Slow Start (TCP Additive Increase)
TCP congestion control overview

Sender uses **congestion window (cwnd)**
- Sending rate (≈cwnd/RTT)

Sender uses loss as network congestion signal

Follow AIMD control law for a good allocation
- Goal is efficient and (roughly) fair allocation
TCP “Slow Start” Problem

We want to quickly get to the right cwnd but it varies

• Fixed window can be too inefficient or too aggressive

• Additive Increase adapts cwnd gently, but might take a long time to become efficient
Slow-Start Solution

Start by doubling cwnd every RTT
  • Exponential growth (1, 2, 4, 8, 16, …)
  • Start slow, quickly reach large values
Slow-Start Solution (2)

Eventually packet loss will occur when the network is congested

• Loss timeout tells us cwnd is too large
• Next time, switch to AI beforehand
• Slowly adapt cwnd near right value

In terms of cwnd:

• Expect loss for cwnd_C ≈ 2BD+queue
• Use ssthresh = cwnd_C/2 to switch to AI
Slow-Start Solution (3)

- Combined behavior, **after first time**
  - Most time spent near right value

![Graph showing slow-start and AI phases](image-url)
Slow-Start (Doubling) Timeline

Increment cwnd by 1 packet for each ACK
Additive Increase Timeline

Increment cwnd by 1 packet every cwnd ACKs (or 1 RTT)
TCP Tahoe (Implementation)

Initial slow-start (doubling) phase
• Start with $cwnd = 1$ (or small value)
• $cwnd += 1$ packet per ACK

Later Additive Increase phase
• $cwnd += 1/cwnd$ packets per ACK
• Roughly adds 1 packet per RTT

Switching threshold (initially infinity)
• Switch to AI when $cwnd > ssthresh$
• Set $ssthresh = cwnd/2$ after loss
• Begin with slow-start after timeout
Fast Recovery
(TCP Multiplicative Decrease)
Inferring Loss from ACKs

• TCP uses a cumulative ACK
  • Carries highest in-order seq. number
  • Normally a steady advance

• Duplicate ACKs give us hints about what data hasn’t arrived
  • Tell us some new data did arrive, but it was not next segment
  • Thus the next segment may be lost
Fast Retransmit

• Treat three duplicate ACKs as a loss
  • Retransmit next expected segment
  • Some repetition allows for reordering, but still detects loss quickly
Fast Retransmit (2)

Third duplicate ACK, so send 14

ACK jumps after loss is repaired

Data 14 was lost earlier, but got 15 to 20

Retransmission fills in the hole at 14

Data 14 was lost earlier, but got 15 to 20
Fast Retransmit (3)

• It can repair single segment loss quickly, typically before a timeout

• However, we have quiet time at the sender/receiver while waiting for the ACK to jump

• And we still need to MD cwnd ...
Inferring Non-Loss from ACKs

• Duplicate ACKs also give us hints about what data has arrived
  • Each new duplicate ACK means that some new segment has arrived
  • It will be the segments after the loss
  • Thus advancing the sliding window will not increase the number of segments stored in the network
Fast Recovery

• First fast retransmit, and MD cwnd
• Then pretend further duplicate ACKs are the expected ACKs
  • Lets new segments be sent for ACKs
  • Reconcile views when the ACK jumps
Third duplicate ACK, so send 14

Set ssthresh, cwnd = cwnd/2

More ACKs advance window; may send segments before jump

Data 14 was lost earlier, but got 15 to 20

Retransmission fills in the hole at 14

Fast Recovery (2)

Exit Fast Recovery
Fast Recovery (3)

• With fast retransmit, it repairs a single segment loss quickly and keeps the ACK clock running
• This allows us to realize AIMD
  • No timeouts or slow-start after loss, just continue with a smaller cwnd
• TCP Reno combines slow-start, fast retransmit and fast recovery
  • Multiplicative Decrease is $\frac{1}{2}$
TCP Reno

MD of $\frac{1}{2}$, no slow-start

TCP sawtooth

ACK clock running
TCP Reno, NewReno, and SACK

• Reno can repair one loss per RTT
  • Multiple losses cause a timeout
• NewReno further refines ACK heuristics
  • Repairs multiple losses without timeout
• Selective ACK (SACK) is a better idea
  • Receiver sends ACK ranges so sender can retransmit without guesswork
Network-Assisted Congestion Control
Congestion Avoidance vs. Control

• Classic TCP drives the network into congestion and then recovers
  • Needs to see loss to slow down
• Would be better to use the network but avoid congestion altogether!
  • Reduces loss and delay
• But how can we do this?
Feedback Signals

• Delay and router signals can let us avoid congestion

<table>
<thead>
<tr>
<th>Signal</th>
<th>Example Protocol</th>
<th>Pros / Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet loss</td>
<td>Classic TCP</td>
<td>Hard to get wrong</td>
</tr>
<tr>
<td></td>
<td>Cubic TCP (Linux)</td>
<td>Hear about congestion late</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other events can cause loss</td>
</tr>
<tr>
<td>Packet delay</td>
<td>Compound TCP</td>
<td>Hear about congestion early</td>
</tr>
<tr>
<td></td>
<td>(Windows)</td>
<td>Need to infer congestion</td>
</tr>
<tr>
<td>Router indication</td>
<td>TCPs with Explicit</td>
<td>Hear about congestion early</td>
</tr>
<tr>
<td></td>
<td>Congestion Notification</td>
<td>Require router support</td>
</tr>
</tbody>
</table>

ECN (Explicit Congestion Notification)

- Router detects the onset of congestion via its queue
  - When congested, it marks affected packets (IP header)
ECN (2)

• Marked packets arrive at receiver
• TCP receiver informs TCP sender of the congestion
ECN (3)

• Advantages:
  • Routers deliver clear signal to hosts
  • Congestion is detected early, no loss
  • No extra packets need to be sent

• Disadvantages:
  • Routers and hosts must be upgraded
What’s new in transport protocols?

QUIC
MPTCP
BBR
DCTCP
QUIC

Zero RTT Connection Establishment

TCP

100 ms

TCP + TLS

200 ms¹
300 ms²

QUIC
(equivalent to TCP + TLS)

0 ms¹
100 ms²

1. Repeat connection
2. Never talked to server before

https://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html
MPTCP: Multipath TCP

Data transmission with plain TCP

Alice

3G (10.11.12.13)

Plain TCP connexion

Eth (20.21.22.23)

One link is available but unused

WiFi (10.11.12.14)

data

Bob

Data transmission with MPTCP

Alice

3G (10.11.12.13)

Plain TCP connexion (subflow 1)

WiFi (10.11.12.14)

Plain TCP connexion (subflow 2)

data

Eth (20.21.22.23)

All the available links are used

By Aclarembeau - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=49727919
BBR: Bottleneck Bandwidth and Round trip propagation

https://queue.acm.org/detail.cfm?id=3022184
DCTCP at a glance

- **Reaction Point (RP)**
  - Sender
  - Estimate fraction of marked packets
  - Adjust cwnd accordingly

- **Congestion Point (CP)**
  - Router
  - If qlen >= K, mark ECN

- **Notification Point (NP)**
  - Receiver
  - Echo ECN to RP

If $qlen \geq K$, mark ECN

Estimate fraction of marked packets

Adjust cwnd accordingly
Recap: Transport protocols

Goal: Provide end-to-end message delivery to applications
  • Reliable (or not), messages vs streams

Challenges:
  • Dealing with packet losses
  • Dealing with slow receivers (flow control) and network (congestion control)
  • Adapting to network conditions
    • Determine the right sending rate for yourself
    • Individual behaviors resulting in efficient and fair resource use

Toolbox
  • Timeouts/retransmissions, sliding windows, max-min fairness, AIMD, ....