Transmission Control Protocol

ITS 413 – Internet Technologies and Applications

Contents

- Review of TCP Services and Features
- Connection Management
- Error Detection and Recovery
- Flow Control
- Congestion Control
- Performance Issues
- TCP for Wireless Networks

TCP Services

- Reliable stream transport service
  - Stream of bits (or bytes) flow between end-points
    - Stream is unstructured
  - Virtual circuit connection
    - Set up a connection before sending data
  - Buffered transfer
    - Applications generate any sized messages
    - TCP may buffer messages until large datagram is formed
    - Option to force (push) the transmission
  - Full duplex connection
  - Reliability
    - Positive acknowledgement with retransmission

TCP Message Format

- 24 byte header + Data = segment

<table>
<tr><td colspan="3">Source Port</td><td colspan="2">Destination Port</td></tr>
<tr><td colspan="5">Sequence Number</td></tr>
<tr><td colspan="5">Acknowledgement Number</td></tr>
<tr><td>Hdr Len</td><td>Reserved</td><td>Code Bits</td><td colspan="2">Window</td></tr>
<tr><td colspan="3">Checksum</td><td colspan="2">Urgent Pointer</td></tr>
<tr><td colspan="4">Options (if any)</td><td>Padding</td></tr>
<tr><td colspan="5">Data</td></tr>
<tr><td colspan="5">…</td></tr>
</table>

  - Header Length needed because Options field varies in length
  - Code Bits: indicate meaning of segment (SYN, ACK, URG, …)
  - Urgent Pointer: position in segment where urgent data ends
  - Checksum uses pseudo-header like UDP
  - Options: max segment size, window scaling, SACK, …

Segments, Bytes and Sequence Numbers

- TCP messages send are called *segments*
- But TCP operates on a stream of bytes
  - Sliding windows and sequence numbers refer to bytes (not segments or messages)

Connection Management

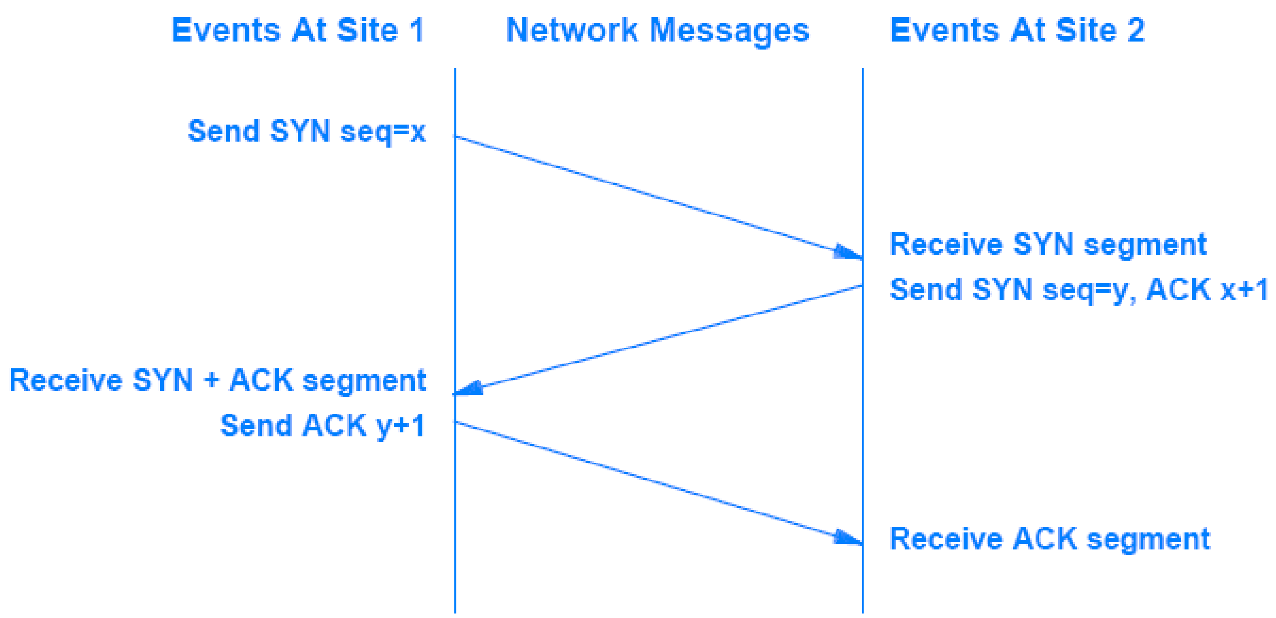
- Establishing a connection:
  - Three-way handshake
    - Both sides ready to transmit
    - Agree upon initial sequence numbers
  - Ensure no segments from previous connection accepted
- Closing a connection:
  - Each side can close connection
    - One direction can be closed, the other can be active
- Code Bits indicate segment type in connection management:

Type	*Description*
SYN	Synchronise sequence numbers
ACK	Acknowledge data
FIN	Sender is finished sending data
RST	Reset connection
PSH	Push data to receiver asap
URG	Use urgent pointer field

Ports and Connections

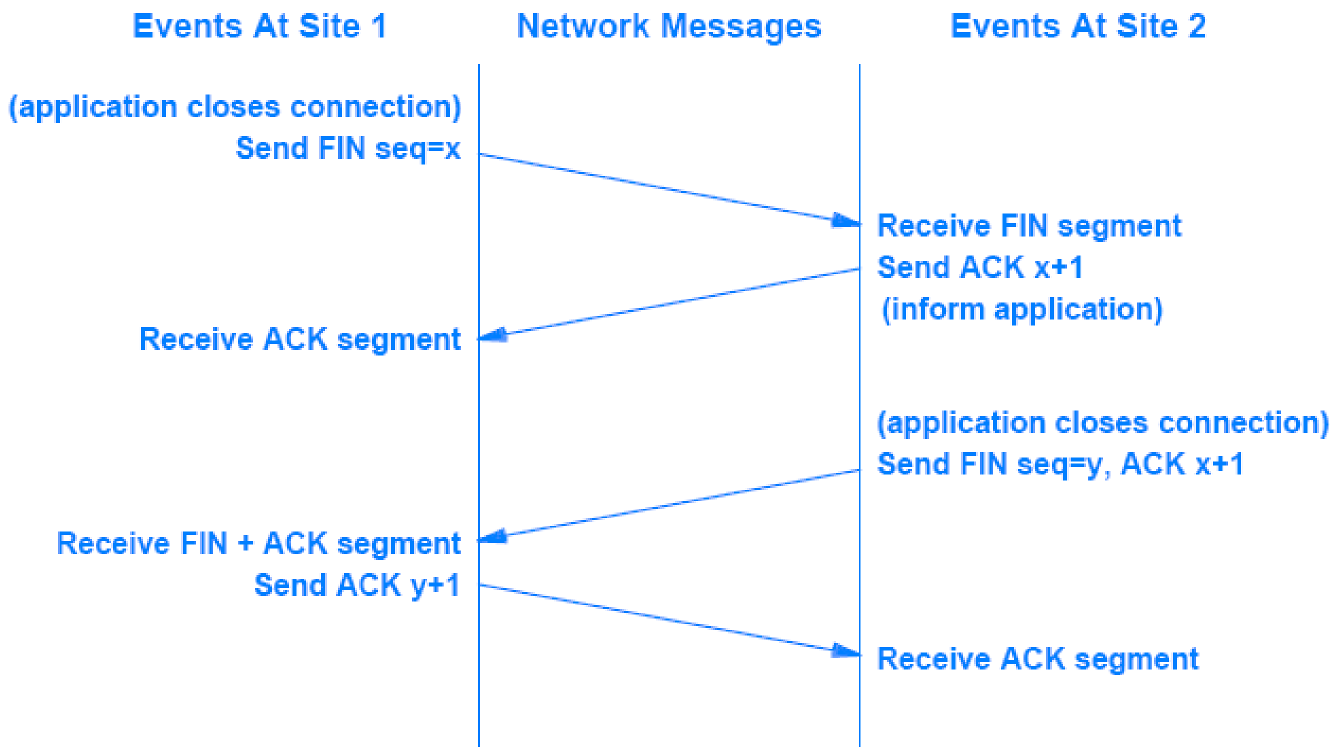
- TCP uses (host, port) pair for source and destination as connection identifier
  - Source IP: 61.47.67.136; Port: 1045
  - Destination IP: 64.233.189.184; Port: 80
- Allows for many TCP connections to same port on same machine
  - E.g. web server on port 80 can accept multiple incoming connections
- TCP allows for passive and active open of connections
  - Passive: wait for incoming connection, e.g. web server
  - Active: start incoming connection, e.g. web browser

TCP Three-way Handshake



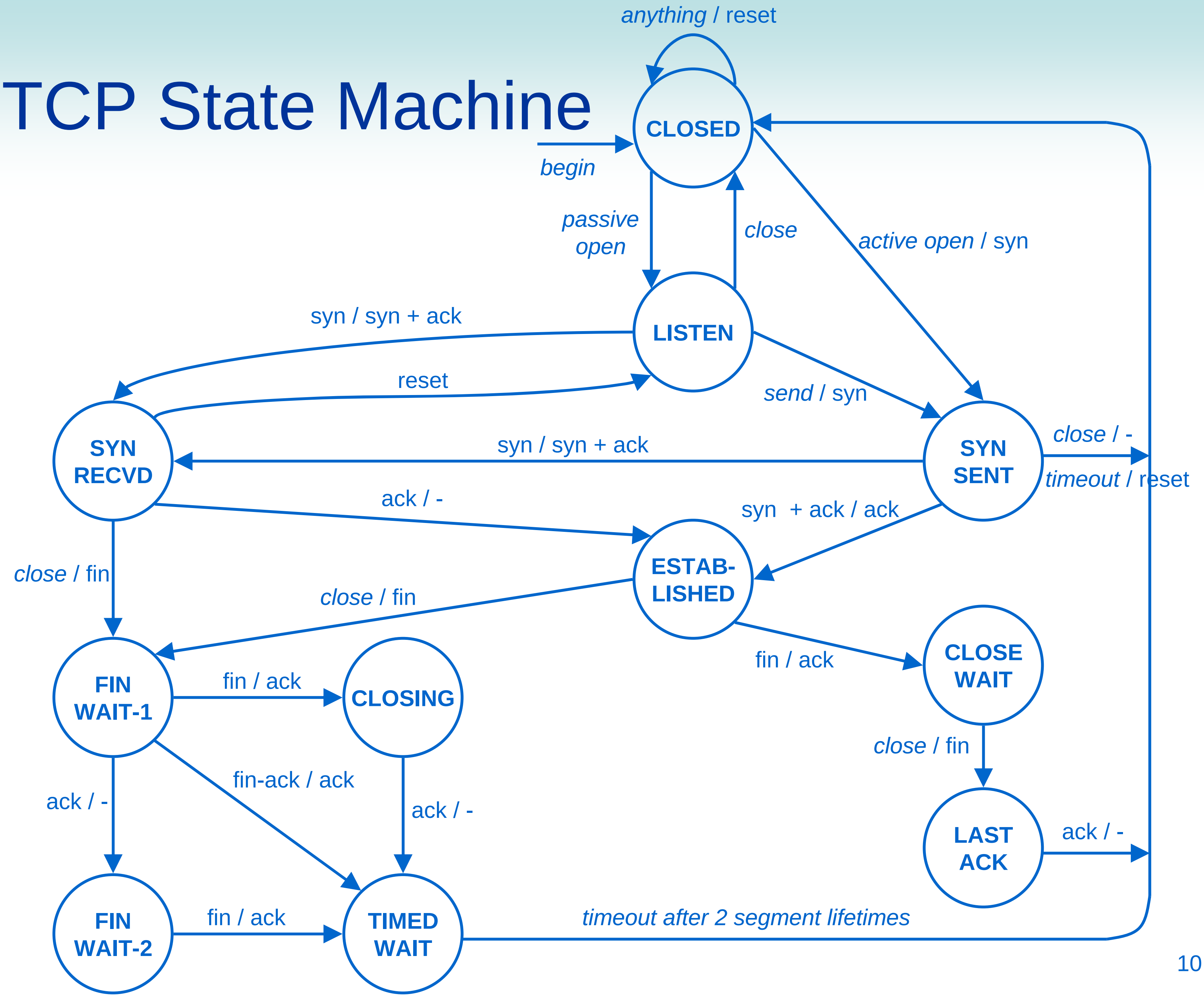
- Handshake synchronises sequence numbers used by both machines
- Handles the loss of messages and receiving duplicates from old connections
- Can send data with the initial SYN packets (not shown above)

Closing TCP Connection



- Program issues the *close()* command for graceful close
- Can close in one direction, but still open in other direction
- Also possible for connection reset (abort)

TCP State Machine

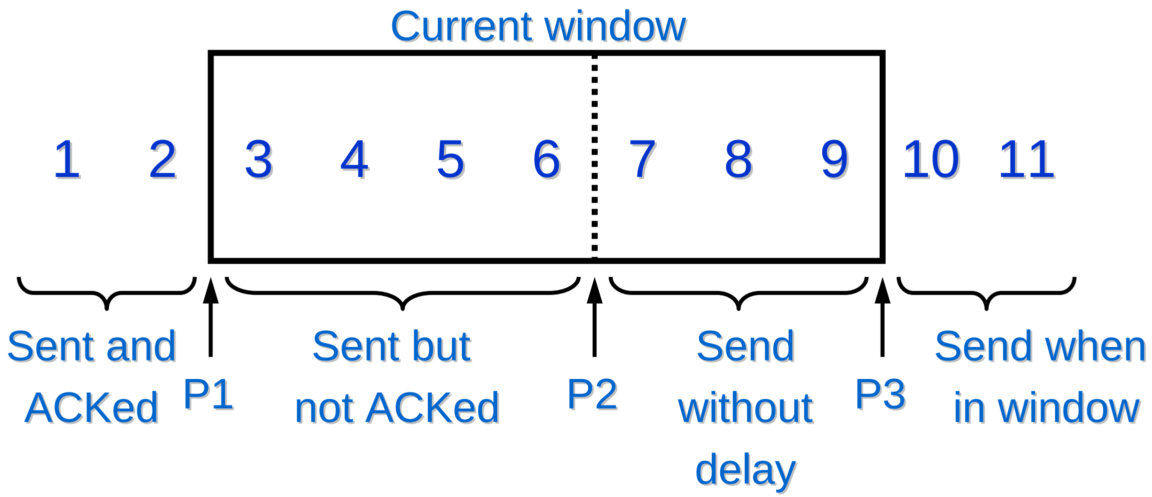


Data Transfer in TCP

- Once a connection is opened:
  - Need reliable delivery of DATA
    - Acknowledgements and retransmissions
  - Do not overflow the receivers
    - Flow control
  - Do not overflow the network (e.g. routers along the path)
    - Congestion control
- TCP using a sliding window mechanism
  - For efficient transmissions
  - To avoid overflow of receivers and network

TCP Sliding Window

- Operates on the byte level, not segment
  - Three pointers (P1, P2, P3) to bytes in the data stream
- Sender and receiver maintain windows for each direction

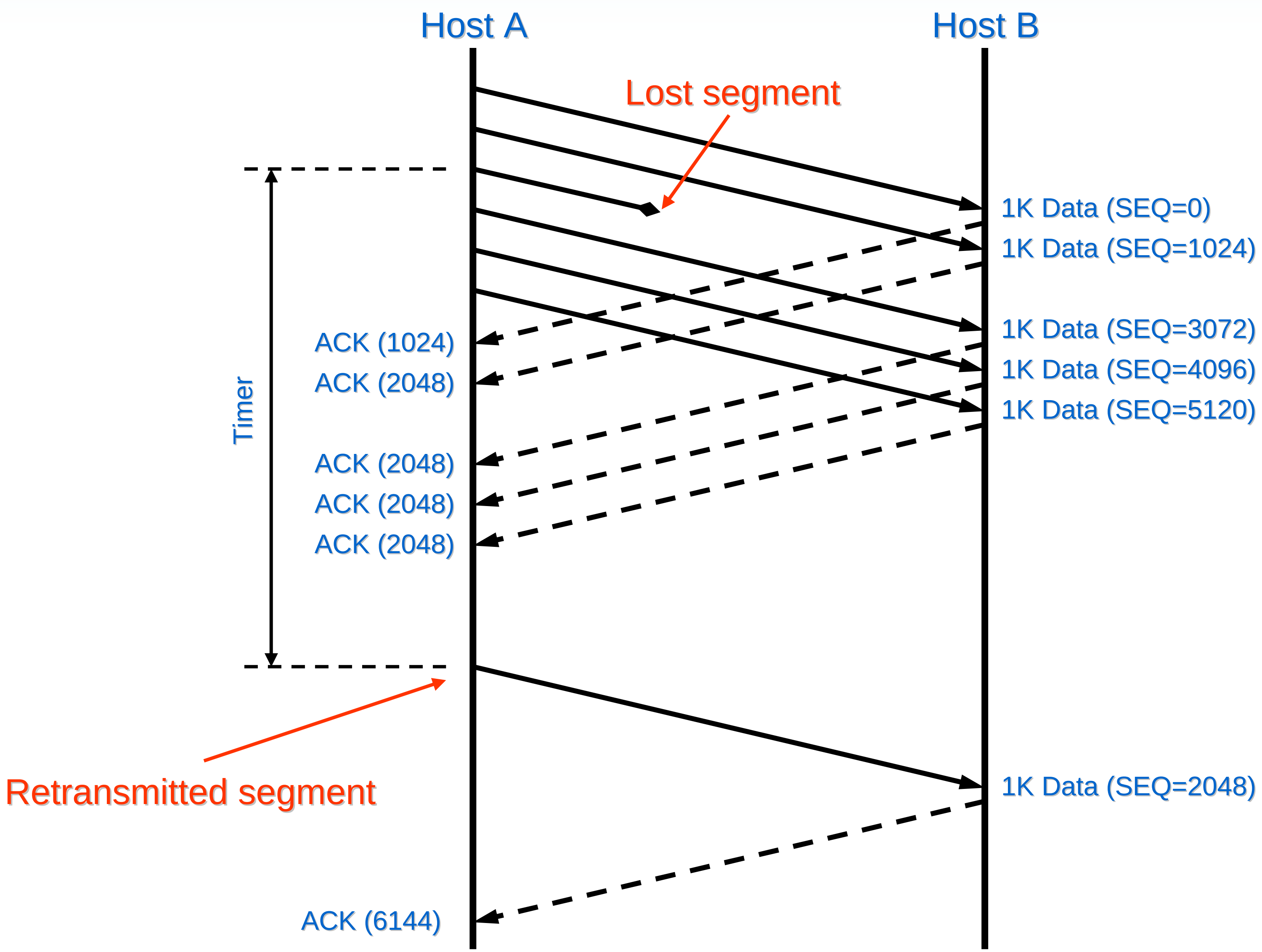


- Variable sized window, based on advertised window

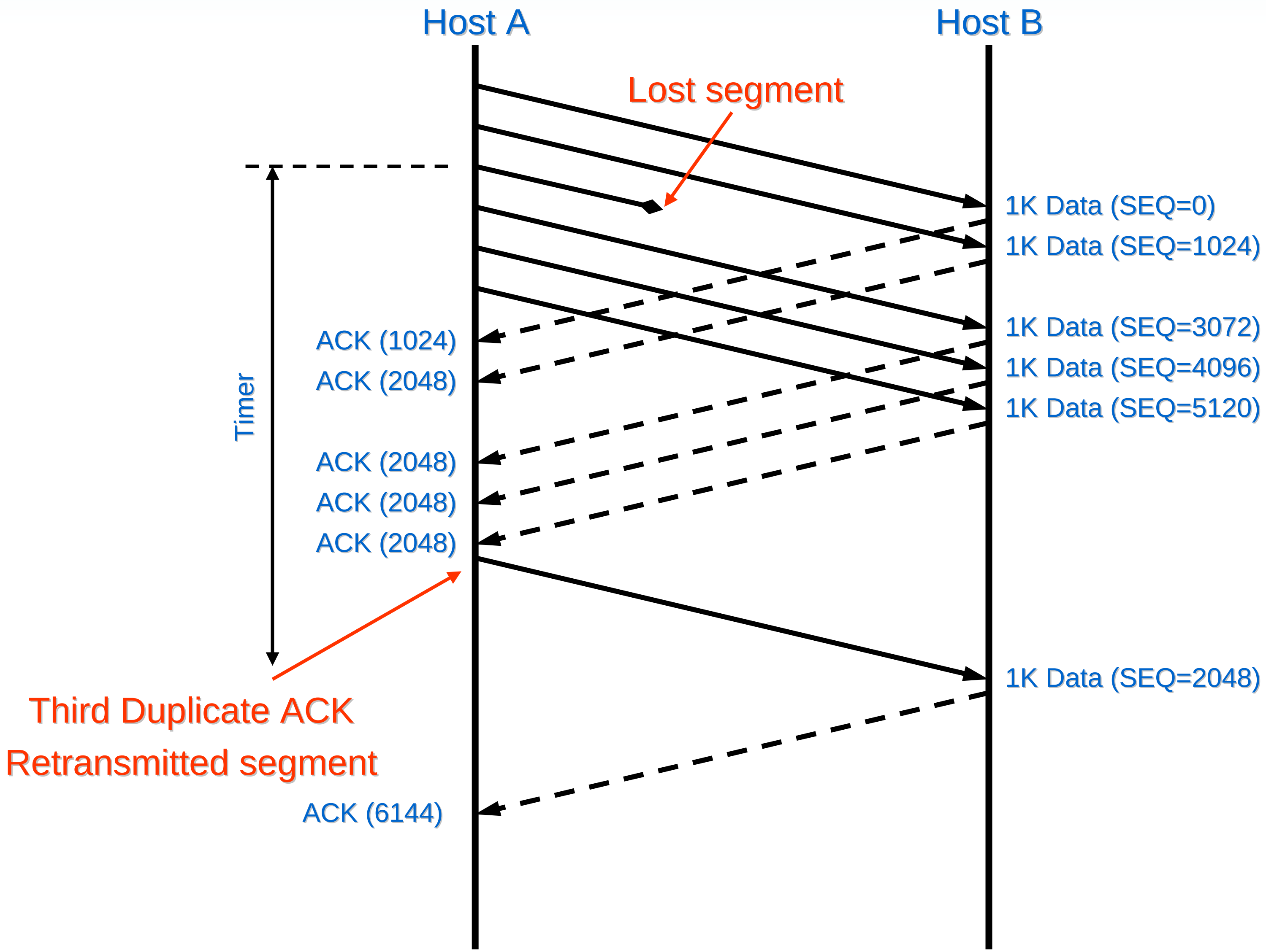
ACK with Retransmit

- Timer started by sender for each segment transmitted
- Receiver sends cumulative acknowledgement for each segment received
  - Sequence number of next byte expected to receive
    - If byte with sequence number 1000 received, ACK will indicate 1001 as next expected byte
- If timer expires, segment is retransmitted
- Improvement - Fast Retransmit:
  - If 3 duplicate ACKs received, retransmit
  - No need to wait for timeout
- In practice, implementations may be different than above (to avoid many timers) – but same principle

TCP Retransmission



Fast Retransmit



Estimating Timeouts

- TCP doesn't know how long it takes for ACK to be received
  - End-to-end path may contain various link layer technologies and various routers
  - Queuing at routers depends upon network traffic
- TCP monitors path performance and estimates timeouts
  - Estimate RTT:

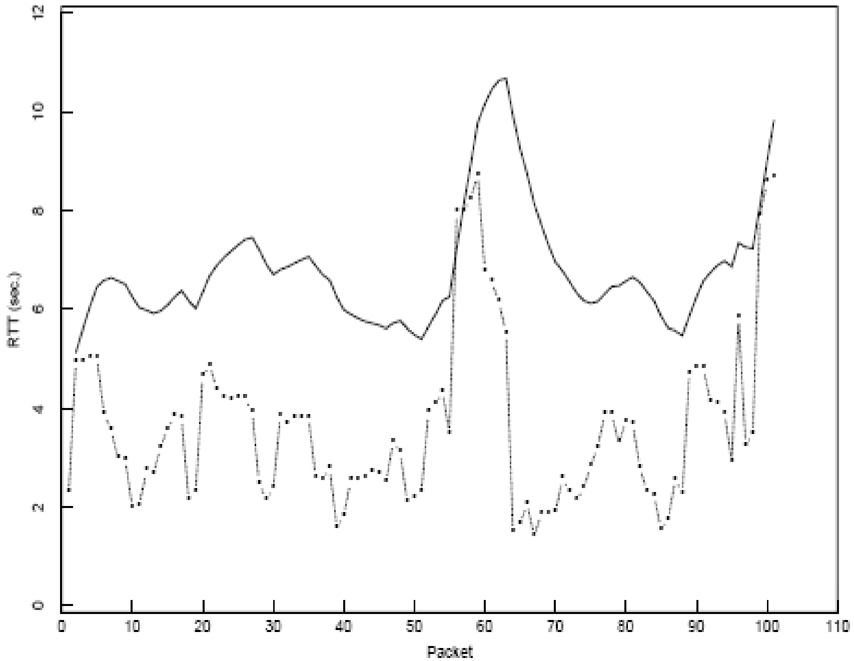
    $RTT = \alpha oldRTT + (1- \alpha)NewRTTSample$

    - α - typically 7/8

    Timeout = RTT + 4*D

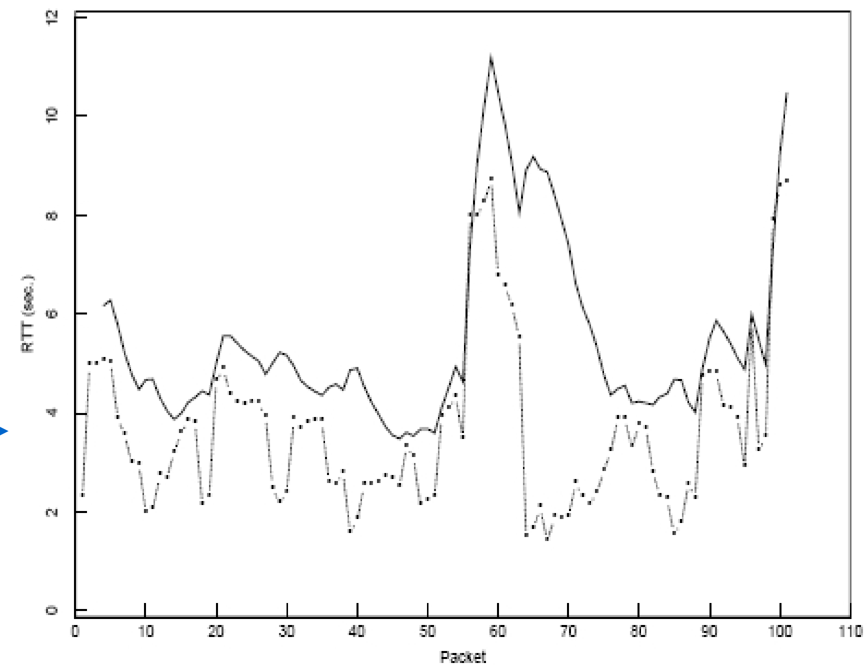
    - $D = \alpha D + (1- \alpha) \mid RTT – M \mid$
      - $\mid RTT – M \mid$ is difference between expected & observed RTT
      - α may not be same as α used above
  - Karn's algorithm: do not update RTT on retransmitted segments; instead Timeout doubled on each failure until success

Timer Estimation Examples



Original RFC793 RTT estimation algorithm

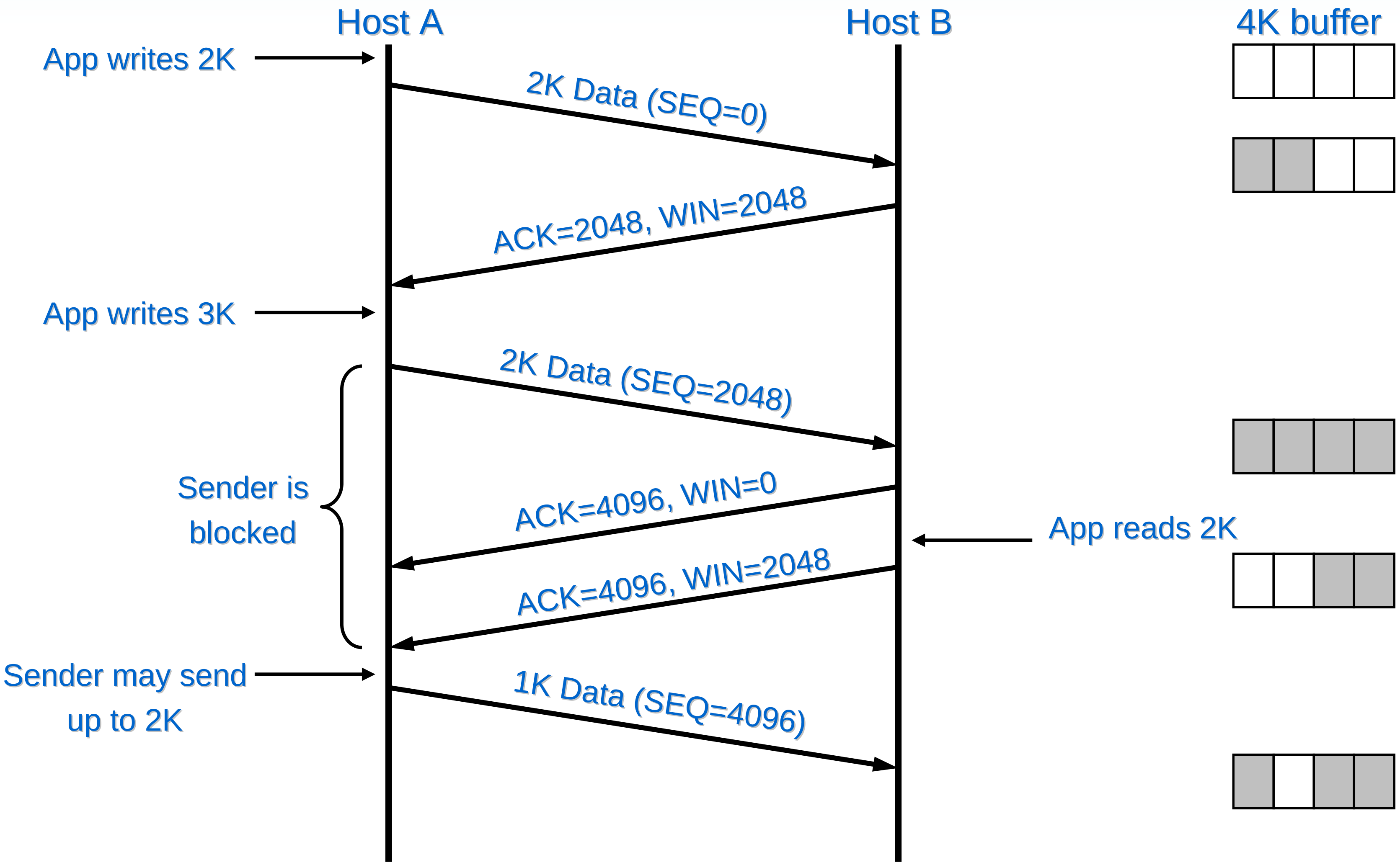
Updated Van Jacobson
estimation



Flow Control

- Aim: Prevent sender from overrunning capacity of receivers
- Needed because:
  - Application cannot keep up with incoming data
  - TCP cannot keep up with incoming segments
- Must take into account:
  - Variable end-to-end round trip times (RTT)
  - Interactions between TCP and IP and application protocols
- General options for flow control:
  - Discard segments that overflow
  - Refuse to accept packets from IP
  - Sliding window protocol (withholding ACKs)
    - All result in retransmissions that consume bandwidth
- TCP flow control
  - Receiver notifies sender of amount of buffer space left
  - Advertised Window (window field in TCP header)

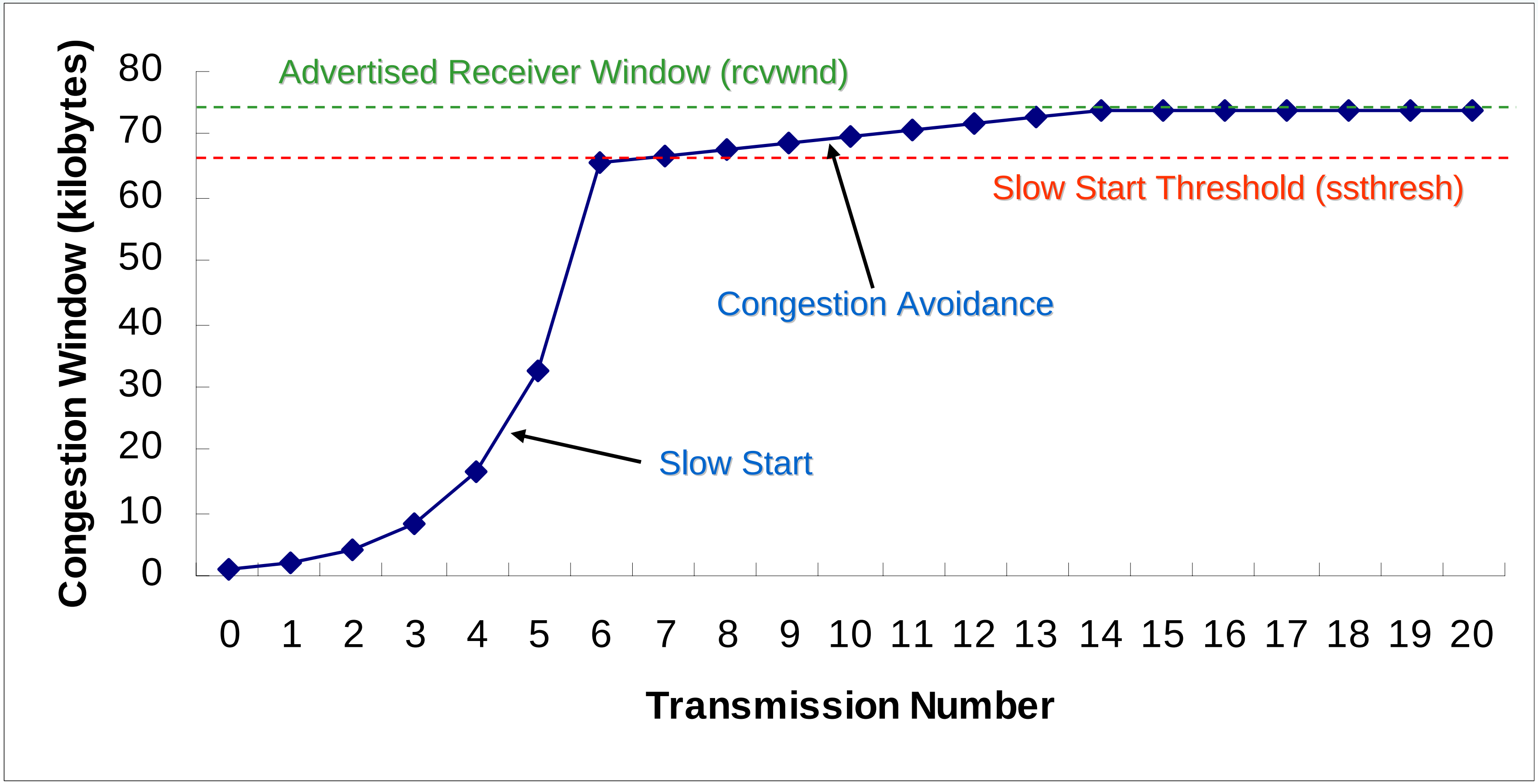
Flow Control Example



Congestion Control

- Flow control protects slow receiver from a fast sender
- Congestion control protects the network from a fast sender
- Without congestion control:
  - To transport protocol, congestion is seen as increased delay
  - Increased delay results in more retransmissions
  - More retransmissions results in more congestion
  - Leads to congestion collapse
- TCP Congestion Control
  - Implicit congestion detection: loss of segments imply congestion
  - Slow Start
  - Multiplicative Decrease
  - Maintain second congestion window at sender
  - Allowed window = minimum (advertised window, congestion window)

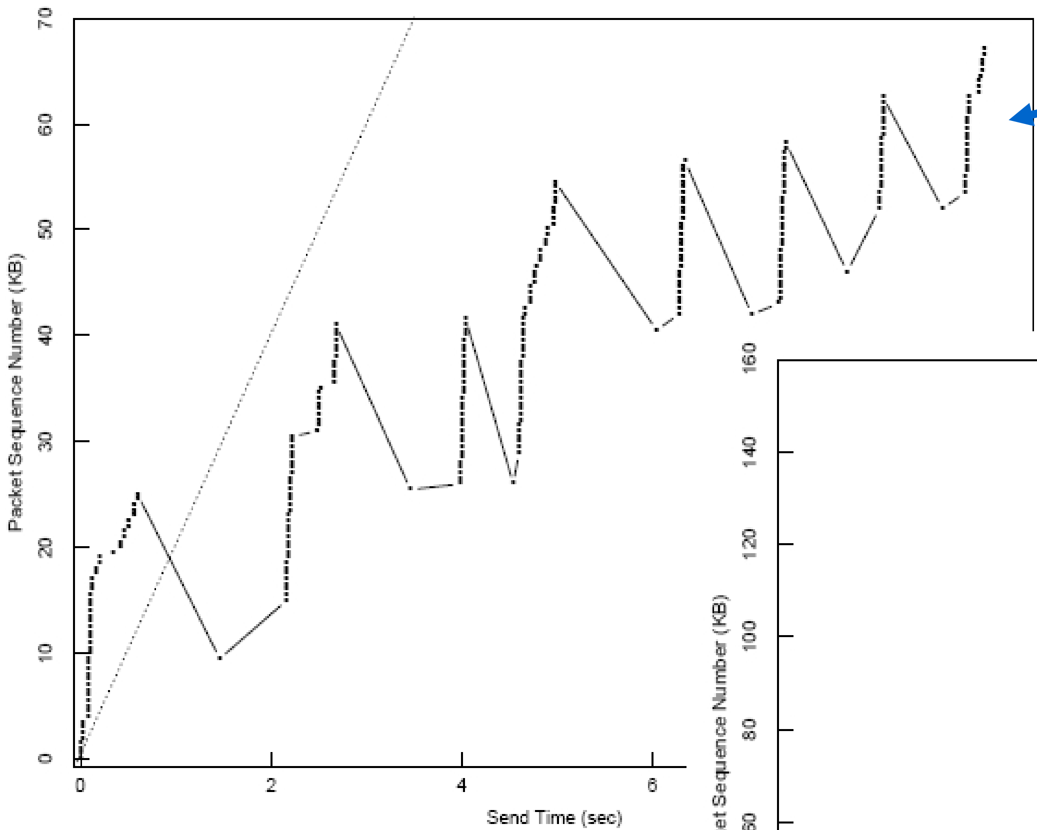
TCP Congestion Control



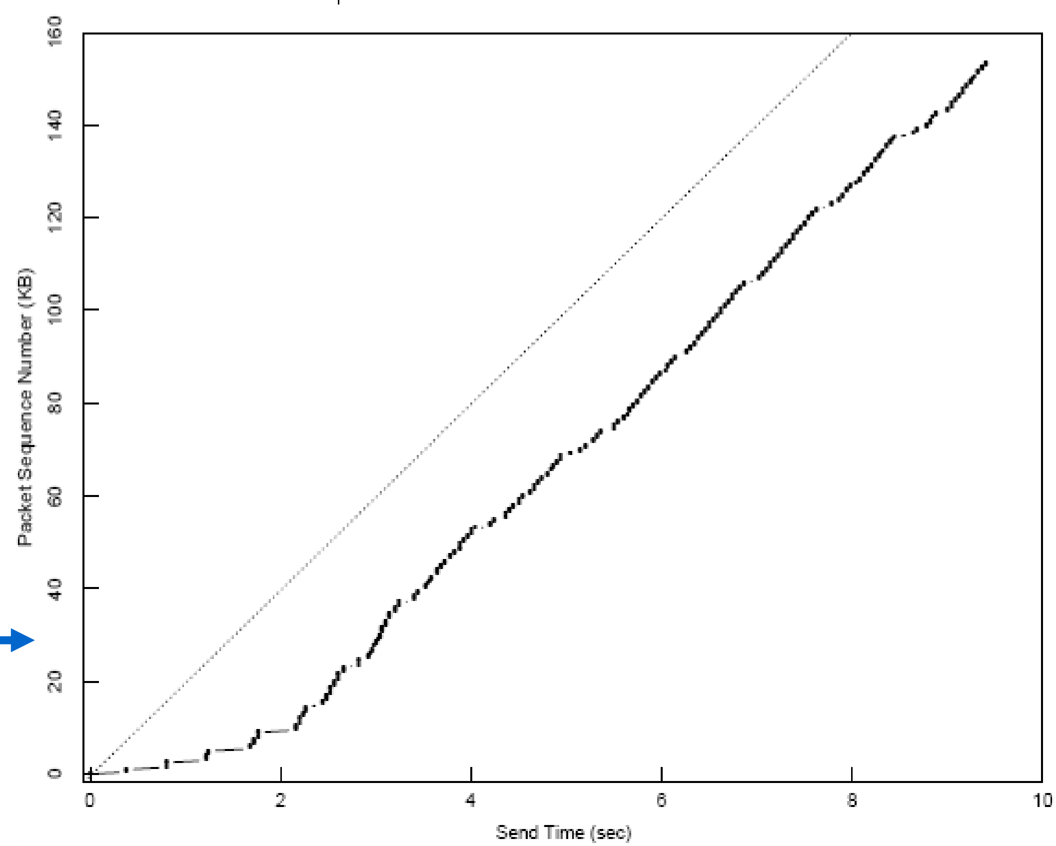
Slow start: increase by number of bytes ACKed. Effectively exponential increase.

Avoidance: At most, increase by 1 segment per RTT. Effectively a linear increase.

Why Slow Start?



Without slow start; large bursts but many retransmissions

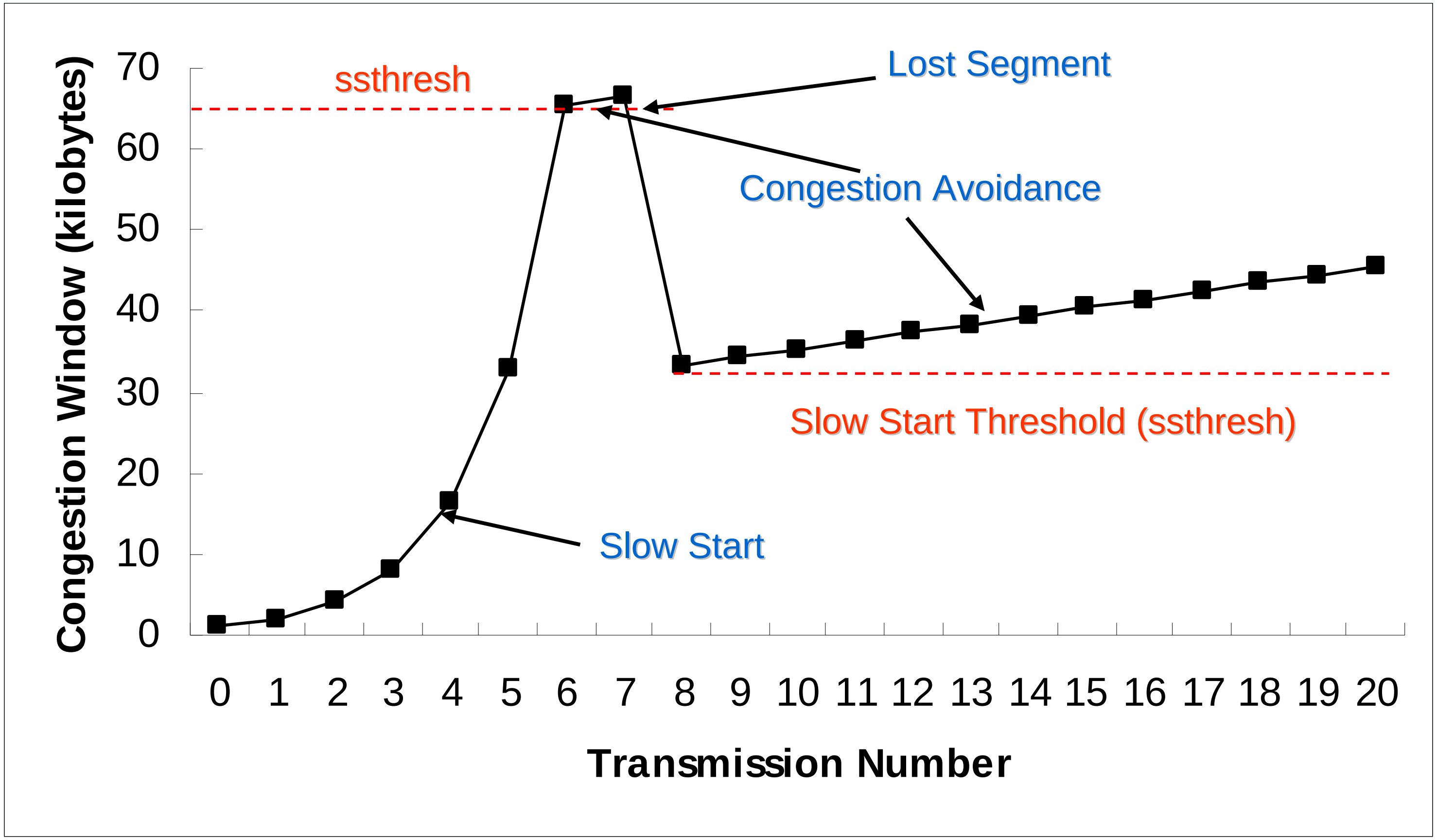


With slow start; no retransmissions

Reaction to Congestion

- Errors (segment loss) is taken as congestion indication
- Response to congestion event:
  - TCP Tahoe (1988)
    - Set ssthresh to half current congestion window
    - Set congestion window to 1 segment
    - Re-start slow start phase
  - TCP Reno (1990) – Fast Recovery
    - After Fast Retransmit, set ssthresh and congestion window to half current congestion window
    - Enter congestion avoidance phase
    - Sender retransmits at most 1 dropped packet per RTT
  - TCP NewReno (1995)
    - Only half congestion window once when multiple segments lost from transmitted window
      - Packets 1-10 are sent; 4, 6 and 7 lost
      - Congestion window halved when 4 retransmitted
      - Congestion window unaltered when 6 and 7 retransmitted

TCP Reno Congestion Control



TCP Versions and Options

- TCP RFC 793 (1981)
  - Reliability (sequence numbers), Flow control (receiver window), Connection management
- TCP Tahoe (1988)
  - Adds Slow Start, Congestion Avoidance, Fast Retransmit
- TCP Reno (1990)
  - Adds Fast Recovery
- TCP NewReno (1995)
  - Only halves congestion window once
- Other Options:
  - Selective Acknowledgement (SACK)
  - TCP Vegas