Congestion Control

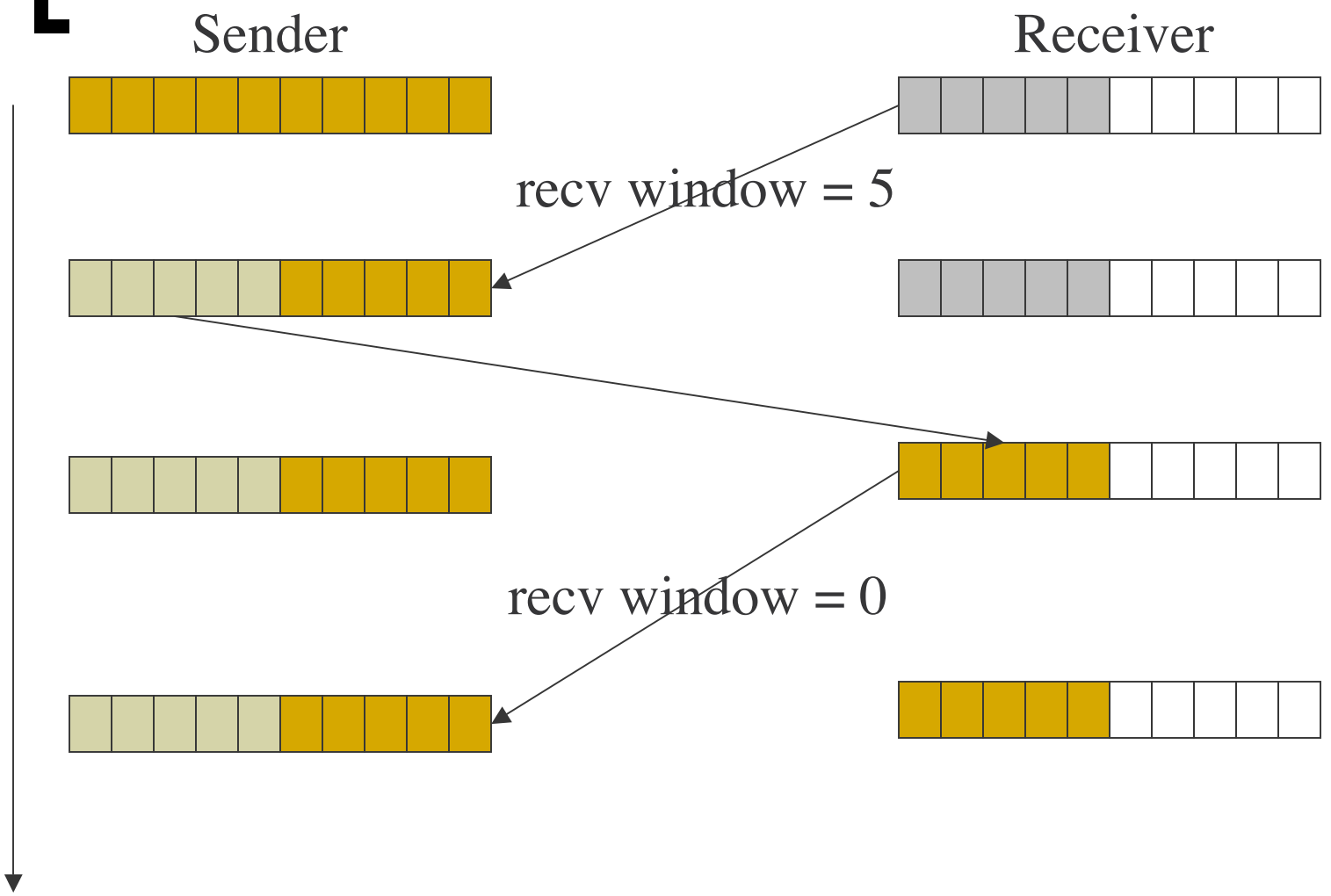
Congestion Control

- What is congestion?
- Methods for dealing with congestion
- TCP congestion control

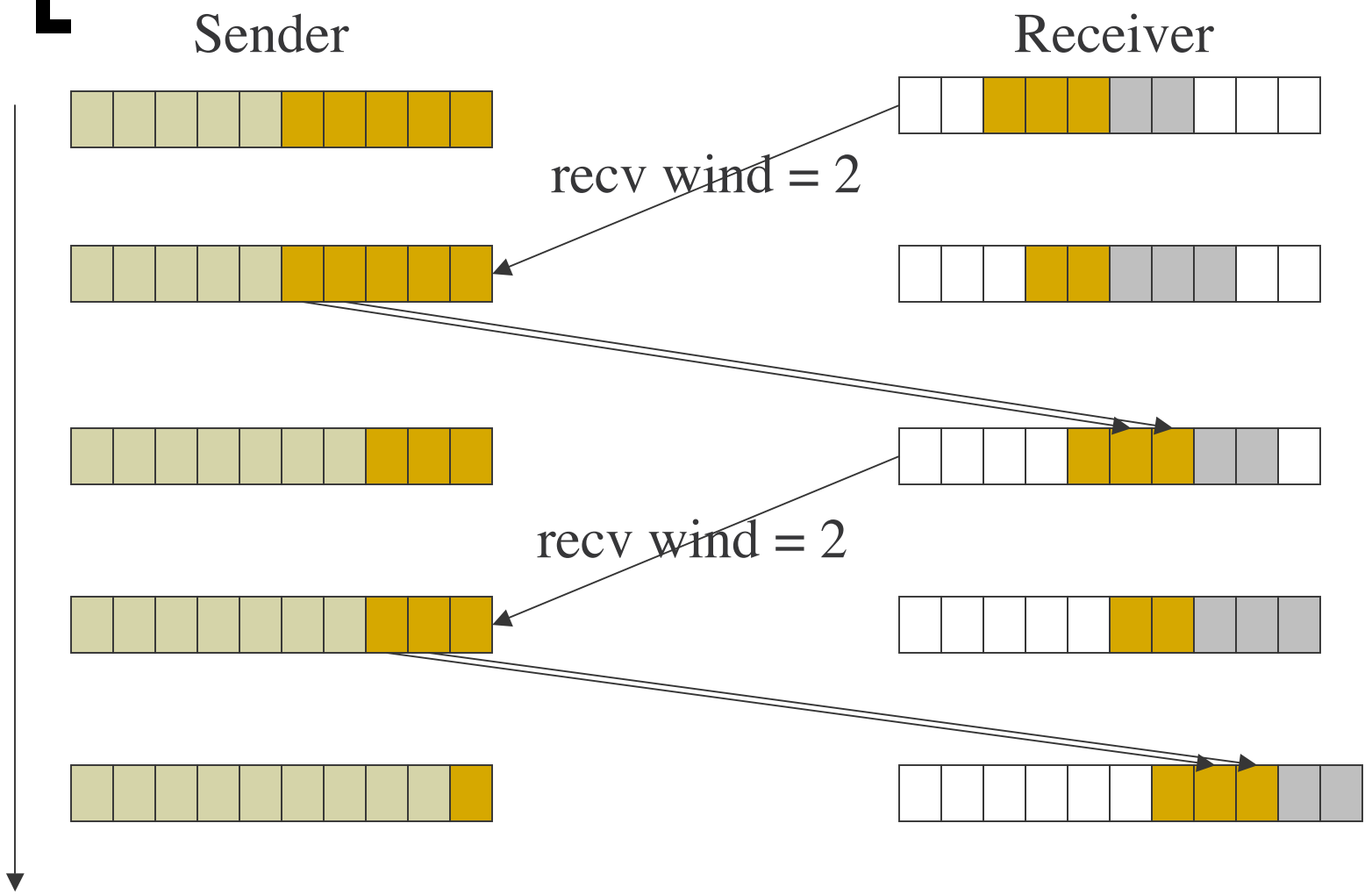
Review of Flow Control

- Receiver advertises window in ACK packets
  - Receiver window size = amount of free space in receive buffer
- Sender will limit number of unACK'ed packets to receiver window
  - Invariant: every packet that arrives at receiver can be buffered

Flow Control



Flow Control, Cont'd



TCP Flow Control

- Problem: Slow application reads data in tiny pieces
  - Receiver advertises tiny window
  - Sender fills tiny window
  - Known as silly window syndrome
- Solution
  - Advertise window opening only when MSS or 1/2 of buffer is available
  - Sender delays sending until window is MSS or 1/2 of receiver's buffer (estimated)

TCP Flow Control

- Problem: Slow receiver application
  - Advertised window goes to 0
  - Sender cannot send more data
  - Receiver application finally reads some data, receiver window opens up
  - How does the sender learn this?
- Solution
  - Sender periodically sends 1-byte segment, ignoring advertised window of 0
  - Eventually window opens
  - Sender learns of opening from next ACK of 1-byte segment

TCP Flow Control

- Problem: Application delivers tiny pieces of data to TCP
  - Example: telnet in character mode
  - Each piece sent as a segment, returned as ACK
  - Very inefficient
- Solution
  - Delay transmission to accumulate more data
  - Nagle's algorithm
    - Send first piece of data
    - Accumulate data until first piece ACK'd
    - Send accumulated data and restart accumulation
    - Not ideal for some traffic (e.g. mouse motion)

TCP Bit Allocation Limitations

- Sequence numbers vs. packet lifetime
  - Assumed that IP packets live less than 60 seconds
  - Can we send 2^{32} bytes in 60 seconds?
  - Less than an STS-12 line
- Advertised window vs. delay-bandwidth
  - Only 16 bits for advertised window
  - Cross-country RTT = 100 ms
  - Adequate for only 5.24 Mbps!

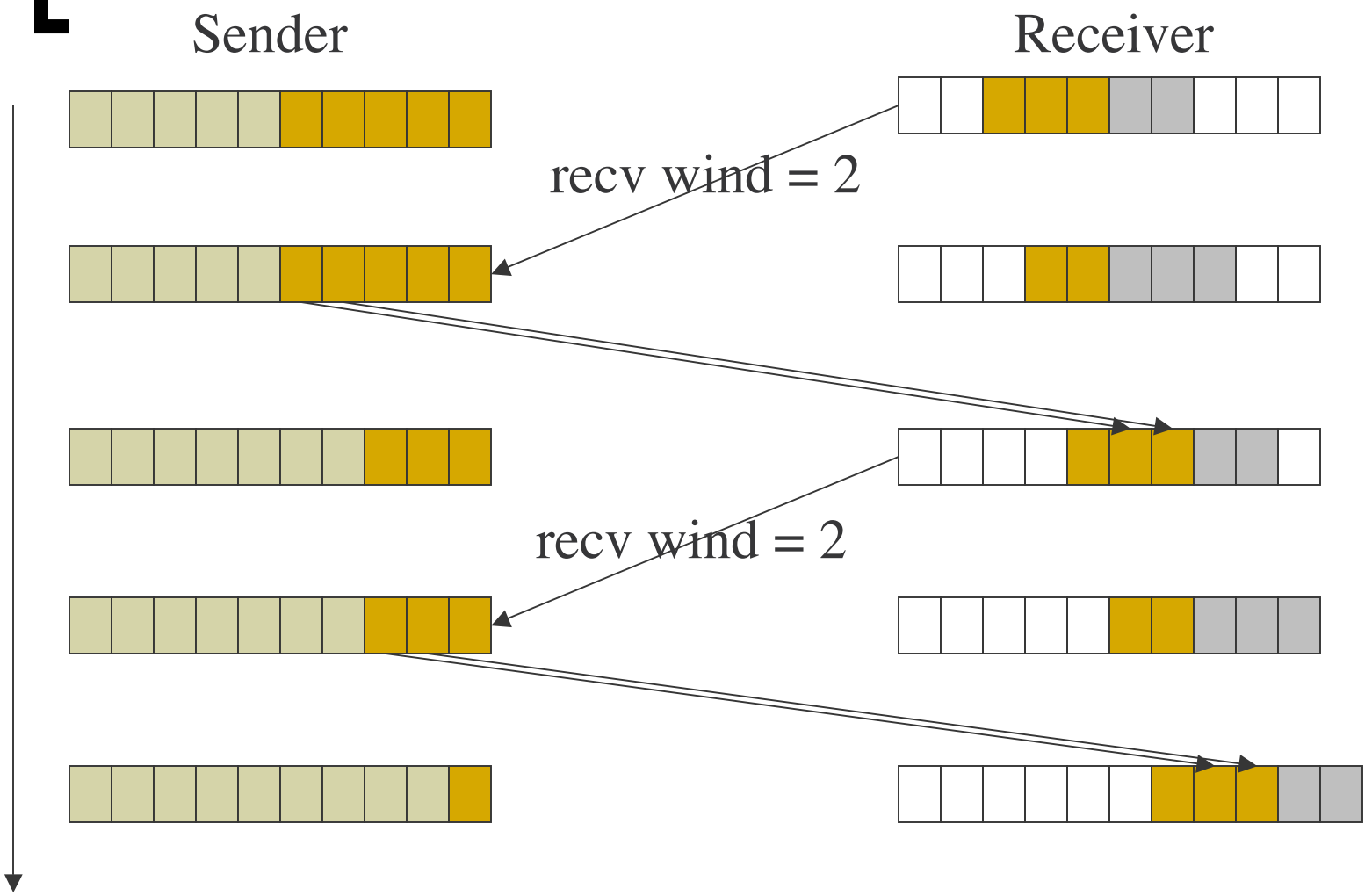
TCP Sequence Numbers – 32-bit

Bandwidth	Speed	Time until wrap around
T1	1.5 Mbps	6.4 hours
Ethernet	10 Mbps	57 minutes
T3	45 Mbps	13 minutes
FDDI	100 Mbps	6 minutes
STS-3	155 Mbps	4 minutes
STS-12	622 Mbps	55 seconds
STS-24	1.2 Gbps	28 seconds

TCP Advertised Window – 16-bit

Bandwidth	Speed	Max RTT
T1	1.5 Mbps	350 ms
Ethernet	10 Mbps	52.4 ms
T3	45 Mbps	11.6 ms
FDDI	100 Mbps	5.2 ms
STS-3	155 Mbps	3.4 ms
STS-12	622 Mbps	843 µs
STS-24	1.2 Gbps	437 µs

Flow Control, Cont'd



Flow Control Justification

- TCP flow control is *conservative*
  - Only send data if receiver is sure to have space for it
- Consider *aggressive* alternative
  - Send data optimistically, hoping receiver has space for it
  - If receiver can't buffer packet, simply discard
- Which is more efficient?

Efficiency Metrics

- Transmission speed
  - Get as many bytes from sender to receiver as quickly as possible
- Network utilization
  - Maximize “goodput”
  - Fraction of bytes spent on delivering new, useful data
- E.g. delayed ack hurts transmission speed, but improves network utilization

Congestion

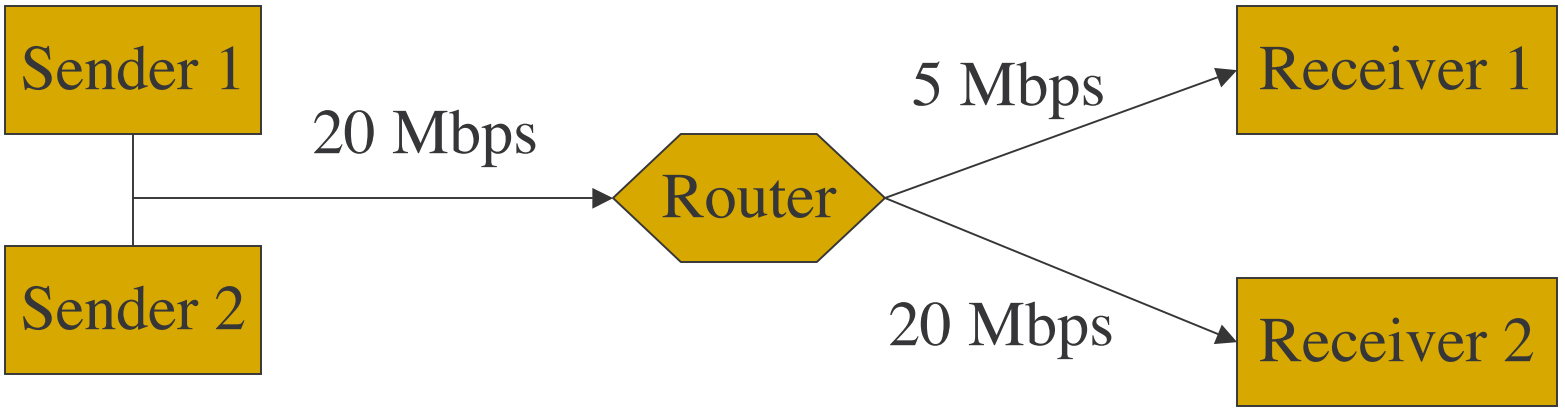
- What is congestion?
  - Higher rate of inputs to a router than outputs
- What are effects of congestion?
  - Delays
  - Loss
- What layer does congestion occur at?
  - Network layer
  - So why are we talking about it now?

Congestion, simple case

- Assume:
  - Sender transmits at full line rate (i.e. receiver window infinite)
  - Instant, free, precise loss notification
  - No other traffic on network

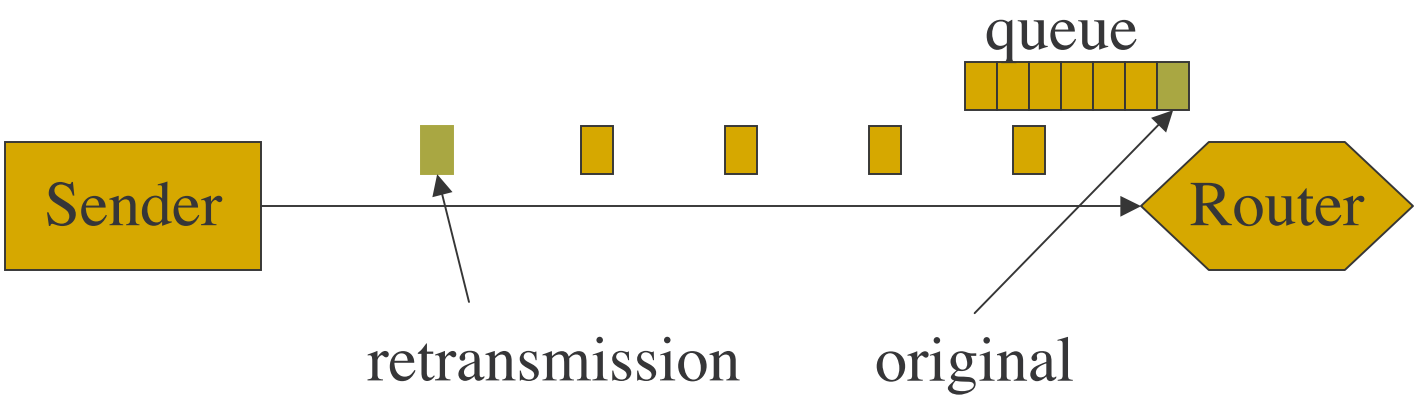
Congestion, two senders

- Assume:
  - Each sender transmits at 1/2 line rate



Congestion and Delays

- During congestion, delay is *much* greater than ordinary delay
- Since loss notification is imperfect, may retransmit packets still in the queue!



Congestion Problems

- Excessive queueing delays
- Wasted network capacity on retransmissions
  - Could have been used by other flows
  - Situation gets worse with multi-hop paths
- Wasteful retransmit of prematurely timed out packets
- How bad does it get?
  - 1000-fold reduction in bandwidth!

Congestion Control

- Congestion control involves two tasks:
  - Detect congestion
  - Limit sending rate
- Today we look at TCP approach to this

TCP Congestion Control

- Idea
  - Assumes best-effort network
    - FIFO or FQ
  - Each source determines network capacity for itself
  - Implicit feedback
  - ACKs pace transmission (self-clocking)
- Challenge
  - Determining initial available capacity
  - Adjusting to changes in capacity in a timely manner

TCP Congestion Control

- Basic idea
  - Add notion of congestion window
  - Effective window is smaller of
    - Advertised window (flow control)
    - Congestion window (congestion control)
  - Changes in congestion window size
    - Slow increases to absorb new bandwidth
    - Quick decreases to eliminate congestion

TCP Congestion Control

- Specific strategy
  - Self-clocking
    - Send data only when outstanding data ACK'd
    - Equivalent to send window limitation mentioned
  - Growth
    - Add one maximum segment size (MSS) per congestion window of data ACK'd
    - It's really done this way, at least in Linux:
      - see tcp_cong_avoid in tcp_input.c.
      - Actually, every ack for new data is treated as an MSS ACK'd
    - Known as additive increase

TCP Congestion Control

- Specific strategy (continued)
  - Decrease
    - Cut window in half when timeout occurs
    - In practice, set window = window /2
    - Known as multiplicative decrease
  - Additive increase, multiplicative decrease (AIMD)

Additive Increase/ Multiplicative Decrease

- Objective
  - Adjust to changes in available capacity
- Tools
  - React to observance of congestion
  - Probe channel to detect more resources
- Observation
  - On notice of congestion
    - Decreasing too slowly will not be reactive enough
  - On probe of network
    - Increasing too quickly will overshoot limits

Additive Increase/ Multiplicative Decrease

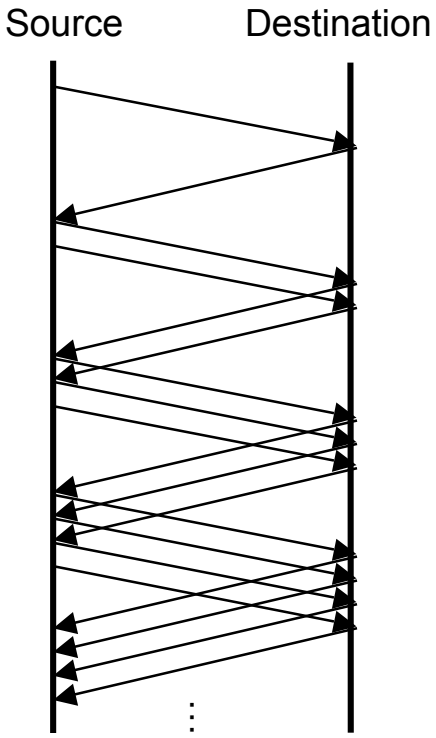
- New TCP state variable
  - **`CongestionWindow`**
    - Similar to **`AdvertisedWindow`** for flow control
  - Limits how much data source can have in transit
    - **`MaxWin = MIN(CongestionWindow, AdvertisedWindow)`**
    - **`EffWin = MaxWin - (LastByteSent - LastByteAcked)`**
    - TCP can send no faster then the slowest component, network or destination
- Idea
  - Increase **`CongestionWindow`** when congestion goes down
  - Decrease **`CongestionWindow`** when congestion goes up

Additive Increase/ Multiplicative Decrease

- Question
  - How does the source determine whether or not the network is congested?
- Answer
  - Timeout signals packet loss
  - Packet loss is rarely due to transmission error (on wired lines)
  - Lost packet implies congestion!

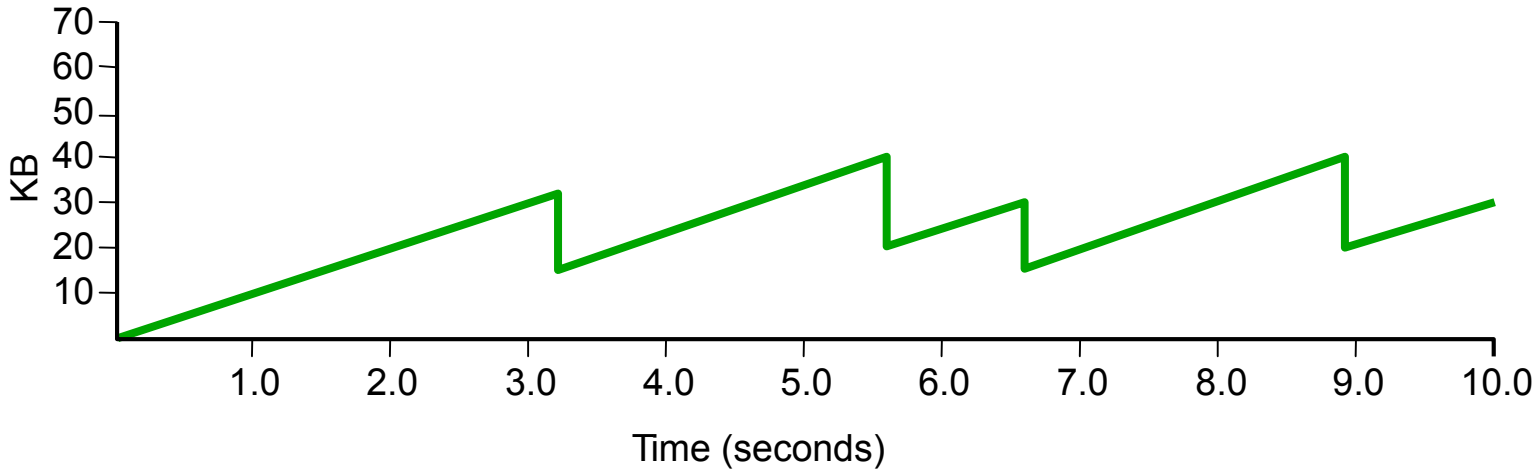
Additive Increase/ Multiplicative Decrease

- Algorithm
  - Increment CongestionWindow by one packet per RTT
    - Linear increase
  - Divide CongestionWindow by two whenever a timeout occurs
    - Multiplicative decrease



Additive Increase/ Multiplicative Decrease

- Sawtooth trace



TCP Start Up Behavior

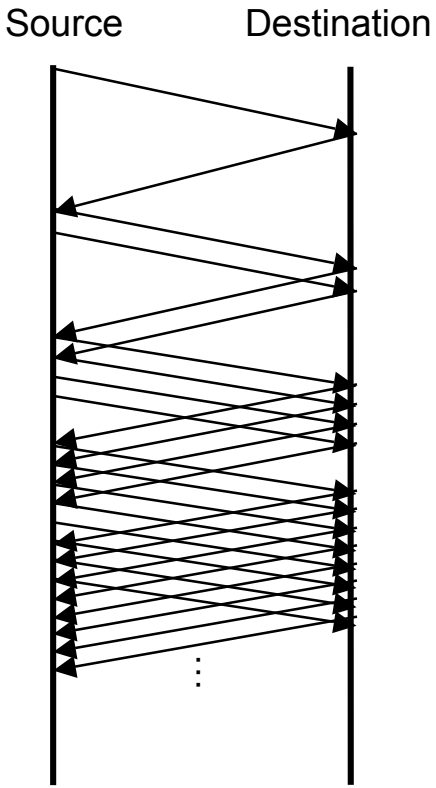
- How should TCP start sending data?
  - AIMD is good for channels operating at capacity
  - AIMD can take a long time to ramp up to full capacity from scratch
  - Use Slow Start to increase window rapidly from a cold start

TCP Start Up Behavior

- Initialization of the congestion window
  - Congestion window should start small
  - Avoid congestion due to new connections
  - Start at 1 MSS, reset to 1 MSS with each timeout (note that timeouts are coarse-grained, ~1/2 sec)
  - Known as slow start

Slow Start

- Objective
  - Determine initial available capacity
- Idea
  - Begin with **`CongestionWindow`** = 1 packet
  - Double **`CongestionWindow`** each RTT
    - Increment by 1 packet for each ACK
  - Continue increasing until loss
- Result
  - Exponential growth
  - Slower than all at once
- Used
  - When first starting connection
  - When connection times out

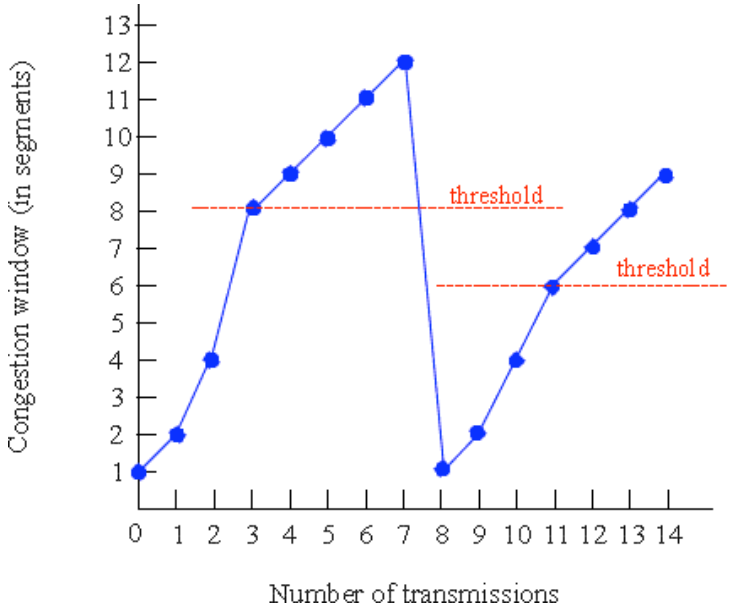


TCP Congestion Control

- To make up for slow start, ramp up congestion window quickly
- Maintain threshold window size
- Use multiplicative increase
  - When congestion window smaller than threshold
  - Double window for each window ACK'd
- Threshold value
  - Initially set to maximum window size
  - Set to 1/2 of current window on timeout
- In practice, increase congestion window by one MSS for each ACK of new data (or N bytes for N bytes)

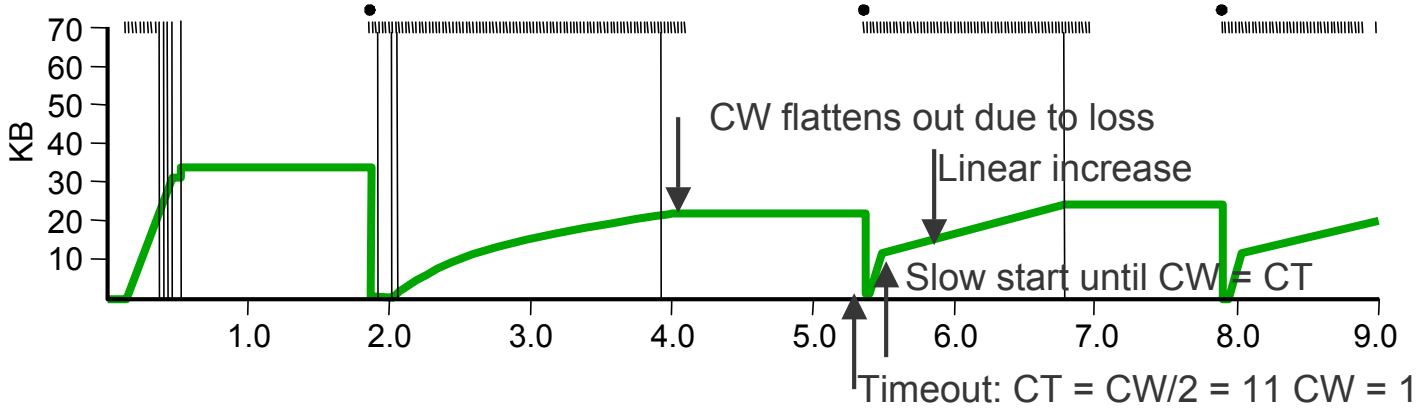
Slow Start

- How long should the exponential increase from slow start continue?
  - New variable: target window size **`CongestionThreshold`**
  - Estimate network capacity
  - When **`CongestionWindow`** reaches **`CongestionThreshold`** switch to additive increase
- Initial values
  - **`CongestionThreshold = 8`**
  - **`CongestionWindow = 1`**
- Loss after transmission 7
  - **`CongestionWindow`** currently 12
  - Set **`Congestionthreshold = CongestionWindow/2`**
  - Set **`CongestionWindow = 1`**



Slow Start

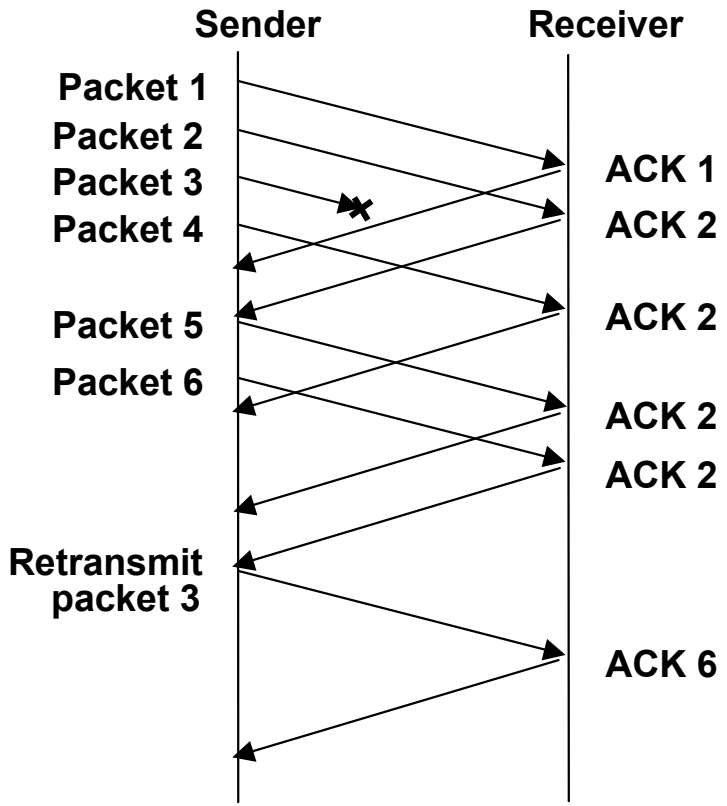
- Example trace of **`CongestionWindow`**



- Problem
  - Have to wait for timeout
  - Can lose half **`CongestionWindow`** of data

Fast Retransmit and Fast Recovery

- Problem
  - Coarse-grain TCP timeouts lead to idle periods
- Solution
  - Fast retransmit: use duplicate ACKs to trigger retransmission

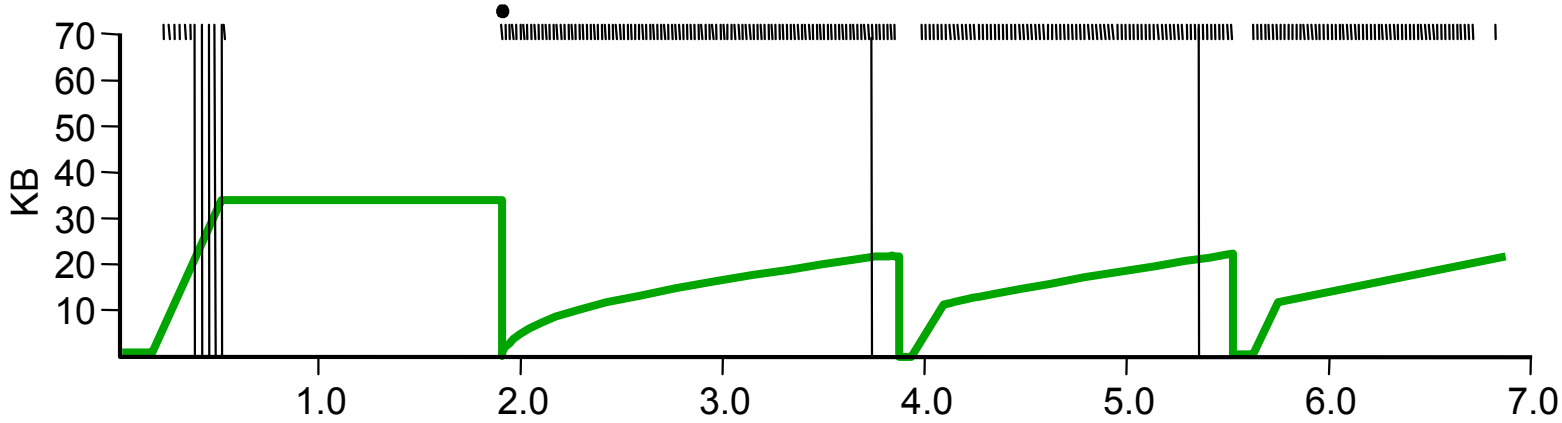


Fast Retransmit and Fast Recovery

- Send ACK for each segment received
- When duplicate ACK's received
  - Resend lost segment immediately
  - Do not wait for timeout
  - In practice, retransmit on 3rd duplicate
- Fast recovery
  - When fast retransmission occurs, skip slow start
  - Congestion window becomes 1/2 previous
  - Start additive increase immediately

Fast Retransmit and Fast Recovery

- Results



  - Fast Recovery
    - Bypass slow start phase
      - Increase immediately to one half last successful `CongestionWindow` (`ssthresh`)

TCP Congestion Window Trace