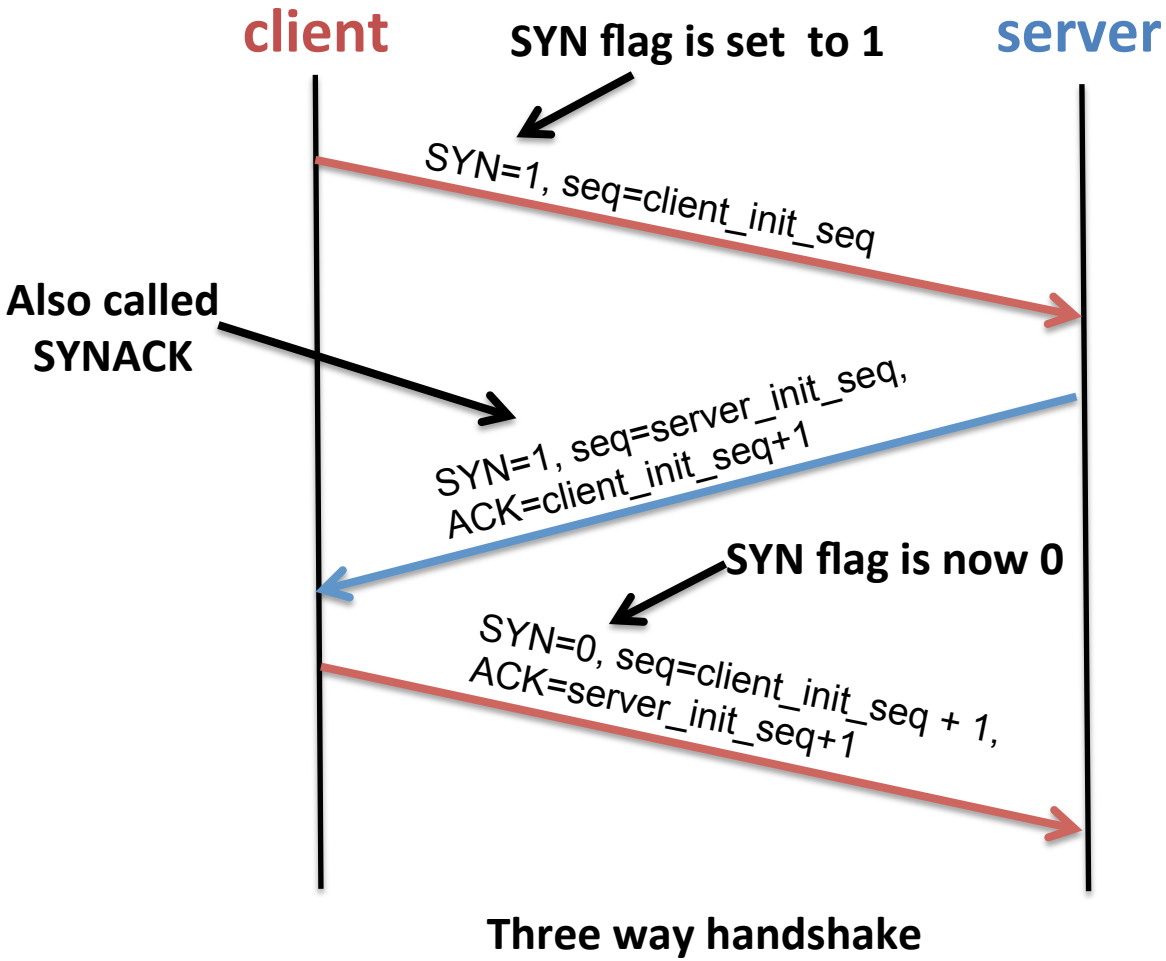


Cumulative ACK scenario

TCP Connection Establishment

- TCP is a connection-oriented protocol
  - The "connection" has to be established before the data can be exchanged
- Three way handshake
  - **Step 1**: Client sends TCP SYN segment to the server
    - Specifies initial Seq#
    - No data
  - **Step 2**: Server receives SYN, replies with SYNACK
    - SYNACK is not just an ACK, but the SYN from server side
    - Server allocates receive buffer
  - **Step 3**: Client receives SYNACK, replies with ACK segment, which may contain data

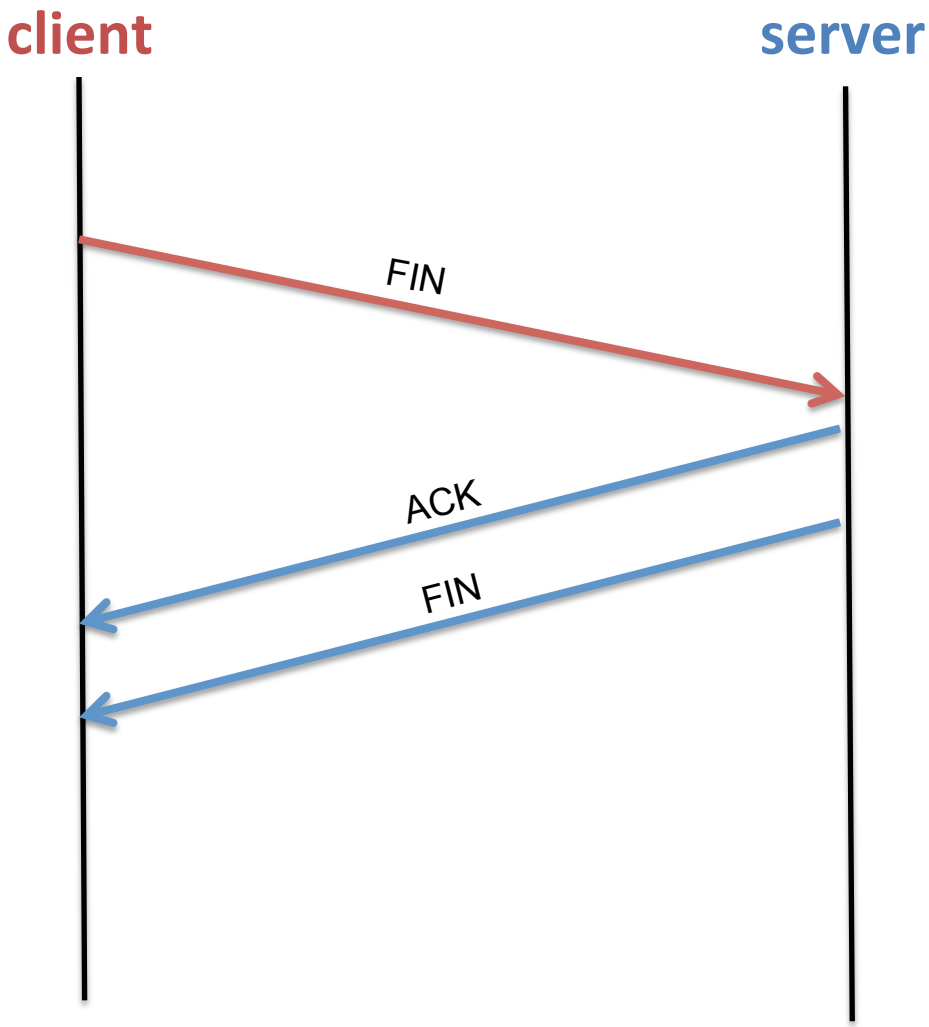
TCP Connection Establishment (2)



Three way handshake

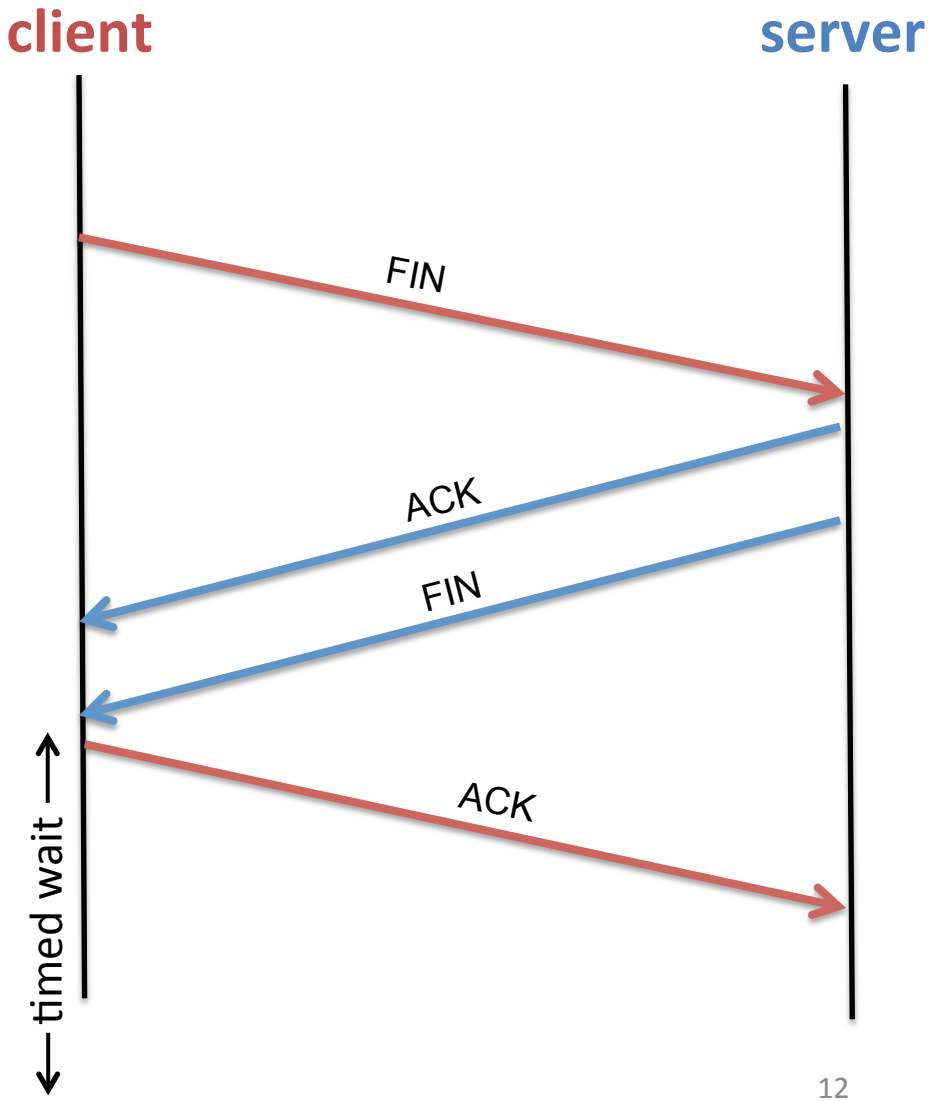
TCP Connection Closing

- **Step 1**:
  - Client closes connection by sending TCP FIN control segment
- **Step 2**:
  - Server receives FIN, replies with ACK
  - Server closes the connection, send FIN



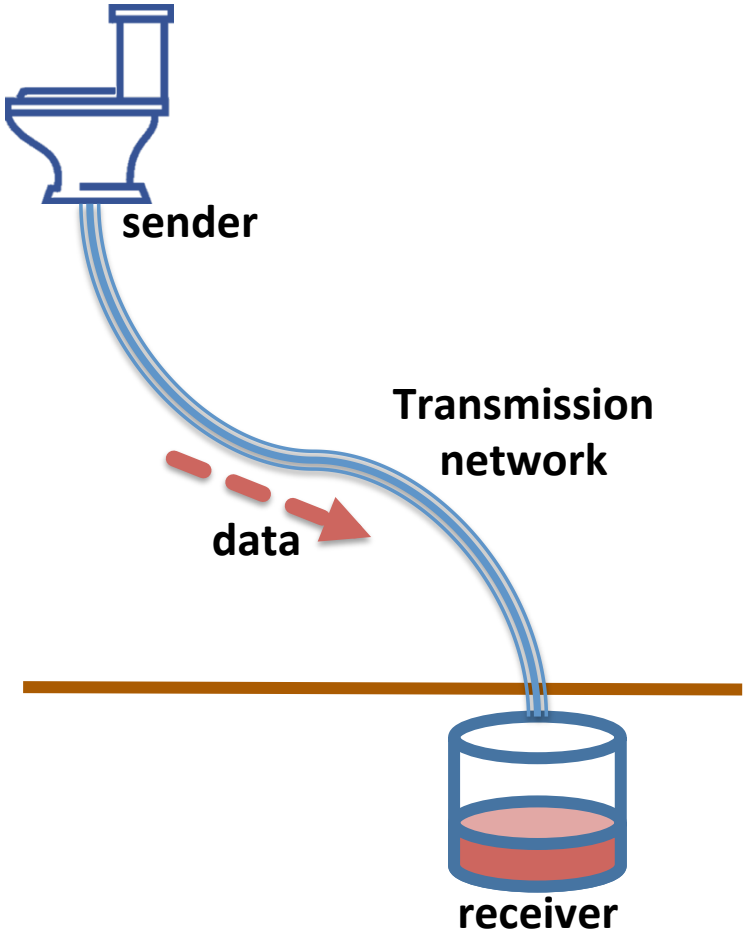
TCP Connection Closing

- **Step 3**:
  - Client receives FIN, replies with ACK
  - Client enters "timed wait" and will respond with ACK if FINs are received
- **Step 4**:
  - Server receives ACK, sends nothing
  - Connection is closed
- **Note**: Server has to send FIN because TCP allows you to close only half of the connection (only one-way)



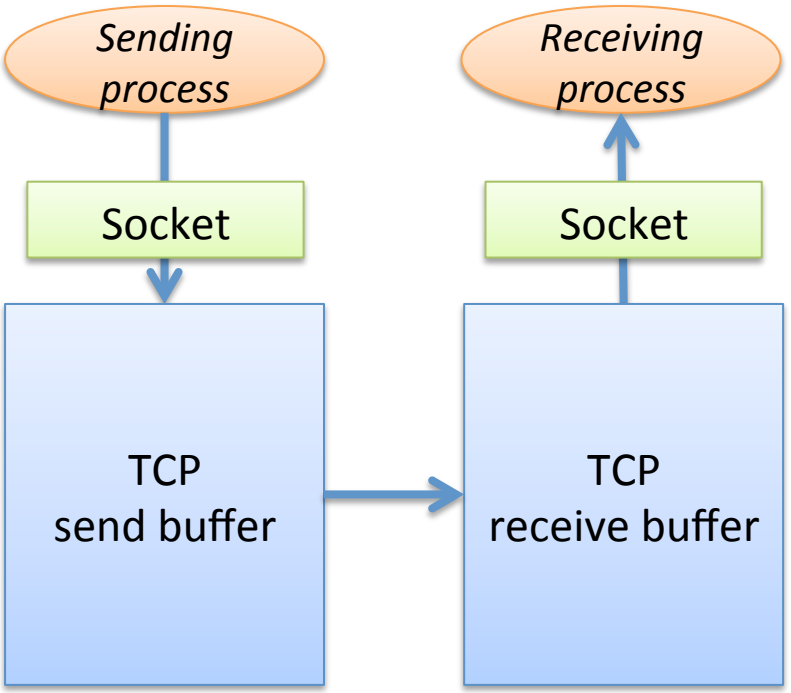
Flow and Congestion Control

- TCP regulates the data-sending rate based on two factors
  - **Capability of the receiver**: How much and how fast can the receiver process the data: **Flow Control**
  - **Capability of the network**: How much data can be sent through the network: **Congestion Control**



TCP Flow Control

- The TCP flow control is designed to prevent the sender to send the data too fast than the receiver's processing capability
  - The receive buffer is full
  - **Overflow**
- The receiver constantly informs the sender how much buffer it has left
- The sender throttles the sending rate accordingly



TCP Flow Control (2)

- At the receiver side

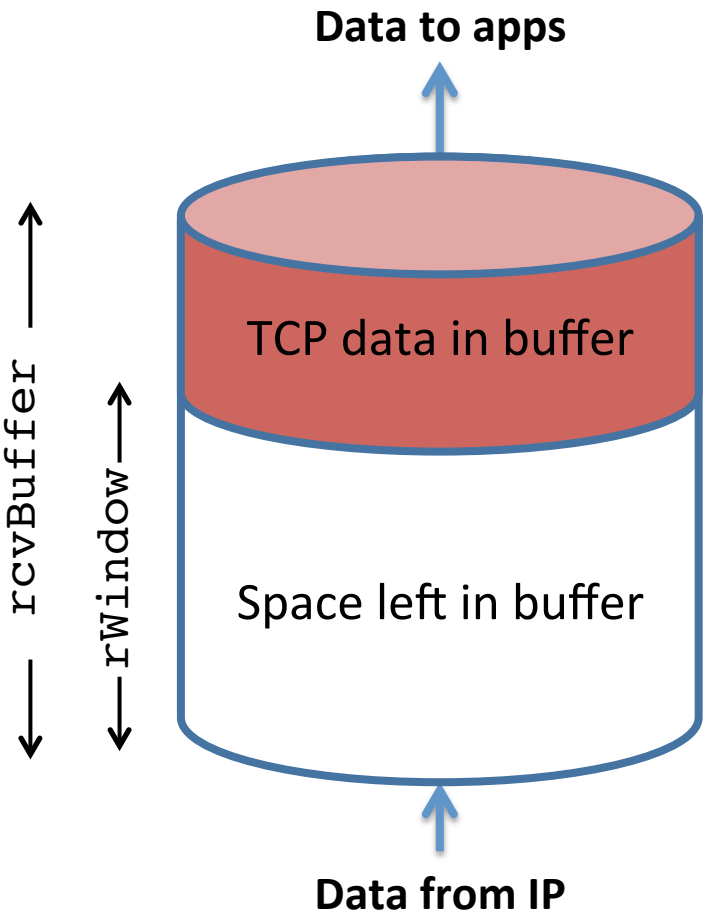
  `lastByteRcvd – lastByteRead =< rcvBuffer`

  `rWindow` `= rcvBuffer – [lastByteRcvd-lastByteRead]`

- `rWindow` is sent to the sender in the **window field in the TCP header**
- The sender makes sure that

  `lastByteSent–lastByteAck =< rWindow`

- `lastByteSent–lastByteAck` is the amount of data in transit



TCP Congestion Control

- Congestion:
  - Too many sources sending too much data too fast for the **network** to handle
  - The same as in traffic in Bangkok
- Consequence of congestion:
  - Packet lost (buffer overflow at routers)
  - Long delay (queuing in the router buffers)
- **Congestion control** is set of methods which try to prevent network congestion
  - It is designed to prevent the sender to send too much data than the network can handle
  - It is not the same as flow control!

TCP Congestion Control (2)

- Sending rate is limited by **congestion window**, denoted by `cWindow`,
  - `lastByteSent–lastByteAck =< cWindow`
- Note that the equation above is similar to that of `rWindow`
  - `lastByteSent–lastByteAck =< rWindow`
- Thus, the sender can send the data at the rate such that
  - `lastByteSent–lastByteAck =< min {rWindow,cWindow}`
- The is `rWindow` specified by the receiver but for the `cWindow`, the sender has to determine by itself.

Determining Sending Rate

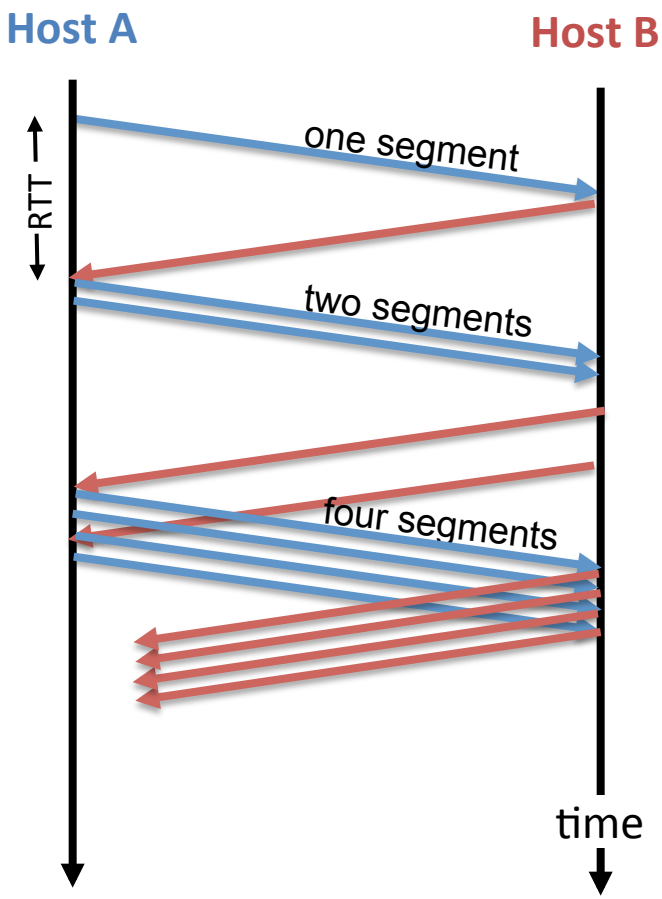
- TCP tries to find a sending rate such that:
  - It is not too fast to congest the network
  - It is not too slow to underutilize the network capacity
- TCP follows these principles:
  - Packet lost implies congestion, thus sending rate should be decreased
  - Arriving ACKs means the network is delivering the packets perfectly, the sending rate can be increased
  - Bandwidth probing: The sender keeps increasing the sending rate until packet loss occurs, then back off from that rate and begins to probe again

Determining Sending Rate (2)

- Sending-rate determination consists of three phases:
  - **Slow Start**: Quickly increase the sending rate. If congestion occurs, switch to congestion avoidance mode
  - **Congestion Avoidance**: Fine-tuning the sending rate
  - **Fast Recovery**: When congestion occurs, it helps maintaining the sending rate from falling back

Slow Start

- Slow start is a bandwidth probing technique used by TCP
- When connection begins, increase rate exponentially until first loss event:
  - Starts `cWindow` at 1 MSS
  - Increase `cWindow` for every ACK received
- Initial rate is slow but ramps up exponentially fast



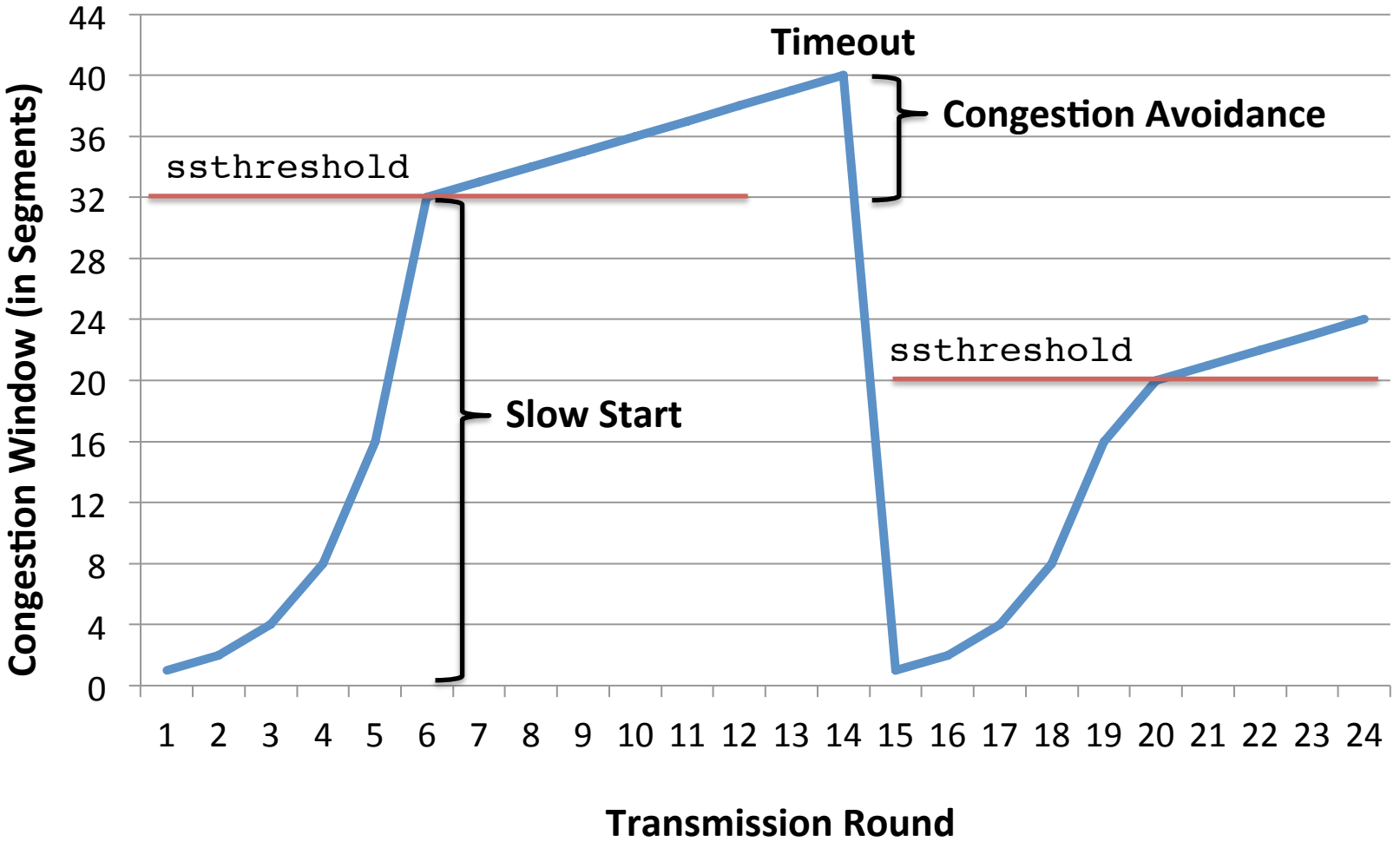
Slow Start (2)

- The exponential growth cannot continue forever
  - It stops at a threshold: **`ssthreshold`**
  - After `ssthreshold` is reached, the `cWindow` increases **linearly**, instead of exponentially
  - This linear growth step is called **congestion avoidance**
- `ssthreshold` is changed every time congestion (loss event) occurs
  - If a loss event occurs at `cWindow = k`, TCP sets `ssthreshold` to `k` / 2
  - That is, `ssthreshold` is set to half the size of `cWindow` which causes the congestion

Congestion Avoidance

- On entry to congestion avoidance state, `cWindow` is approx. half its value when congestion was last encountered
- Here, TCP keeps increasing the `cWindow` linearly until:
  - Timeout occurs:
    - Reset the `cWindow` to 1MSS
    - Set `ssthreshold` to `cWindow/2`
    - Switch to **Slow Start** again
  - Three duplicate ACKs occur:
    - Set `ssthreshold` to `cWindow/2`
    - Enters **Fast Recovery** mode

Congestion Avoidance (2)



Fast Recovery

- TCP enters fast recovery when 3 duplicate ACKs are encountered
- This indicates that the network is still capable of data transmission
- In this state, instead of resetting the `cWindow` back to 1 MSS, is `cWindow` set to the new `ssthreshold`

Fast Recovery (2)