3 Flow Control

TCP Flow Control

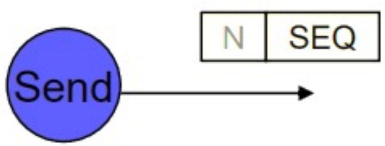
- TCP uses a modified version of the sliding window

- In acknowledgements, TCP uses the "Window size" field to tell the sender how many bytes it may transmit

- TCP uses bytes, not packets, as sequence numbers

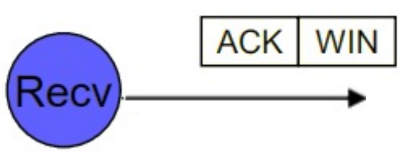
TCP Flow Control *(cont'd)*

Important information in TCP/IP packet headers



Number of bytes in packet (N)

Sequence number of first data byte in packet (SEQ)



ACK bit set

Sequence number of next expected byte (ACK)

Window size at the receiver (WIN)

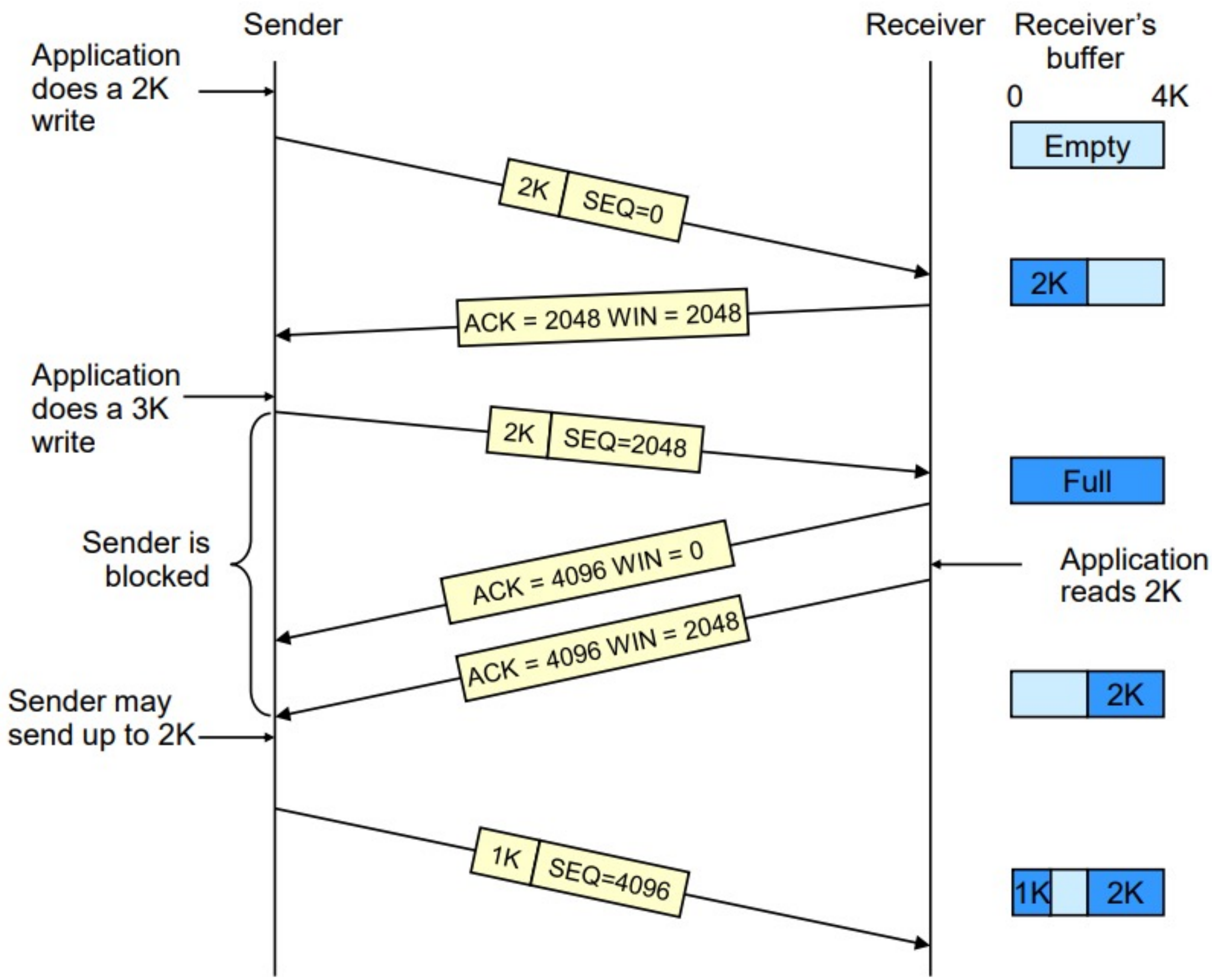
TCP Flow Control (cont'd)

Number of bytes in packet (N)

Sequence number of first data byte in packet (SEQ)

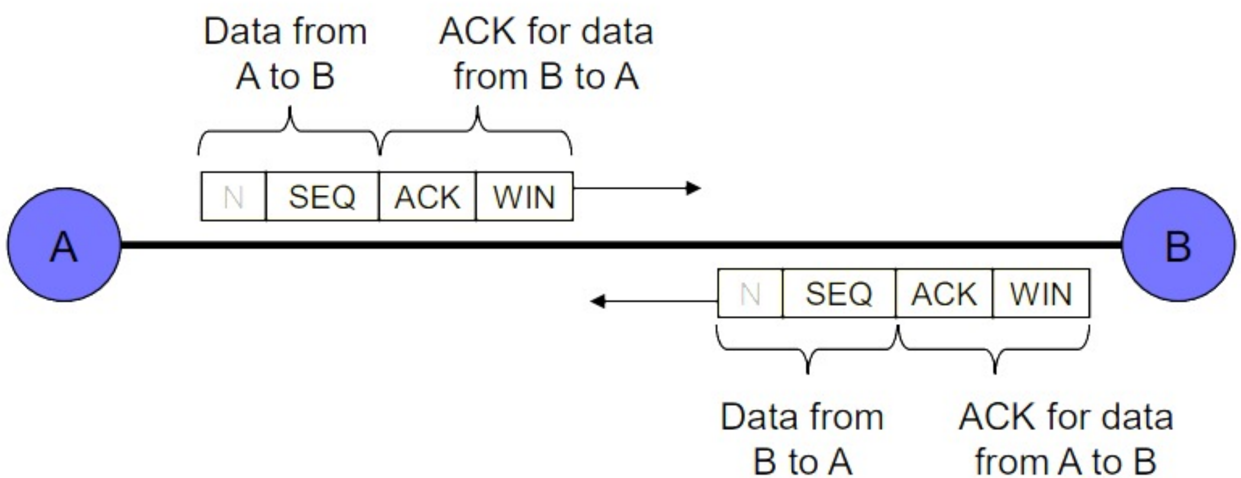
Sequence number of next expected byte (ACK)

Window size at the receiver (WIN)



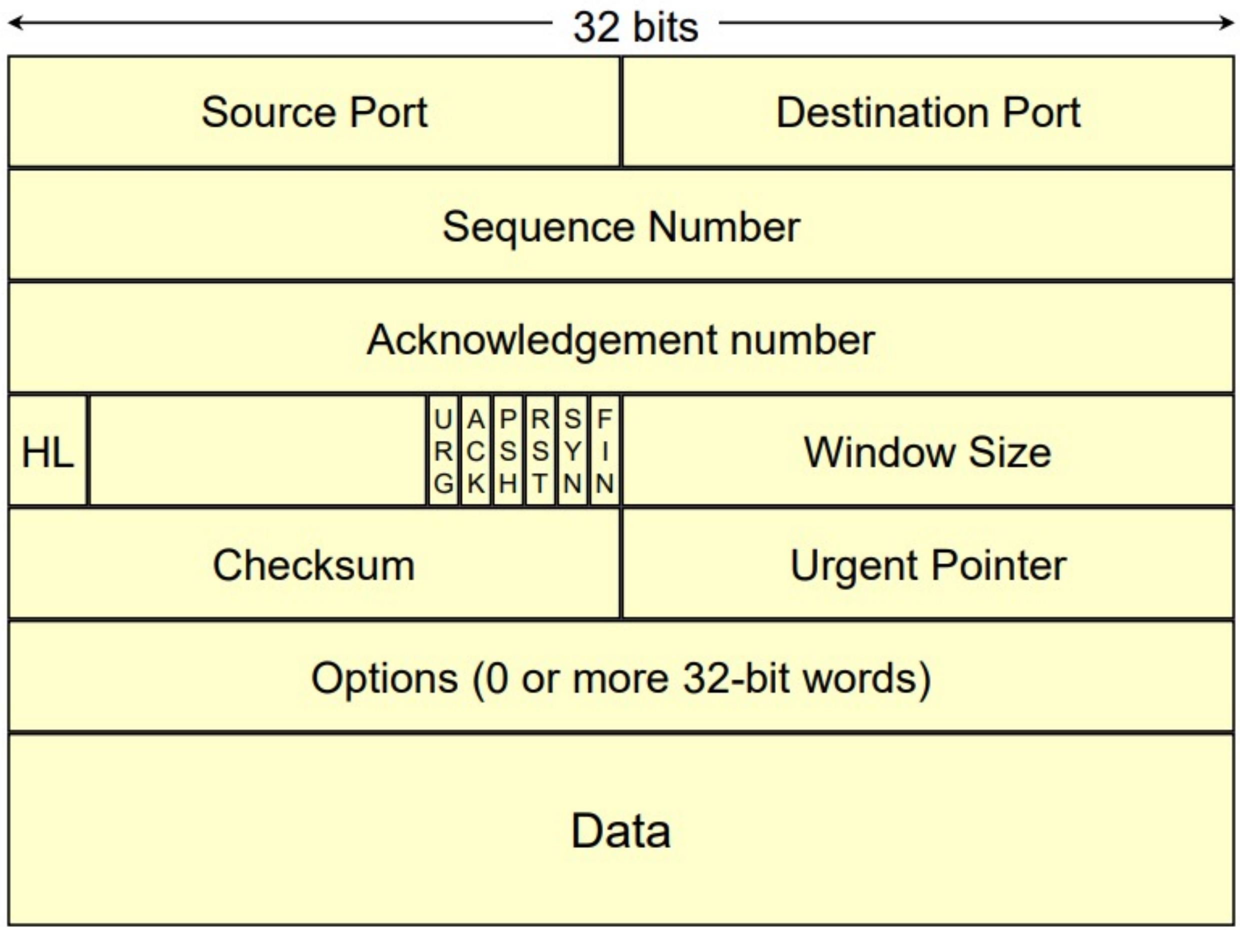
TCP Flow Control (cont'd)

Piggybacking: Allows more efficient bidirectional communication



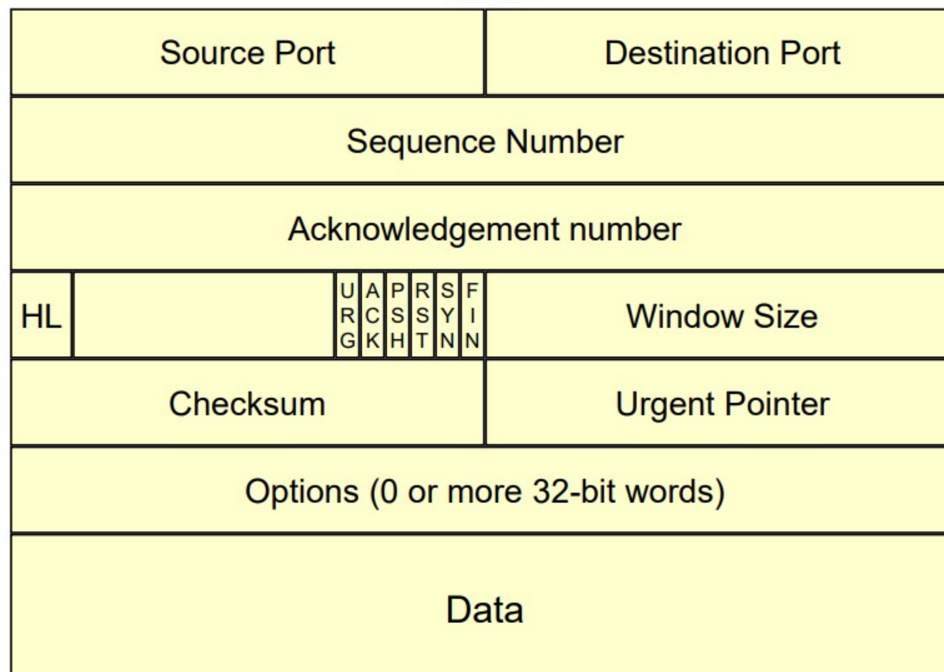
4 TCP Frame Format

TCP Header Format



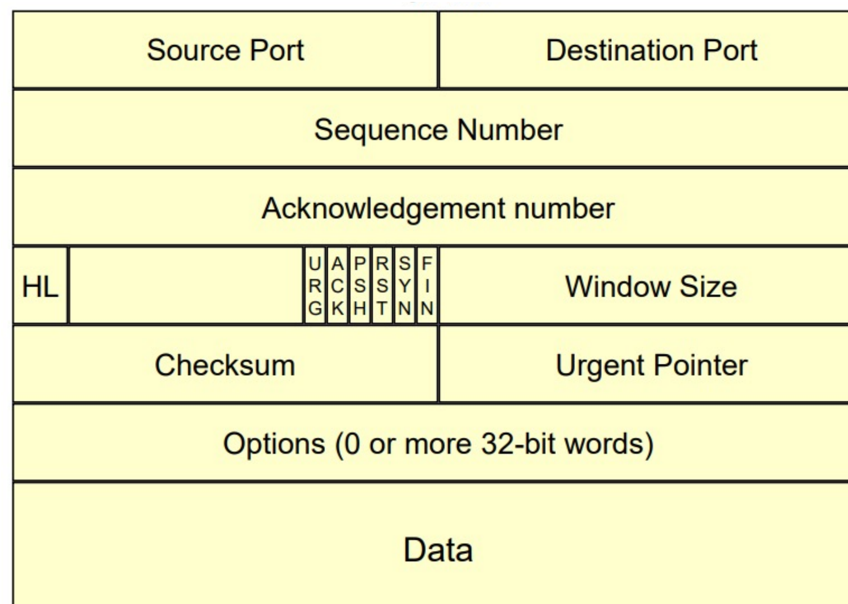
TCP Header Fields

- Source & Destination Ports
  - 16 bit port identifiers for each packet
- Sequence number
  - The packet's unique sequence ID
- Acknowledgement number
  - The sequence number of the next packet expected by the receiver



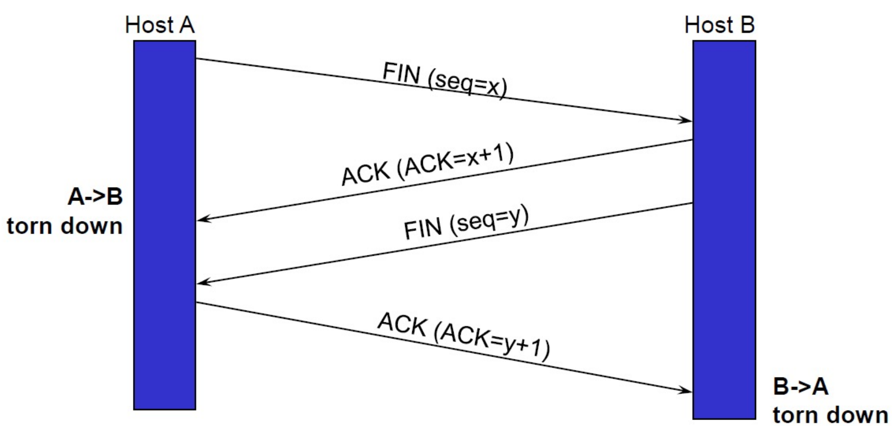
TCP Header Fields *(cont'd)*

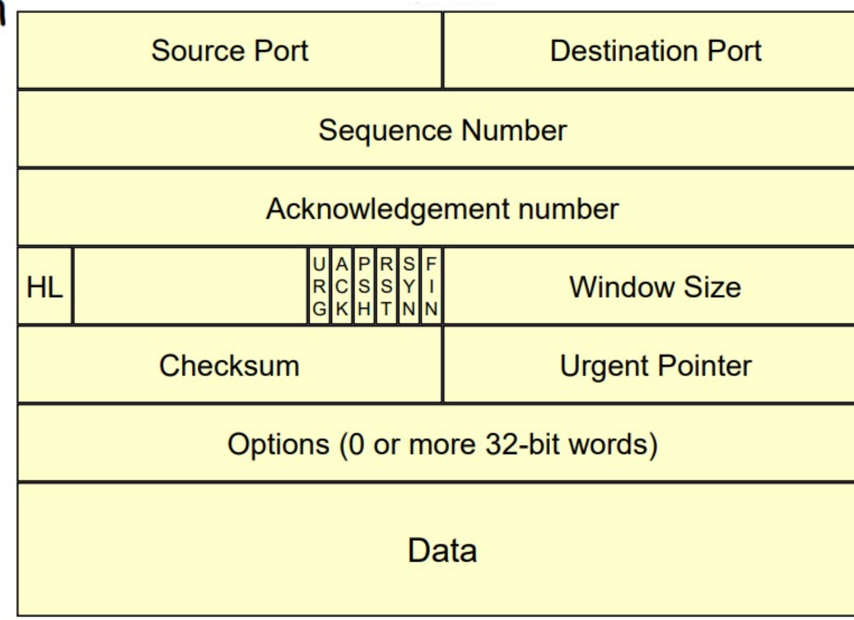
- Window size
  - Specifies how many bytes may be sent after the first acknowledged byte
- Checksum
  - Checksums the TCP header and IP address fields
- Urgent Pointer
  - Points to urgent data in the TCP data field



TCP Header Fields (cont'd)

- Header bits
  - URG = Urgent pointer field in use
  - ACK = Indicates whether frame contains acknowledgement
  - PSH = Data has been "pushed". It should be delivered to higher layers right away.
  - RST = Indicates that the connection should be reset
  - SYN = Used to establish connections
  - FIN = Used to release a connection





5 Congestion Control

Principles of Congestion Control

Congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)

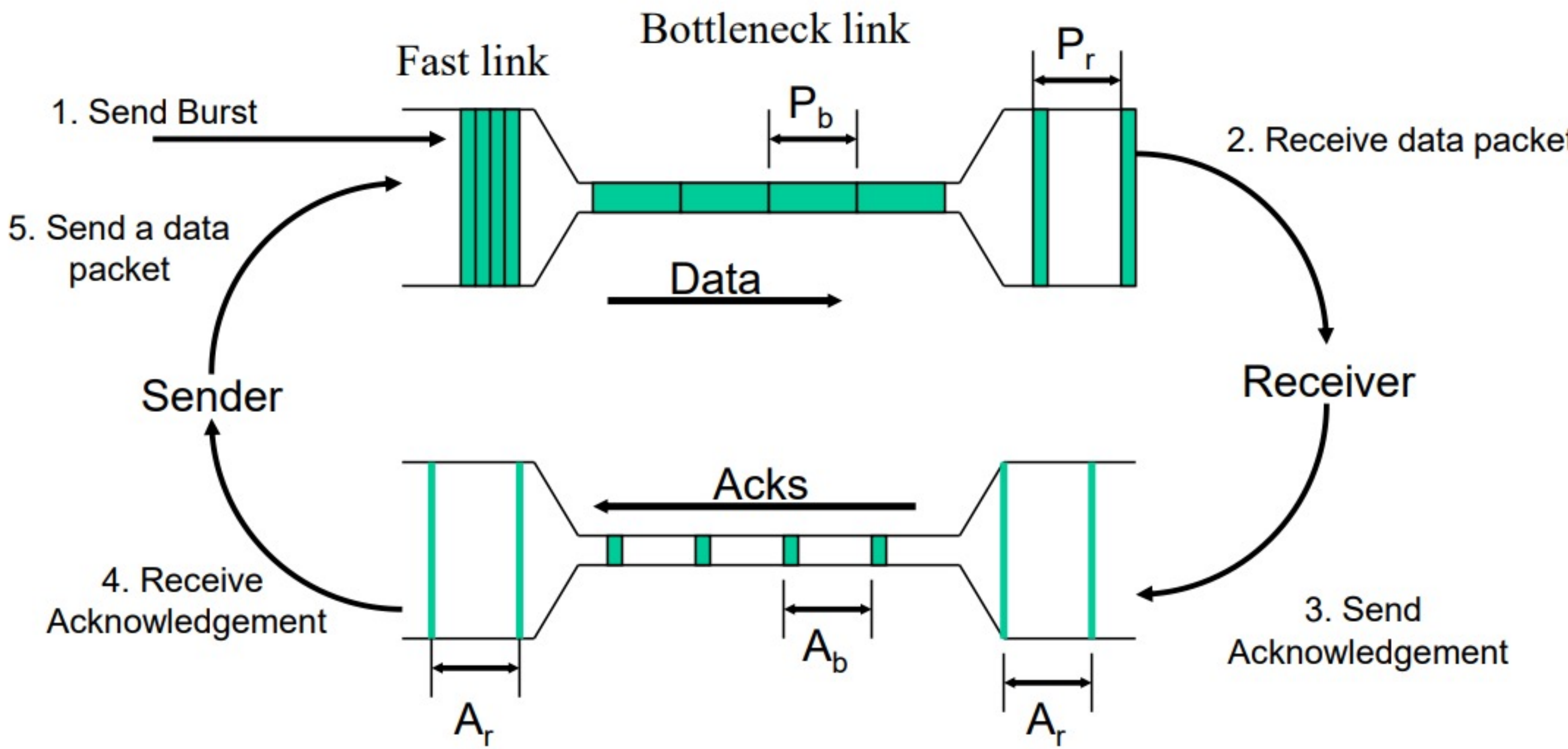
TCP Congestion Control

- Recall: Network layer is responsible for congestion control
- However, TCP/IP blurs the distinction
- In TCP/IP:
  - the network layer (IP) simply handles routing and packet forwarding
  - congestion control is done end-to-end by TCP

TCP Congestion Control

- Goal: fully (fairly) utilize the resource (bandwidth)
  - Don't over use - congestion
  - Don't under use - waste
- Goal: achieve self-clocking state
  - Even if don't know bandwidth of bottleneck
  - Bottleneck may change over time

Self-Clocking Model



Given: $P_b = P_r = A_r = A_b = A_r$ (in units of time)
Sending a packet on each ACK keeps the bottleneck link busy

TCP Congestion Window

- TCP introduces a second window, called the "congestion window"
- This window maintains TCP's best estimate of amount of outstanding data to allow in the network to achieve self-clocking
- Sending size = min(congestion control window, flow control window)

TCP Congestion Control

- Two phases to keep bottleneck busy (fully utilize the resource):
  - Increase the usage (window size) to keep probing the network
  - Decrease the usage when congestion is detected

TCP Slow Start

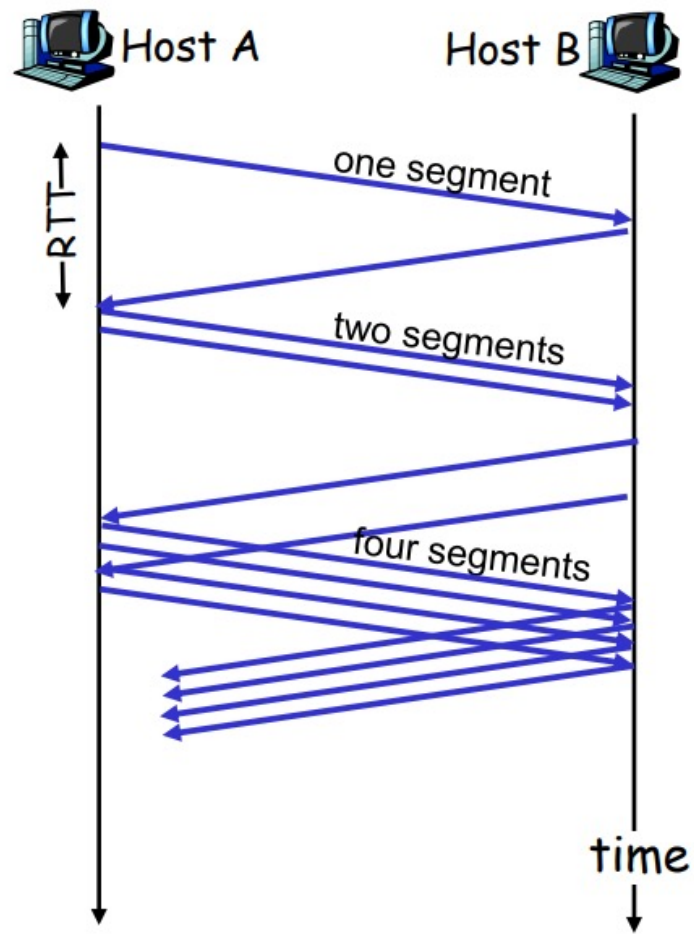
- When connection begins, `CongWin` = 1 MSS
  - Example: MSS = 500 bytes
- available bandwidth may be >> MSS/RTT
  - desirable to quickly ramp up to respectable rate
  - Increase exponentially until first loss

MSS - “maximum segment size”, the maximum size a TCP packet can be (including header)

TCP Slow Start (more)

- incrementing `CongWin` for every ACK received
- double `CongWin` every RTT

initial rate is slow but ramps up exponentially fast

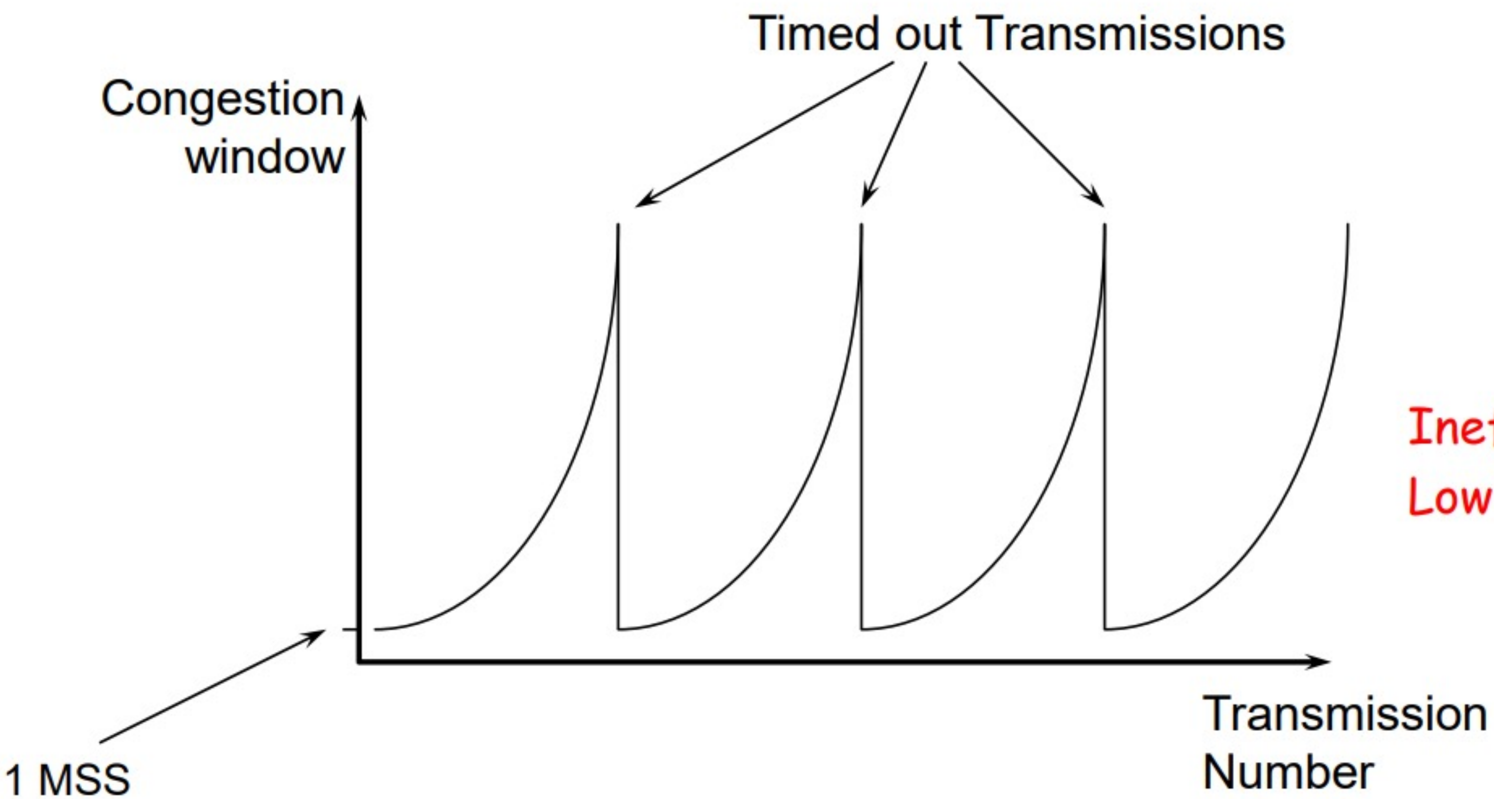


TCP Slow Start (cont'd)

- Congestion detection
  - Packet losses
  - Sender side Timeout

- Timeout
  - the congestion window is reduced to 1 MSS
  - everything starts over

TCP Slow Start *(cont'd)*



Inefficient,
Low throughput

TCP Linear Increase

- Don't push the network too fast

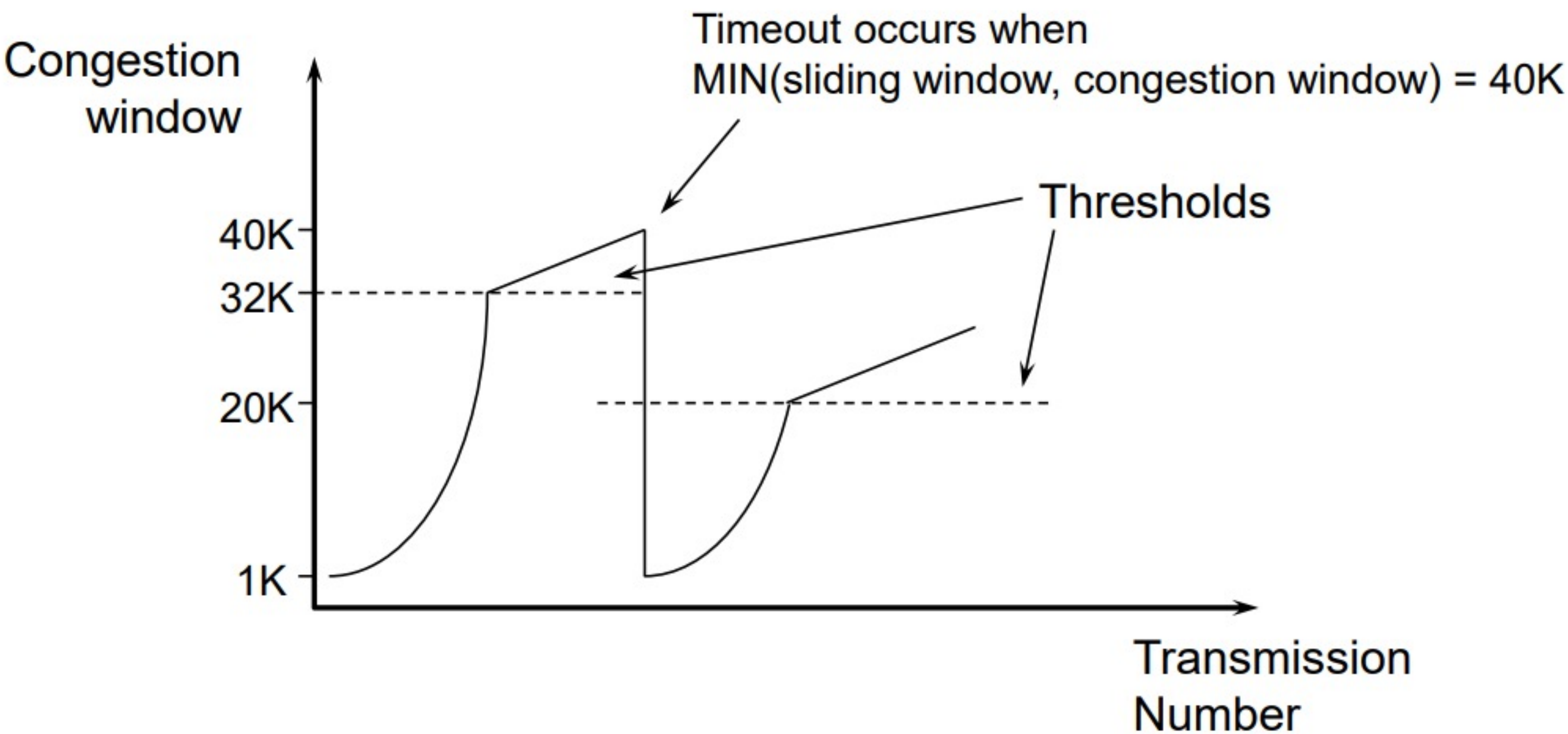
- Slow start (exponential)
  -> Threshold -> linear increase

TCP Linear Increase Algorithm

- Algorithm:
  - Start the threshold at 64K
  - Slow start
  - Once the threshold is passed
    - For each ack received, cwnd = cwnd + (mss*mss)/cwnd
      - 1 MSS for each congestion window of data transmitted
    - Timeout
      - reset the congestion window size to 1 MSS
      - Set threshold to max(2*mss,1/2 of MIN(sliding window, congestion window))

TCP Linear Increase Threshold Phase

Example: Maximum segment size = 1K
Assume thresh=32K



TCP Congestion Control

- Can we do better at detecting congestion than using timeout?

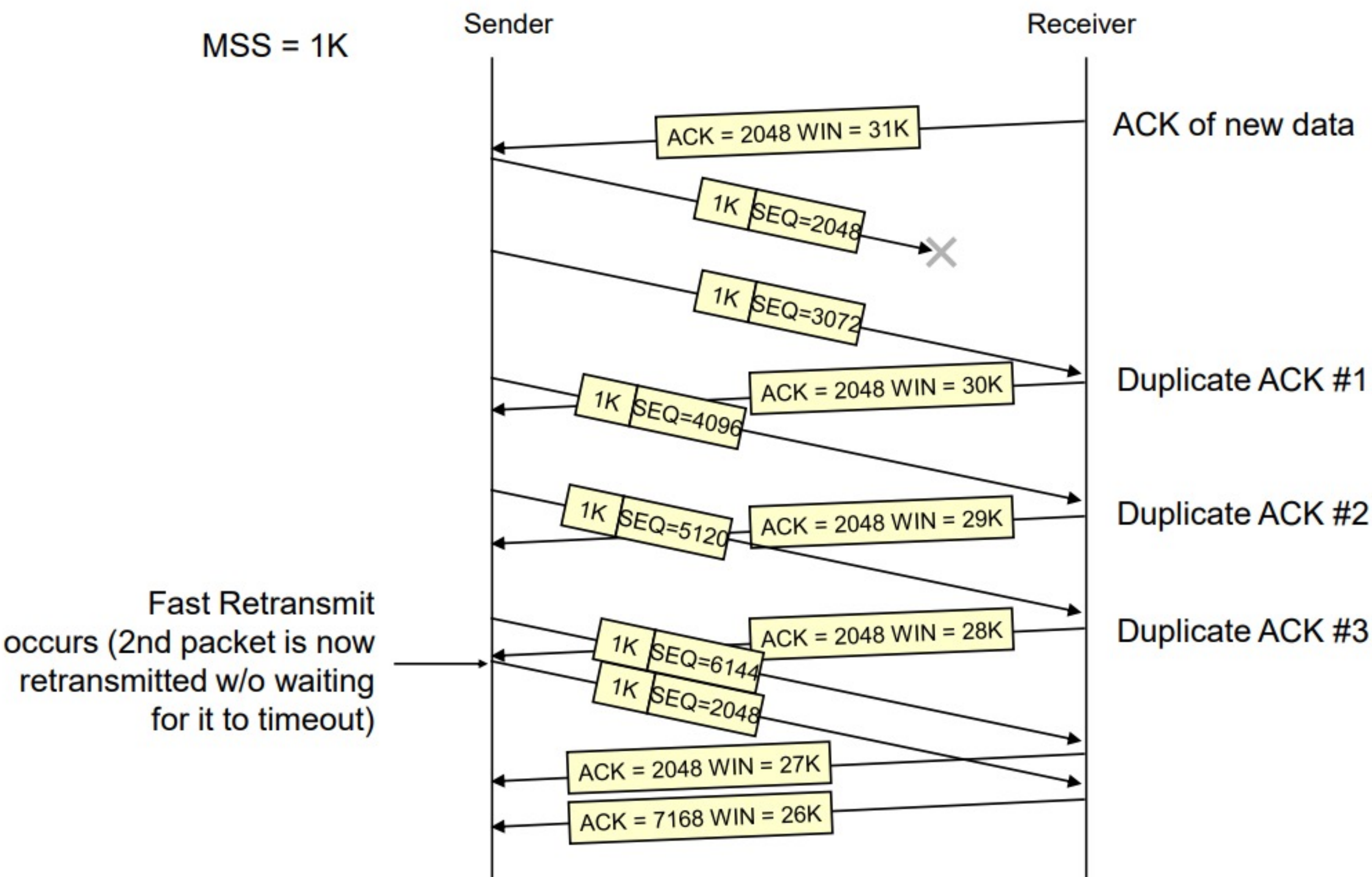
- Receiver send duplicate ack for out-of-order packets
  - Possible loss?

TCP Fast Retransmit

- Idea: When sender sees 3 duplicate ACKs, it assumes something went wrong
- The packet is immediately retransmitted instead of waiting for it to timeout

TCP Fast Retransmit

Example



TCP Recap

- Timeout Computation
  - Timeout is a function of 2 values
    - the weighted average of sampled RTTs
    - The sampled variance of each RTT
- Congestion control:
  - Goal: Keep the self-clocking pipe full in spite of changing network conditions
  - 3 key Variables:
    - Sliding window (Receiver flow control)
    - Congestion window (Sender flow control)
    - Threshold (Sender's slow start vs. linear mode line)

- Slow start
- Linear mode (Congestion Avoidance)

Algorithm Summary: TCP Congestion Control

- When `CongWin` is below `Threshold`, sender in slow-start phase, window grows exponentially.
- When `CongWin` is above `Threshold`, sender is in congestion-avoidance phase, window grows linearly.
- When a triple duplicate ACK occurs, `Threshold` set to max(`FlightSize/2,2*mss`) and `CongWin` set to `Threshold+3*mss`. (Fast retransmit, Fast recovery)
- When timeout occurs, `Threshold` set to max(`FlightSize/2,2*mss`) and `CongWin` is set to 1 MSS.

FlightSize: The amount of data that has been sent but not yet acknowledged.