TCP congestion control:

- “Probing” for usable bandwidth:
  - Ideally: transmit as fast as possible (cwnd as large as possible) without loss
  - Increase cwnd until loss (congestion)
  - Loss: decrease cwnd, then begin probing (increasing) again
TCP congestion control: Additive increase, multiplicative decrease

- **Approach:** Increase transmission rate (window size), probing for usable bandwidth, until loss occurs
  - **Additive increase:** Increase \( cwnd \) by 1 MSS every RTT until loss detected
  - **Multiplicative decrease:** Cut \( cwnd \) in half after loss

Saw tooth behavior: probing for bandwidth
TCP congestion control: Details

- **Sender limits transmission:**
  \[ \text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd} \]

- **Roughly,**
  \[ \text{rate} = \frac{\text{cwnd}}{\text{RTT}} \text{ Bytes/sec} \]

- **Cwnd is dynamic, function of perceived network congestion**

**How does sender perceive congestion?**

- Loss event = timeout or 3 duplicate acks
- TCP sender reduces rate (cwnd) after loss event

**Three mechanisms:**
- AIMD
- Slow start
- Conservative after timeout events
TCP slow start

- How do we get the clocking behavior to start?
  - Initialize cwnd = 1 MSS (typically 1460 bytes)
  - Upon receipt of every ack, cwnd = cwnd + 1 MSS

- Implications
  - Window actually increases to $W$ in $\text{RTT} \times \log_2(W)$
  - Exponential increase up to first loss event
  - Can overshoot window and cause packet loss

- **Summary:** initial rate is slow but ramps up exponentially fast
Slow start example

0R

1

One pkt time

1R

①

2

3

2R

②  ③

4   6

5   7

3R

④  ⑤  ⑥  ⑦

8   10  12  14

9   11  13  15
Slow start example (cont.)

- One segment
- Two segments
- Four segments
Slow start sequence number plot

Sequence No

Time
Congestion window
Congestion avoidance

- When reaching threshold (ssthresh) go from slow start to congestion avoidance
- Upon receiving ACK
  - Increase cwnd by MSS/cwnd
  - Results in additive increase
- Loss implies congestion – why?
  - Not necessarily true on all link types
Congestion avoidance sequence plot
Return to slow start

- If packet is lost we lose our self-clocking as well
  - Need to implement slow-start and congestion avoidance together
Congestion window
Return to slow start (cont.)

- If packet is lost we lose our self clocking as well
  - Need to implement slow-start and congestion avoidance together

- When loss occurs set
  - ssthresh to 0.5 w
  - cwnd to 1 MSS (TCP-Tahoe)
Overall TCP behavior
TCP congestion control: Summary

- "Probing" for usable bandwidth:
  - Ideally: transmit as fast as possible (cwnd as large as possible) without loss
  - Increase cwnd until loss (congestion)
  - Loss: decrease cwnd, then begin probing (increasing) again

- Two "phases"
  - Slow start
  - Congestion avoidance

- Important variables:
  - Cwnd
  - Threshold: defines threshold between two slow start phase, congestion control phase
TCP slow start

**Slowstart algorithm**

- initialize: $cwnd = 1$ MSS
- for (each segment ACKed)
  - $cwnd += 1$ MSS
- until (loss event OR $cwnd >$ threshold)

- Exponential increase (per RTT) in window size (not so slow!)
- Loss event: timeout and/or or three duplicate ACKs
TCP congestion avoidance

Congestion avoidance

/* slowstart is over */
/* cwnd > threshold */
Until (loss event) {
    every w segments ACKed:
        cwnd += 1 MSS
}
threshold = cwnd /2

cwnd = 1 MSS
perform slow start

1: TCP Reno skips slowstart (fast recovery) after three duplicate ACKs
TCP flavors

- Tahoe, Reno, Vegas, SACK

- TCP Tahoe (distributed with 4.3BSD Unix)
  - Original implementation of Van Jacobson’s mechanisms
  - Includes:
    - Slow start
    - Congestion avoidance
    - Fast retransmit
Fast retransmit

- What are duplicate acks (dupacks)?
  - Repeated acks for the same sequence
- When can duplicate acks occur?
  - Loss
  - Packet re-ordering
    - Window update – advertisement of new flow control window
- Assume re-ordering is infrequent and not of large magnitude
  - Use receipt of 3 or more duplicate acks as indication of loss
  - Don’t wait for timeout to retransmit packet
Fast retransmit
TCP Reno (1990)

- All mechanisms in Tahoe
- Addition of fast-recovery
  - Opening up congestion window after fast retransmit
- Delayed acks
- Header prediction
  - Implementation designed to improve performance
  - Has common case code inlined
- With multiple losses, Reno typically timeouts because it does not see duplicate acknowledgements
Fast recovery

- Skip slow start. On 3 dup ack event:
  - sssthresh <- 0.5 cwnd
  - cwndn <- ssstresh
- Each duplicate ack notifies sender that single packet has cleared network
- When < cwnd packets are outstanding
  - Allow new packets out with each new duplicate acknowledgement
Fast recovery

Sent for each dupack after W/2 dupacks arrive
# TCP sender congestion control (Reno)

<table>
<thead>
<tr>
<th>State</th>
<th>Event</th>
<th>TCP Sender Action</th>
<th>Commentary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow Start (SS)</td>
<td>ACK receipt for previously unacked data</td>
<td>cwnd = cwnd + MSS, If (cwnd &gt; Threshold) set state to “Congestion Avoidance”</td>
<td>Resulting in a doubling of cwnd every RTT</td>
</tr>
<tr>
<td>Congestion Avoidance (CA)</td>
<td>ACK receipt for previously unacked data</td>
<td>cwnd = cwnd + MSS / (cwnd )</td>
<td>Additive increase, resulting in increase of cwnd by 1 MSS every RTT</td>
</tr>
<tr>
<td>SS or CA</td>
<td>Loss event detected by triple duplicate ACK</td>
<td>Threshold = cwnd /2, cwnd = Threshold, Set state to “Congestion Avoidance”</td>
<td>Fast recovery, implementing multiplicative decrease. cwnd will not drop below 1 MSS.</td>
</tr>
<tr>
<td>SS or CA</td>
<td>Timeout</td>
<td>Threshold = cwnd /2, cwnd = 1 MSS, Set state to “Slow Start”</td>
<td>Enter slow start</td>
</tr>
<tr>
<td>SS or CA</td>
<td>Duplicate ACK</td>
<td>Increment duplicate ACK count for segment being acked</td>
<td>cwnd and Threshold not changed</td>
</tr>
</tbody>
</table>
Multiple losses

Sequence No

Time

Now what?
Retransmission
Duplicate Acks
Tahoe
Reno

Now what? → timeout
NewReno

- The ack that arrives after retransmission (partial ack) should indicate that a second loss occurred.

- When does NewReno timeout?
  - When there are fewer than three dupacks for first loss
  - When partial ack is lost

- How fast does it recover losses?
  - One per RTT
NewReno

Now what? → partial ack recovery
SACK

- Basic problem is that cumulative acks only provide little information
  - Ack for just the packet received
    - What if acks are lost? → carry cumulative also
    - Not used
  - Bitmask of packets received
    - Selective acknowledgement (SACK)

- How to deal with reordering
Now what? – send retransmissions as soon as detected
Performance issues

- Timeout >> fast rexmit
  - Need 3 dupacks/sacks
  - Not great for small transfers
    - Don’t have 3 packets outstanding
  - What are real loss patterns like?

- Right edge recovery
  - Allow packets to be sent on arrival of first and second duplicate ack
  - Helps recovery for small windows

- How to deal with reordering?
TCP extensions

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

- Wraparound time vs. link speed
  - 1.5 Mbps: 6.4 hours
  - 10 Mbps: 57 minutes
  - 45 Mbps: 13 minutes
  - 100 Mbps: 6 minutes
  - 622 Mbps: 55 seconds → < MSL!
  - 1.2 Gbps: 28 seconds

- Use timestamp to distinguish sequence number wraparound
Large windows

- Delay-bandwidth product for 100 ms delay
  - 1.5 Mbps: 18 KB
  - 10 Mbps: 122 KB > max 16 bit window
  - 45 Mbps: 549 KB
  - 100 Mbps: 1.2 MB
  - 622 Mbps: 7.4 MB
  - 1.2 Gbps: 14.8 MB

- Scaling factor on advertised window
  - Specifies how many bits window must be shifted to the left
  - Scaling factor exchanged during connection setup
Maximum segment size (MSS)

- Exchanged at connection setup
  - Typically pick MTU of local link
- What all does this effect?
  - Efficiency
  - Congestion control
  - Retransmission
- Path MTU discovery
  - Why should MTU match MSS?
Effects of TCP latencies

Q: client latency for object request from WWW server to receipt?

- TCP connection establishment
- Data transfer delay

Notation, assumptions:

- Assume: fixed congestion window, $W$, giving throughput of $R$ bps
- $S$: MSS (bits)
- $O$: object/file size (bits)
- No retransmissions (no loss, no corruption)

Two cases to consider:

- Case 1: $WS/R > RTT + S/R$: ACK for first segment in window before window’s worth of data sent
- Case 2: $WS/R < RTT + S/R$: wait for ACK after sending window’s worth of data sent
Effects of TCP latencies (cont.)

Case 1: \( \text{latency} = 2\text{RTT} + O/R \)

Case 2: \( \text{latency} = 2\text{RTT} + O/R + (K-1)[S/R + \text{RTT} - WS/R] \)
\( K = O/(WS) \) (number of windows to transmit object)
Transport layer: Summary

- Principles behind transport layer services:
  - Multiplexing/demultiplexing
  - Reliable data transfer
  - Flow control
  - Congestion control
- Instantiation and implementation in the Internet
  - UDP
  - TCP

Next:
- Leaving the network “edge” (application transport layer)
- Into the network “core”