CS 557
Congestion Avoidance

_Congestion Avoidance and Control_
Jacobson and Karels, 1988

Spring 2013
The Story So Far…

Network layer: Addressing, Fragmentation, Dynamic Routing, Best Effort Forwarding

Transport layer: End to End communication, Multiplexing, Reliability, Congestion control, Flow control,

Data Layer: richly connected network (many paths) with many types of unreliable links

Some Essential Apps: DNS (naming) and NTP (time).
Main Points

• Objective:
  – Techniques for dealing with network congestion

• Approach:
  – Slow Start
  – Adaptive Timers.
  – Additive Increase/multiplicative decrease

• Contributions:
  – Essential points of TCP congestion control
Motivation and Context

• Network Collapsing Due to Congestion
  – Throughput drops from 32 Kbps to 40 bps
  – One conclusion, packet switching failed…
  – This paper says we can fix the problem

• Conservation of Packets
  – Can’t have collapse if packets entering network = packets leaving network
  – Can we achieve conservation of packets?
TCP Review
RFCs: 793, 1122, 1323, 2018, 2581

- **point-to-point:**
  - one sender, one receiver
- **reliable, in-order byte steam:**
  - no “message boundaries”
- **pipelined:**
  - *TCP congestion and flow control set window size*
- **send & receive buffers**

- **full duplex data:**
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- **connection-oriented:**
  - handshaking (exchange of control msgs) init’ s sender, receiver state before data exchange

- **flow controlled:**
  - sender will not overwhelm receiver
TCP Flow Control (1/2)

- receive side of TCP connection has a receive buffer:
  - speed-matching service: matching the send rate to the receiving app’s drain rate
  - sender won’t overflow receiver’s buffer by transmitting too much, too fast

- app process may be slow at reading from buffer
TCP Flow control (2/2)

(Suppose TCP receiver discards out-of-order segments)

- spare room in buffer
  \[= \text{RcvWindow} \]
  \[= \text{RcvBuffer} - \text{LastByteRcvd} - \text{LastByteRead} \]

- Rcvr advertises spare room by including value of RcvWindow in segments
- Sender limits unACKed data to RcvWindow
  - guarantees receive buffer doesn’t overflow
TCP seq. #’s and ACKs

Seq. #’s:
- byte stream “number” of first byte in segment’s data

ACKs:
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments
  - TCP spec doesn’t say, - up to implementation

User types ‘C’

Host A

Seq=42, ACK=79, data = ‘C’

host ACKs receipt of ‘C’, echoes back ‘C’

Seq=79, ACK=43, data = ‘C’

host ACKs receipt of echoed ‘C’

Seq=43, ACK=80

User

types

‘C’

Host B

simple telnet scenario
Challenges to Conservation

• Connection never reaches equilibrium
  – Too many initial packets drives network into congestion and then hard to recover….

• Sender adds packets before one leaves
  – Poor timer causes retransmission of packets that are still in-flight on the network.

• Equilibrium can’t be reached due to resource limits on path
  – Assume packet loss is due to congestion and back off by multiplicative factor
Slow Start

• TCP is Self-Clocking
  – Receipt of ack triggers new packet

• Good if Network is in Stable State:
  – How to ramp up at the start?
  – Start slow - 1 packet
  – Each ack triggers two packets

• Quickly Ramp Up Window to Correct Size
TCP: retransmission scenarios

- Host A: Seq=100, 20 bytes data
- Host B: Seq=92, 8 bytes data

Lost ACK scenario:
- Host A: Seq=92, 8 bytes data, ACK=100
- Host B: Seq=92, 8 bytes data, timeout
- Host A: SendBase = 100

Premature timeout scenario:
- Host A: Seq=100, 20 bytes data, ACK=100
- Host B: Seq=92, 8 bytes data, ACK=120
- Host A: SendBase = 120

SendBase = 100
TCP retransmission scenarios

Host A:
- Seq=92, 8 bytes data
- ACK=100
- loss

Host B:
- Seq=100, 20 bytes data
- ACK=100

Cumulative ACK scenario:
- SendBase = 120
- time

SendBase = 120

timeout

ACK=120

Cumulative ACK scenario
TCP Timeout Values

**Q:** how to set TCP timeout value?

- longer than RTT
  - but RTT varies
- too short: premature timeout
  - unnecessary retransmissions
- too long: slow reaction to segment loss

**Q:** how to estimate RTT?

- **SampleRTT**: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **SampleRTT** will vary, want estimated RTT “smoother”
  - average several recent measurements, not just current **SampleRTT**
TCP Round Trip Time (RTT)

EstimatedRTT = (1 - \( \alpha \)) \times \text{EstimatedRTT} + \alpha \times \text{SampleRTT}

- Exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: \( \alpha = 0.125 \)
Example RTT estimation:

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr

---

![Diagram showing RTT measurements](image-url)

- **RTT**: The round-trip time between gaia.cs.umass.edu and fantasia.eurecom.fr.
- **SampleRTT**: The actual measured RTT values over time.
- **Estimated RTT**: The estimated RTT values based on the measured data.
TCP Round Trip Time and Timeout

Setting the timeout

- **EstimatedRTT** plus “safety margin”
  - large variation in **EstimatedRTT** -> larger safety margin
- first estimate of how much SampleRTT deviates from EstimatedRTT:

\[
\text{DevRTT} = (1-\beta) \cdot \text{DevRTT} + \beta \cdot |\text{SampleRTT} - \text{EstimatedRTT}|
\]

(typically, \(\beta = 0.25\))

Then set timeout interval:

\[
\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \cdot \text{DevRTT}
\]
TCP Window Size Over Time

Long-lived TCP connection
TCP Congestion Control Review

• When $\text{CongWin}$ is below $\text{Threshold}$, sender is in slow-start phase, window grows exponentially.

• When $\text{CongWin}$ is above $\text{Threshold}$, sender is in congestion-avoidance phase, window grows linearly.

• When a triple duplicate ACK occurs, $\text{Threshold}$ set to $\text{CongWin}/2$ and $\text{CongWin}$ set to $\text{Threshold}$.

• When timeout occurs, $\text{Threshold}$ set to $\text{CongWin}/2$ and $\text{CongWin}$ is set to 1 MSS.
# TCP ACK generation

**[RFC 1122, RFC 2581]**

<table>
<thead>
<tr>
<th>Event at Receiver</th>
<th>TCP Receiver action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed</td>
<td>Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK</td>
</tr>
<tr>
<td>Arrival of in-order segment with expected seq #. One other segment has ACK pending</td>
<td>Immediately send single cumulative ACK, ACKing both in-order segments</td>
</tr>
<tr>
<td>Arrival of out-of-order segment higher-than-expect seq. # . Gap detected</td>
<td>Immediately send duplicate ACK, indicating seq. # of next expected byte</td>
</tr>
<tr>
<td>Arrival of segment that partially or completely fills gap</td>
<td>Immediate send ACK, provided that segment starts at lower end of gap</td>
</tr>
<tr>
<td>Event</td>
<td>State</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>ACK receipt for previously unacked data</td>
<td>Slow Start (SS)</td>
</tr>
<tr>
<td>ACK receipt for previously unacked data</td>
<td>Congestion Avoidance (CA)</td>
</tr>
<tr>
<td>Loss event detected by triple duplicate ACK</td>
<td>SS or CA</td>
</tr>
<tr>
<td>Timeout</td>
<td>SS or CA</td>
</tr>
<tr>
<td>Duplicate ACK</td>
<td>SS or CA</td>
</tr>
</tbody>
</table>
Fast Retransmit

- Time-out period often relatively long:
  - long delay before resending lost packet
- Detect lost segments via duplicate ACKs.
  - Sender often sends many segments back-to-back
  - If segment is lost, there will likely be many duplicate ACKs.

- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - **fast retransmit**: resend segment before timer expires
Fast retransmit algorithm:

event: ACK received, with ACK field value of y
  if (y > SendBase) {
    SendBase = y
    if (there are currently not-yet-acknowledged segments)
      start timer
  }
  else {
    increment count of dup ACKs received for y
    if (count of dup ACKs received for y = 3) {
      resend segment with sequence number y
    }
  }
TCP Fairness

Fairness goal: if $K$ TCP sessions share same bottleneck link of bandwidth $R$, each should have average rate of $R/K$. 

Diagram:
- TCP connection 1
- TCP connection 2
- Bottleneck router capacity $R$
Why is TCP fair?

Two competing sessions:
- Additive increase gives slope of 1, as throughput increases
- Multiplicative decrease drops throughput proportionally

Diagram showing equal bandwidth share with additive increase and loss, decrease window by factor of 2, congestion avoidance: additive increase.
TCP Throughput

- What’s the average throughput of TCP as a function of window size and RTT?
  - Ignore slow start
- Let W be the window size when loss occurs.
- When window is W, throughput is W/RTT
- Just after loss, window drops to W/2, throughput to W/2RTT.
- Average throughout: .75 W/RTT
Window Size and Loss Rate...

- 10 Gbps throughput required window size $W = 83,333$ in-flight segments
- TCP assumes every loss is due to congestion
  - Generally safe assumption for reasonable window size.
- Throughput in terms of loss rate:
  $$\frac{1.22 \cdot MSS}{RTT \sqrt{L}}$$
- $L = 2 \cdot 10^{-10}$ Wow
- New versions of TCP for high-speed needed!
Fairness (more)

**Fairness and UDP**
- Multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- Instead use UDP:
  - pump audio/video at constant rate, tolerate packet loss
- Research area: TCP friendly unreliable transport

**Fairness and parallel TCP connections**
- nothing prevents app from opening parallel connections between 2 hosts.
- Web browsers do this
- Example: link of rate $R$ supporting 9 connections;
  - new app asks for 1 TCP, gets rate $R/10$
  - new app asks for 11 TCPs, gets $R/2$!
TCP Delay Modeling

**Q:** How long does it take to receive an object from a Web server after sending a request?

Ignoring congestion, delay is influenced by:

- TCP connection establishment
- data transmission delay
- slow start

**Window size:**
- First assume: fixed congestion window
- Then dynamic window, modeling slow start
Calculating TCP Delay

**Simple case:**
Client requests a web page for server
Assume no congestion and *fixed window size*
Define the following
- \( O \) = object size (bits)
- \( W \) = window Size
- \( S \) = MSS (Max Segment Size)
  - Assume always send segment of size \( S \)
- \( R \) = Bandwidth
- \( RTT \) = Round Trip Time
ACKs and HTTP Request is very very small.

What is the best case scenario??
Delay Modeling

**Q:** How long does it take to receive an object from a Web server after sending a request?

Ignoring congestion, delay is influenced by:
- TCP connection establishment
- data transmission delay
- slow start

**Window size:**
- First assume: *fixed congestion window*
- Then dynamic window, modeling slow start
Calculating TCP Delay

Simple case:
Client requests a web page for server
Assume no congestion and fixed window size
Define the following
  O = object size (bits)
  W = window Size
  S = MSS (Max Segment Size)
    Assume always send segment of size S
  R = Bandwidth
  \textbf{RTT} = Round Trip Time = 2 \times \text{prop delay}
ACKs and HTTP Request is very very small.

What is the best case scenario??
Fixed congestion window (1)

Best case:
WS/R > RTT + S/R:
  ACK for first segment in window returns before window’s worth of data sent

\[ \text{delay} = 2RTT + O/R \]
Fixed congestion window (2)

Empty Pipe case:
- WS/R < RTT + S/R: wait for ACK after sending window’s worth of data sent
- K = number of windows before done

\[
\text{delay} = 2\text{RTT} + \frac{O}{R} + (K-1)[S/R + \text{RTT} - WS/R]
\]
TCP Delay Modeling: Slow Start (1)

Now suppose window grows according to slow start

The delay is:

\[
Delay = 2RTT + \frac{O}{R} + P \left[ RTT + \frac{S}{R} \right] - (2^p - 1) \frac{S}{R}
\]

where \( P \) is the number of times TCP idles at server:

\[
P = \min\{Q, K - 1\}
\]

- where \( Q \) is the number of times the server idles if the object were of infinite size.

- and \( K \) is the number of windows that cover the object.
TCP Delay Modeling: Slow Start (2)

**Delay components:**
- 2 RTT for connection estab and request
- O/R to transmit object
- time server idles due to slow start

**Server idles:**
\[ P = \min\{K-1,Q\} \] times

**Example:**
- \( O/S = 15 \) segments
- \( K = 4 \) windows
- \( Q = 2 \)
- \( P = \min\{K-1,Q\} = 2 \)

**Server idles \( P=2 \) times**
TCP Delay Modeling (3)

\[ \frac{S}{R} + RTT = \text{send data to receive ack} \]

\[ 2^{k-1} \frac{S}{R} = \text{transmit kth window} \]

\[ \left[ \frac{S}{R} + RTT - 2^{k-1} \frac{S}{R} \right]^+ = \text{kth window idle} \]

\[ \text{delay} = \frac{O}{R} + 2RTT + \sum_{p=1}^{P} \text{idleTime}_p \]

\[ = \frac{O}{R} + 2RTT + \sum_{k=1}^{P} \left[ \frac{S}{R} + RTT - 2^{k-1} \frac{S}{R} \right] \]

\[ = \frac{O}{R} + 2RTT + P \left[ RTT + \frac{S}{R} \right] - (2^P - 1) \frac{S}{R} \]
TCP Delay Modeling (4)

Recall $K =$ number of windows that cover object

How do we calculate $K$?

$$K = \min\{ k : 2^0 S + 2^1 S + L + 2^{k-1} S \geq O \}$$
$$= \min\{ k : 2^0 + 2^1 + L + 2^{k-1} \geq O/S \}$$
$$= \min\{ k : 2^k - 1 \geq \frac{O}{S} \}$$
$$= \min\{ k : k \geq \log_2 (\frac{O}{S} + 1) \}$$
$$= \left\lceil \log_2 (\frac{O}{S} + 1) \right\rceil$$

Calculation of $Q$, number of idles for infinite-size object, is similar (see HW).
TCP Evolution Continues…

• Consider the impact of high speed links:
  – 1500 byte segments,
  – 100ms RTT
  – 10 Gbps throughput

• What is the required window size?
  – Throughput = .75 W/RTT
  • (probably a good formula to remember)
  – Requires window size $W = 83,333$ in-flight segments
Summary

• Essential Components of TCP
  – Slow Start
  – AI/MD
  – Timer Estimates