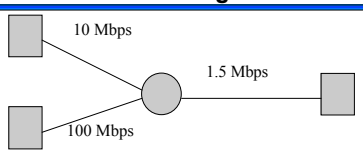
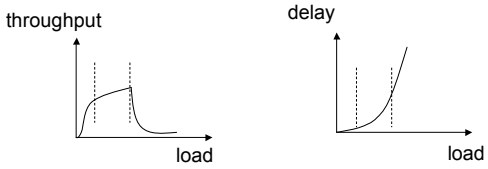
Congestion



- If both sources send full windows, we may get congestion collapse
- Other forms of congestion collapse:
  - Retransmissions of large packets after loss of a single fragment
  - Non-feedback controlled sources

Congestion Response



Avoidance keeps the system performing at the *knee*
Control kicks in once the system has reached a congested state

Separation of Functionality

- Sending host must adjust amount of data it puts in the network based on detected congestion
- Routers can help by:
  - Sending accurate congestion signals
  - Isolating well-behaved from ill-behaved sources

6.3 TCP Congestion Control

- Idea
  - assumes best-effort network (FIFO or FQ routers)each source determines network capacity for itself
  - uses implicit feedback
  - ACKs pace transmission (*self-clocking*)
- Challenge
  - determining the available capacity in the first place
  - adjusting to changes in the available capacity

TCP Congestion Control

- A collection of interrelated mechanisms:
  - Slow start
  - Congestion avoidance
  - Accurate retransmission timeout estimation
  - Fast retransmit
  - Fast recovery

Congestion Control

- Underlying design principle: packet conservation
  - At equilibrium, inject packet into network only when one is removed
  - Basis for stability of physical systems
- A mechanism which:
  - Uses network resources efficiently
  - Preserves fair network resource allocation
  - Prevents or avoids collapse
- Congestion collapse is not just a theory
  - Has been frequently observed in many networks

TCP Congestion Control Basics

- Keep a congestion window, cwnd
  - Denotes how much network is able to absorb
- Sender's maximum window:
  - Min (advertised window, cwnd)
- Sender's actual window:
  - Max window - unacknowledged segments

Congestion Under Infinite Buffering

- Nagle (RFC 970) showed that congestion will not go away even with infinite buffers
- Basic argument
  - A datagram network must have TTL
  - With infinite buffering queuing delays increase
  - Even if buffers are not dropped for lack of buffering, they will be dropped because TTL expires

Additive Increase/Multiplicative Decrease

- Objective: adjust to changes in the available capacity
- New state variable per connection: `CongestionWindow`
  - limits how much data source has in transit

```
MaxWin = MIN(CongestionWindow,
             AdvertisedWindow)
EffWin = MaxWin - (LastByteSent -
                   LastByteAcked)
```

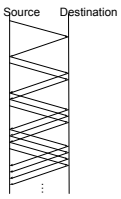
- Idea:
  - increase `CongestionWindow` when congestion goes down
  - decrease `CongestionWindow` when congestion goes up

AIMD (cont)

- Question: how does the source determine whether or not the network is congested?
- Answer: a timeout occurs
  - timeout signals that a packet was lost
  - packets are seldom lost due to transmission error
  - lost packet implies congestion

AIMD (cont)

- Algorithm
  - increment `CongestionWindow` by one packet per RTT (*linear increase*)
  - divide `CongestionWindow` by two whenever a timeout occurs (*multiplicative decrease*)

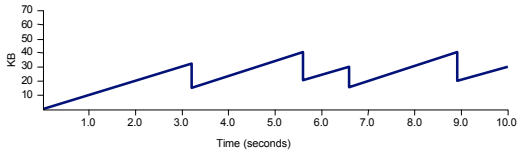


- In practice: increment a little for each ACK

```
Increment = (MSS * MSS)/CongestionWindow
CongestionWindow += Increment
```


AIMD (cont)

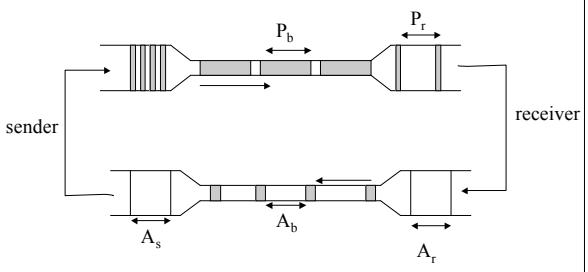
- Trace: sawtooth behavior



Self-clocking

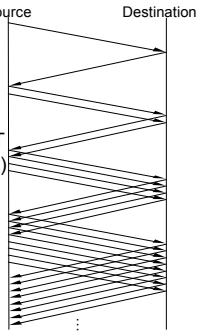
- If we have large actual window, should we send data in one shot?
  - No, use acks to clock sending new data

..Self-clocking

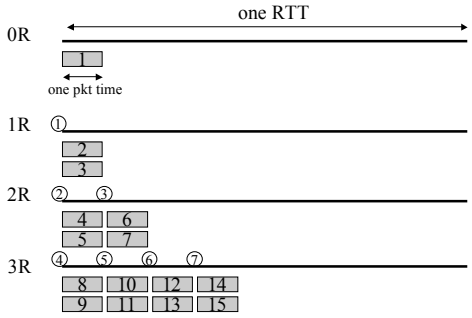


Slow Start

- Objective: determine the available capacity in the first
- Idea:
  - begin with CongestionWindow = 1 packet
  - double CongestionWindow each RTT (increment by 1 packet for each ACK)

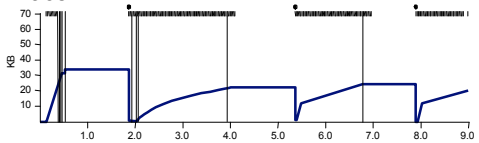


Slow Start Example



Slow Start (cont)

- Exponential growth, but slower than all at once
- Used…
  - when first starting connection
  - when connection goes dead waiting for timeout
- Trace



- Problem: lose up to half a CongestionWindow's worth of data

Congestion Avoidance

- Coarse grained timeout as loss indicator
- If loss occurs when cwnd = W
  - Network can absorb 0.5W ~ W segments
  - Set cwnd to 0.5W (multiplicative decrease)
  - Needed to avoid exponential queue buildup
- Upon receiving ACK
  - Increase cwnd by 1/cwnd (additive increase)
  - Multiplicative increase -> non-convergence

Slow Start and Congestion Avoidance

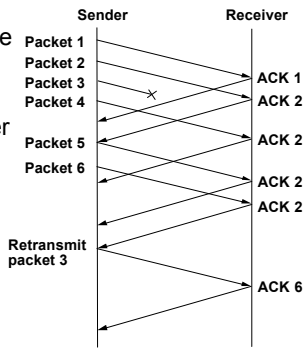
- If packet is lost we lose our self clocking as well
  - Need to implement slow-start and congestion avoidance together
- When timeout occurs set ssthresh to 0.5w
  - If cwnd < ssthresh, use slow start
  - Else use congestion avoidance

Impact of Timeouts

- Timeouts can cause sender to
  - Slow start
  - Retransmit a possibly large portion of the window
- Bad for lossy high bandwidth-delay paths
- Can leverage duplicate acks to:
  - Retransmit fewer segments (fast retransmit)
  - Advance cwnd more aggressively (fast recovery)

Fast Retransmit and Fast Recovery

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



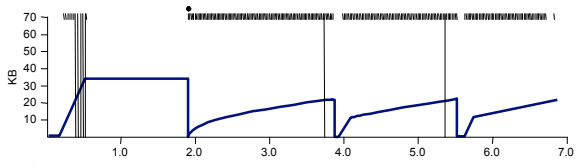
Fast Retransmit and Recovery

- If we get 3 duplicate acks for segment N
  - Retransmit segment N
  - Set ssthresh to 0.5*cwnd
  - Set cwnd to ssthresh + 3
- For every subsequent duplicate ack
  - Increase cwnd by 1 segment
- When new ack received
  - Reset cwnd to ssthresh (resume congestion avoidance)

Fast Recovery

- In congestion avoidance mode, if duplicate acks are received, reduce cwnd to half
- If n successive duplicate acks are received, we know that receiver got n segments after lost segment:
  - Advance cwnd by that number

Results



- Fast recovery
  - skip the slow start phase
  - go directly to half the last successful `CongestionWindow (ssthresh)`

TCP Extensions

- Implemented using TCP options
  - Timestamp
  - Protection from sequence number wraparound
  - Large windows

Timestamp Extension

- Used to improve timeout mechanism by more accurate measurement of RTT
- When sending a packet, insert current timestamp into option
- Receiver echoes timestamp in ACK

Protection Against Wrap Around

- 32-bit `SequenceNum`

Bandwidth	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

- Use timestamp to distinguish sequence number wraparound

Keeping the Pipe Full

- 16-bit `AdvertisedWindow`

Bandwidth	Delay x Bandwidth Product
T1 (1.5 Mbps)	18KB
Ethernet (10 Mbps)	122KB
T3 (45 Mbps)	549KB
FDDI (100 Mbps)	1.2MB
STS-3 (155 Mbps)	1.8MB
STS-12 (622 Mbps)	7.4MB
STS-24 (1.2 Gbps)	14.8MB

Large Windows

- Apply scaling factor to advertised window
  - Specifies how many bits window must be shifted to the left
- Scaling factor exchanged during connection setup

TCP Flavors

- Tahoe, Reno, Vegas
- TCP Tahoe (distributed with 4.3BSD Unix)
  - Original implementation of van Jacobson's mechanisms (VJ paper)
  - Includes:
    - Slow start (exponential increase of initial window)
    - Congestion avoidance (additive increase of window)
    - Fast retransmit (3 duplicate acks)

TCP Reno

- 1990: includes:
  - All mechanisms in Tahoe
  - Addition of fast-recovery (opening up window after fast retransmit)
  - Delayed acks (to avoid silly window syndrome)
  - Header prediction (to improve performance)