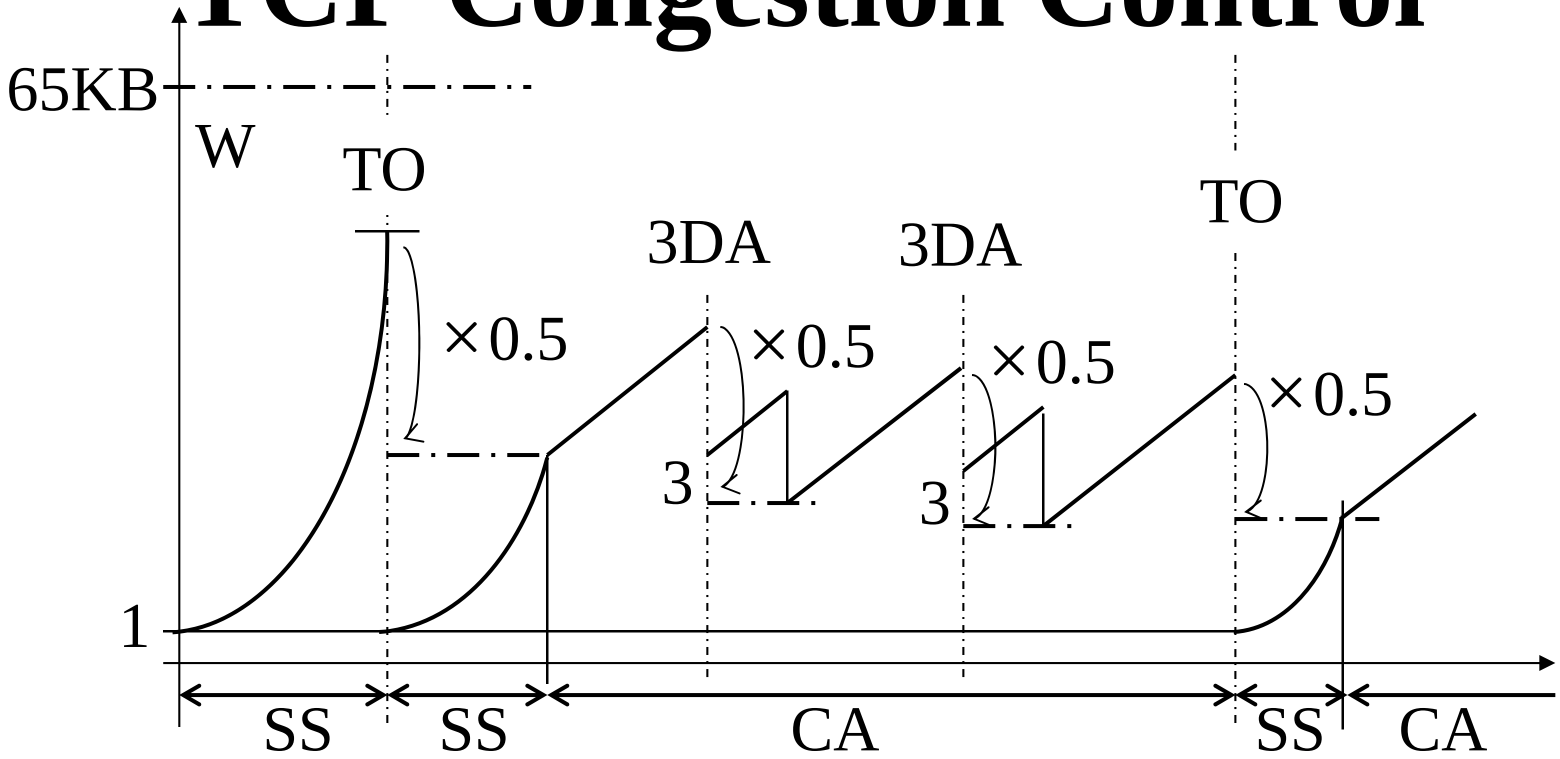
TCP Congestion Control



TCP’s Congestion Window Maintenance

TCP maintains a congestion window (cwnd), based on packets

Sender’s window is limited to

MIN(receiver’s window, cwnd)

Maintenance policy:

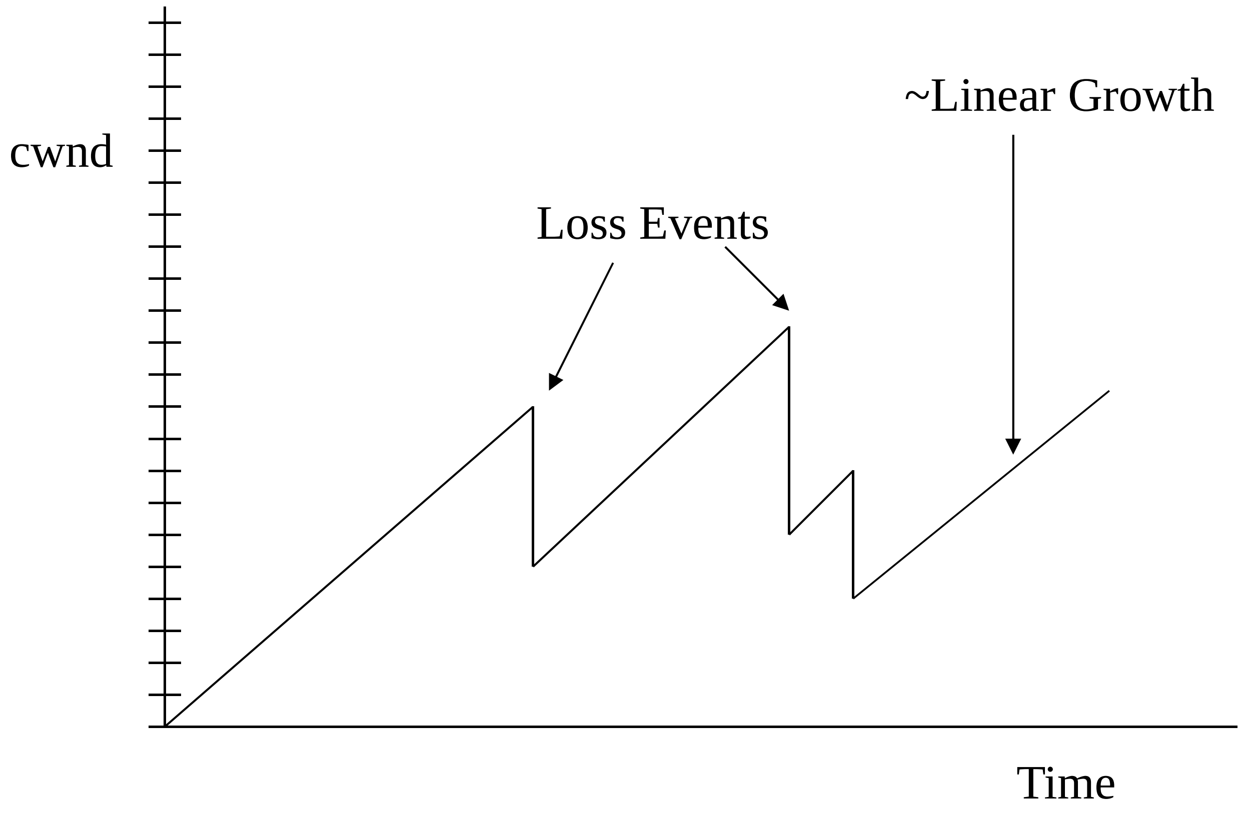
- On congestion signal, multiplicative decrease
- On success, additive increase

Additive increase/multiplicative decrease produces stability

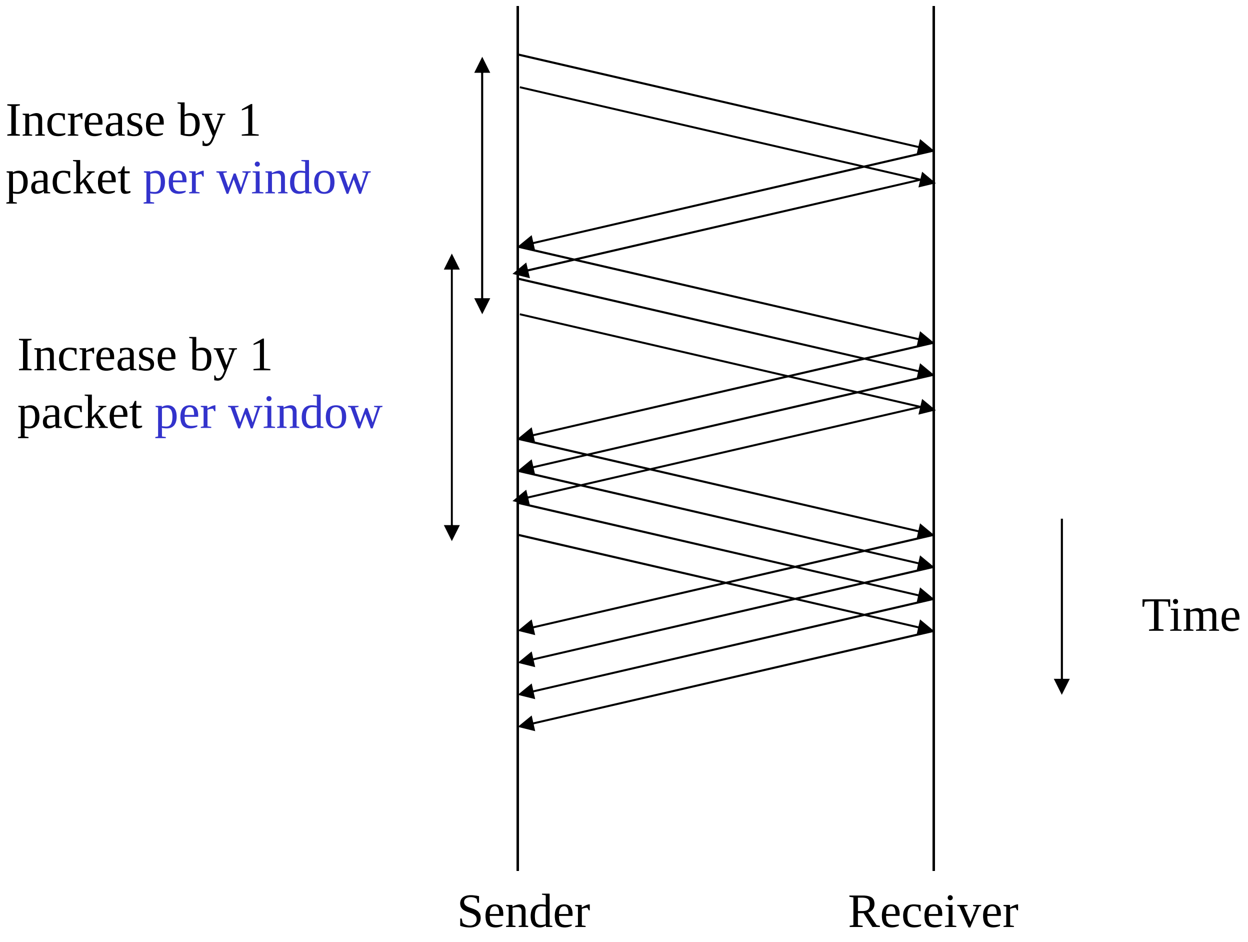
Window Increase/Decrease

- TCP Congestion Avoidance:
  - Use packet loss as indicator of congestion
  - On loss, divide cwnd by 2
  - On successful ACK, increase cwnd by 1/cwnd
- Results in window growth of 1 packet for each window’s worth of ACKs [near linear]

TCP Congestion Avoidance



TCP Congestion Avoidance



Congestion Avoidance

TCP Congestion Avoidance makes sense when the connection is operating near capacity (in steady-state, but searching for any capacity change)

What about when a connection starts up, or there has been a long pause?

Need a way to get to equilibrium

TCP Slow Start

- Slow-start is a TCP behavior used to get to packet equilibrium
- Slow-start increases the congestion window exponentially, rather than linearly
- Why called “slow-start” then?
  - It is considerably slower than the start based only on the receiver’s advertised window

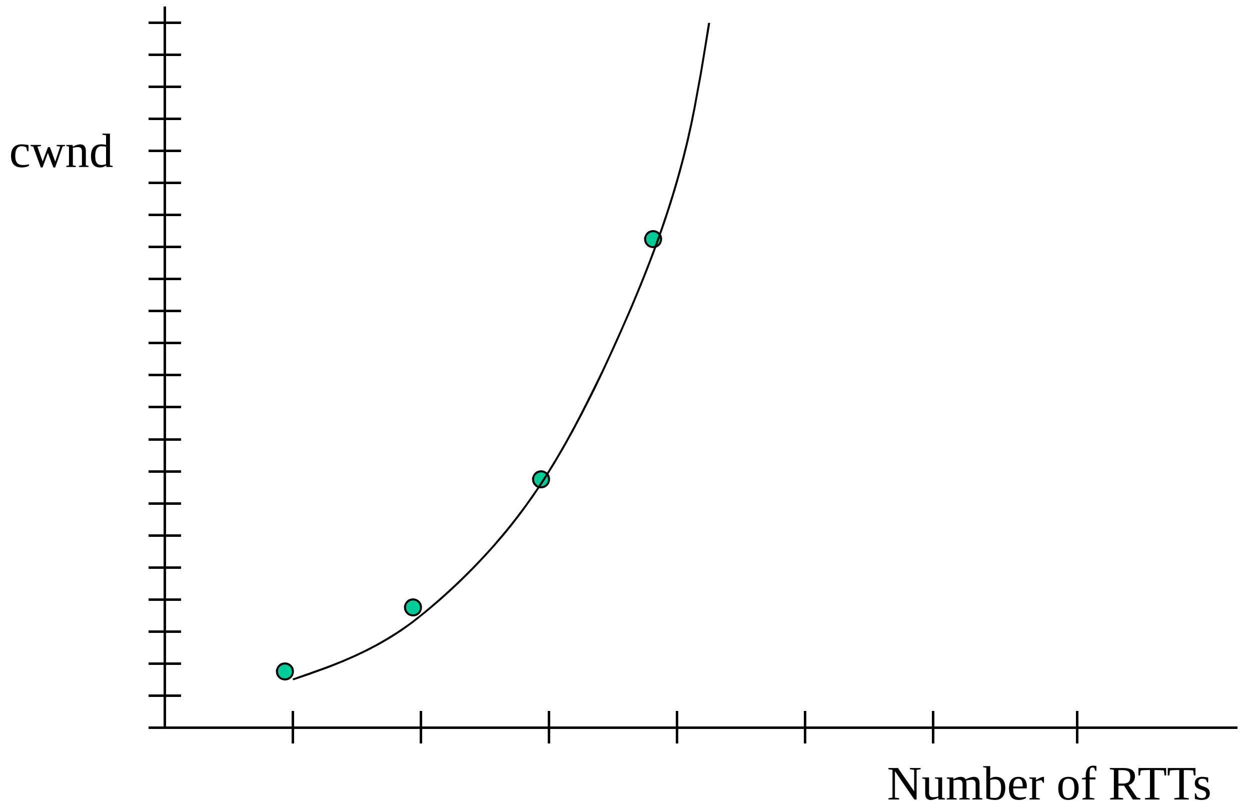
TCP Slow Start

For each ACK received, increase the congestion window by 1

Results in cwnd pattern of: 1, 2, 4, 8, 16, 32, …

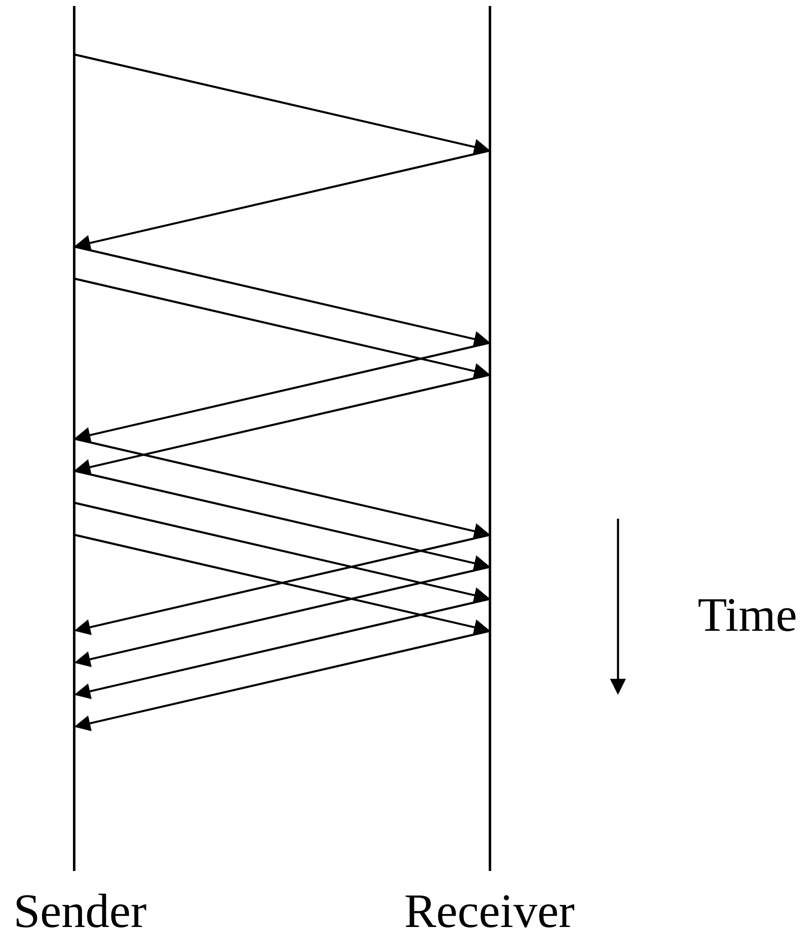
- Takes time proportional to $\log_2 W$ to reach window of W, [longer if ACKs delayed]

TCP Slow Start



TCP Slow Start

Increase by 1
packet per ACK



TCP Congestion Behaviors

Two algorithms:

- Slow-start: getting to equilibrium
- Congestion avoidance: searching for new available bandwidth in path (and reacting to congestion)

The two behaviors are mutually exclusive for any single point in time, but each TCP implements both:

- Establish an operating point to switch between the two algorithms (*ssthresh*)

Slow-Start Threshold (ssthresh)

Need a way to determine whether the TCP should do slow-start or congestion avoidance

New variable (ssthresh):

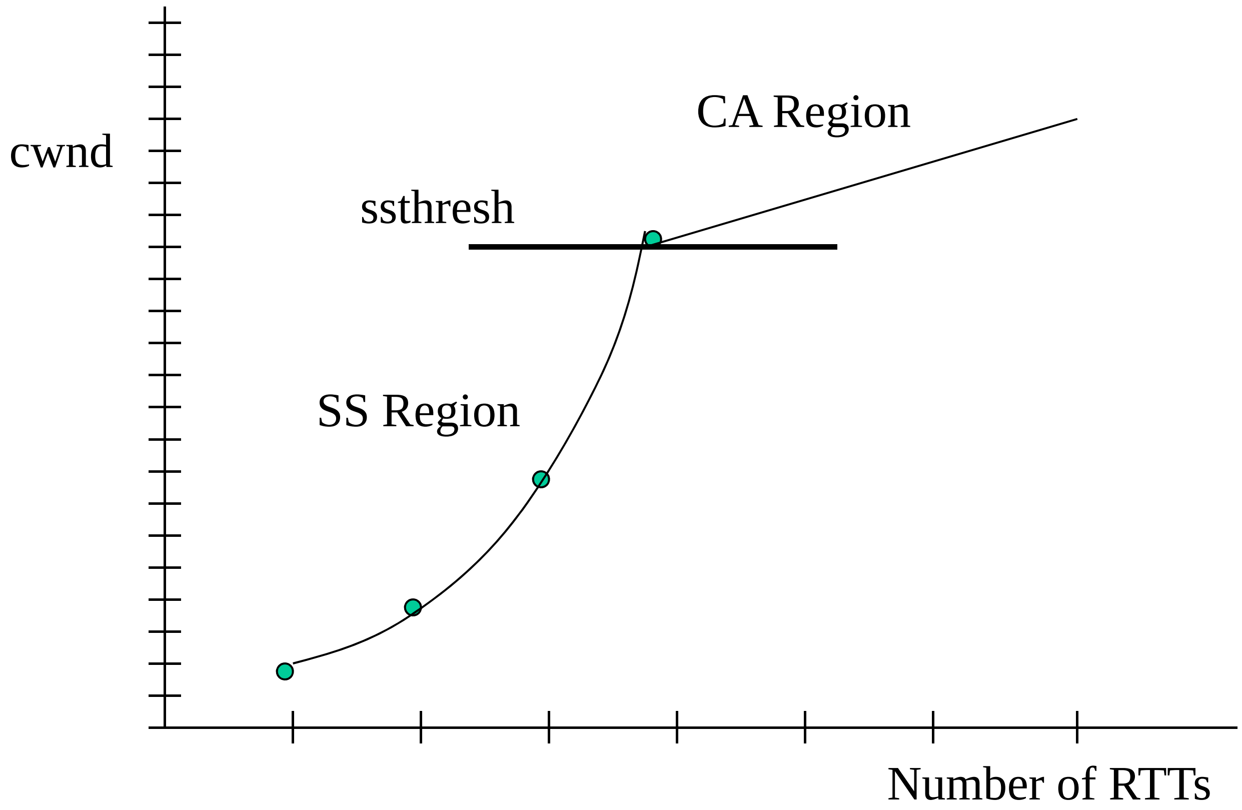
if cwnd ≤ ssthresh, do slow-start

if cwnd > ssthresh, do congestion avoidance

ssthresh is initialized to a large value, after a congestion signal, cwnd is divided in half, and ssthresh is set to cwnd

(can lead to overshoot at start of connection)

TCP Slow-Start and Congestion Avoidance



ssthresh and cwnd maintenance

Congestion window is normally updated on congestion indications (packet drops), and grows linearly if above ssthresh

ssthresh is reset to cwnd after it is reduced to keep a marker of the last operating point

When does the TCP ever enter slow-start after a connection has started? (hint: if we are doing very badly)

Detecting Loss with TCP

TCP uses lost packets as indicators of congestion

Two methods

- Timer expiring
- Fast retransmit

Fast retransmit:

- Because of cumulative ACK, out-of-order data received at receiver may generate *duplicate ACKs* (“dupacks”)

Duplicate ACKs

To arrange out-of-order segments, TCP responds immediately with one ACK per packet:

Receiver gets: 5, 6, 7, 8, 10, 11, 12, 13

ACKs: 6, 7, 8, 9, 9, 9, 9, 9 [4 dupacks]

Actual Sequence_Num and Ack_Num are discrete due to the byte stream

Provides a hint to sender that packet 9 is probably missing at receiver and that 4 packets have arrived after 8 arrived

[think about retransmit!]

Fast Retransmit

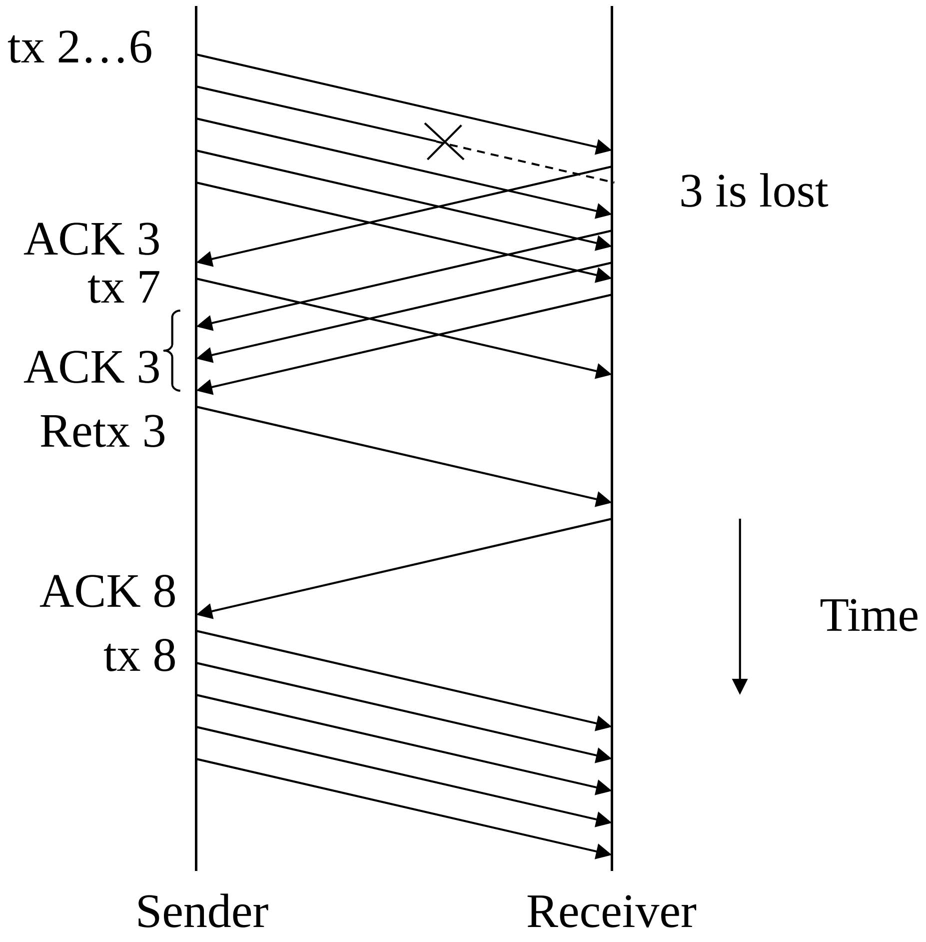
Heuristic at sender to trigger retransmissions without timeouts

To avoid timeout due to delayed packet, look for 3 dupacks

So, on 3rd dupack for packet n, retransmit n, and send more if send window allows

If only one packet lost, fills receiver's "hole", resulting in cumulative ACK for top of window

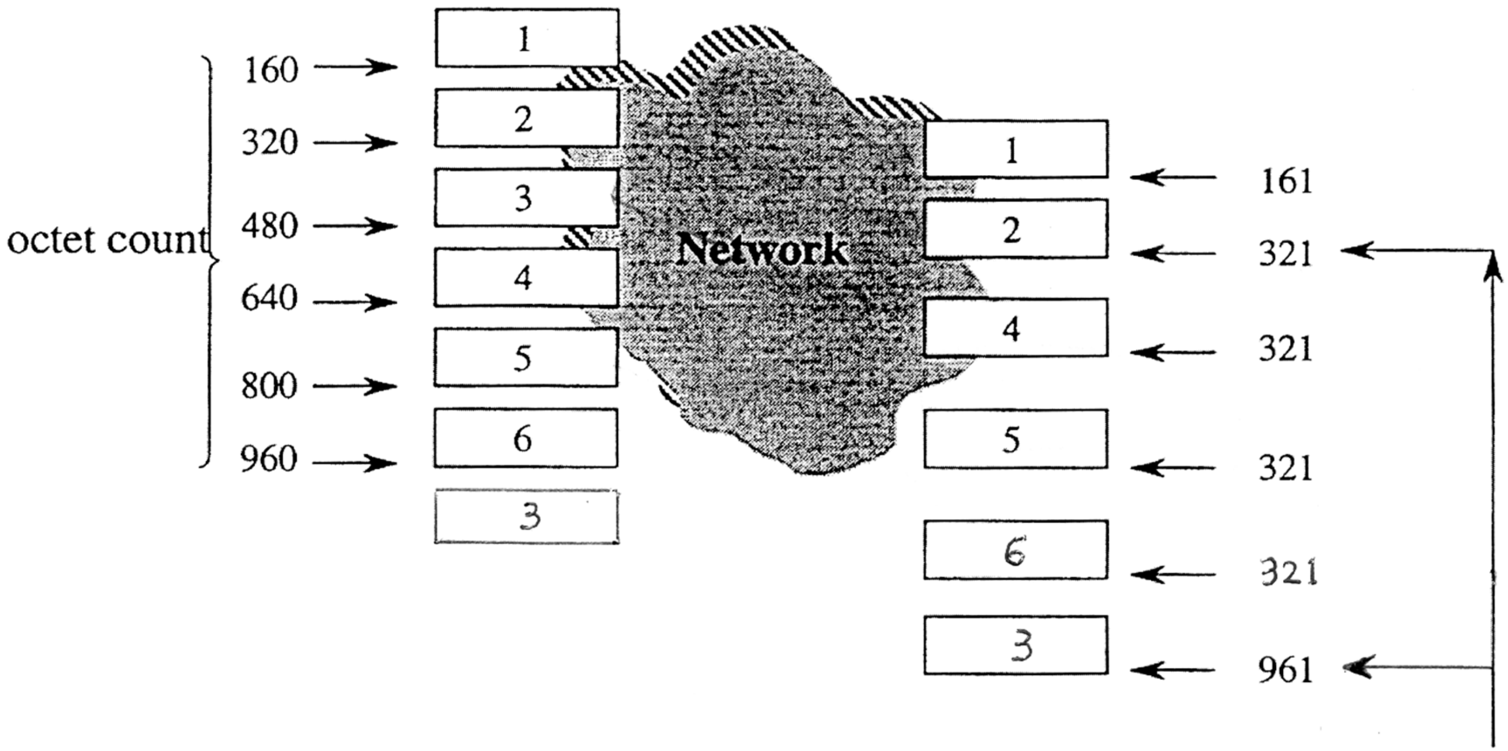
Fast Retransmit Example



TCP Ack and Fast Retransmissions

Transmit Site at Launch Time

Receive Site Some Seconds Later



Fast RTX Observations

Fast retransmit can repair modest packet loss without requiring a retransmission timer to expire

Because it requires 3 dupacks to fire, doesn't work so well with small windows (because there won't be enough ACKs generated at the receiver)

With large numbers of dropped packets, similar problem (not enough ACKs)

Congestion Action on Loss

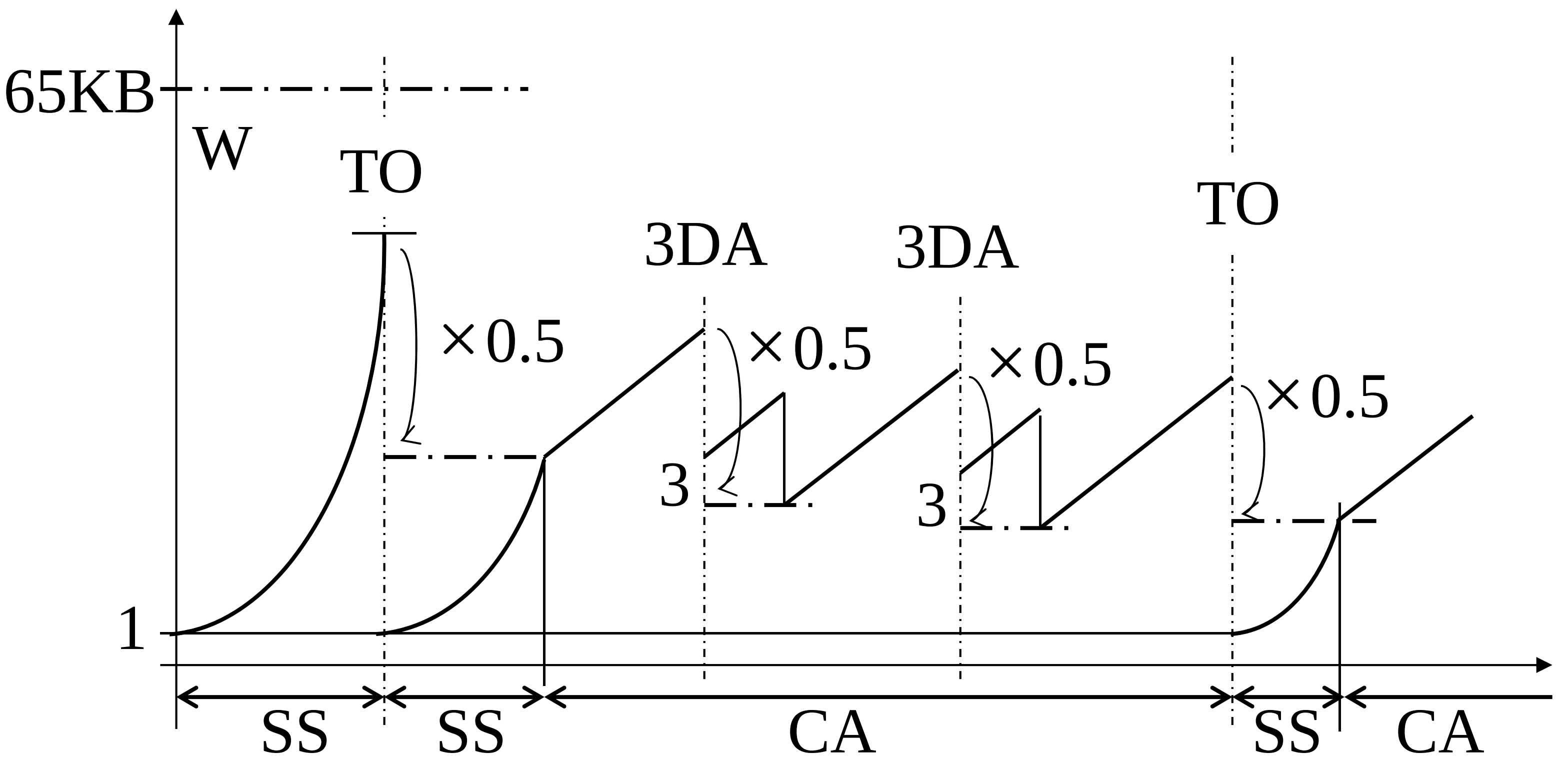
TCP has different behaviors, depending on the way it detects loss (RFC2581)

- RTX timer expires:
  - ssthresh = MAX(MIN(win,cwnd)/2, 2)
  - cwnd = 1 [or other IW] (initiates slow-start)
- Fast retransmit (fast recovery):
  - ssthresh = MAX(MIN(win,cwnd)/2, 2)
  - cwnd = ssthresh + 3
    - each additional dupack increments cwnd by 1
    - fast recovery
  - cwnd = ssthresh on new ACK

Refinements: Summary

Actual window = min{RAW - OUT, W}

where Out = Last sent - Last ACKed



Summary: TCP Congestion Behavior

Slow-start:

- When: new connection, after idle time, after RTX timer expires
- How: set cwnd=1, grow window exponentially
- Why: searches quickly for operating point

Congestion avoidance:

- When: normal operations, fast RTX/recovery
- How: divide operating point in 1/2 after loss
- Why: searches slowly for new bandwidth