SPDY, TCP, and the Single Connection Throttle

Mike Belshe
mbelshe@google.com
04/01/11
A New Protocol? What for?

Speed.
An average Web Page Consists of:
- ~44 resources
- ~7 hosts
- ~320KB
- ~66% compressed (top sites are ~90% compressed)
- Note: HTTPS is < 50% compressed.

Incremental improvements to HTTP don't move the needle
- Transparent proxies change the content.
- Example: pipelining
- Example: stripped "Accept-Encoding" headers
  - we can't even improve "negotiated" compression!
Quick SPDY Background

● Goals:
  ○ Faster web page downloads
  ○ Always secure
  ○ Deployable
  ○ Open

● Features (No rocket science here!)
  ○ Single-connection, Multiplexed, prioritized streams
  ○ Mandatory header compression
  ○ Supports server-push

● SPDY is Basic Networking "blocking and tackling"
  ○ Use fewer connections
  ○ Send fewer bytes
HTTP Connection Use Today

Average: 29 connections per page.

25%-tile = 10  
50%-tile = 20  
75%-tile = 39  
95%-tile = 78
Reducing Upload Bytes

SPDY vs HTTP Upload KB Sent (Top-45 pages)

HTTP

SPDY

51% reduction
Reducing Download Bytes

SPDY vs HTTP Download KB (Top-45 pages)

HTTP

SPDY

4% reduction
Reducing Total Packets

SPDY vs HTTP Total Packets (Top-45 pages)

HTTP

SPDY

19% reduction

# packets
Increasing Parallelism

SPDY vs HTTP Time to First Byte for a Request

<table>
<thead>
<tr>
<th>percentile</th>
<th>HTTP (ms)</th>
<th>SPDY (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>150</td>
<td>100</td>
</tr>
<tr>
<td>20</td>
<td>200</td>
<td>150</td>
</tr>
<tr>
<td>30</td>
<td>250</td>
<td>200</td>
</tr>
<tr>
<td>40</td>
<td>300</td>
<td>250</td>
</tr>
<tr>
<td>50</td>
<td>350</td>
<td>300</td>
</tr>
<tr>
<td>60</td>
<td>400</td>
<td>350</td>
</tr>
<tr>
<td>70</td>
<td>450</td>
<td>400</td>
</tr>
<tr>
<td>80</td>
<td>500</td>
<td>450</td>
</tr>
<tr>
<td>90</td>
<td>550</td>
<td>500</td>
</tr>
<tr>
<td>99.5</td>
<td>600</td>
<td>550</td>
</tr>
</tbody>
</table>
The Single Connection Throttle
Throttle #1: CWND

Problem:
- Server-side slow start limits server to N packets. (in flux)

Workaround:
- Use more client connections.
- Update server to go beyond spec.
- SPDY can use a cookie based cwnd.

Note:
- HTTP's per-domain cwnd is currently ~24 (6*4).
- draft-ietf-tcpm-initcwnd-00.txt helps
Throttle #1 CWND vs # connections
Throttle #2: Receive Windows

Problem:
- Some clients set initial rwnd to 5840 bytes (4 pkts)
- Trumps larger cwnd on servers.
- Patch just shipped this month in linux mainline

Workaround:
- Use more client connections.
Throttle #2: Init rwnd

Effect of init rwnd=32KB
Throttle #3: Intermediaries

Problem:
- "Just a bug"... but... Intermediaries can (and do) tamper.
- window scale enables large receive windows.

Workaround:
- Use more client connections.

Client Side
// Client wants window
// scaling 6.

SYN -> w=5840, ws=6
// Client receives server
// ws as sent.
SYNACK <- w=5840, ws=6

// going to be slow....

Server Side
// Server recvs window
// scale 3. Someone
// tampered with this.

SYN -> w=5840, ws=3
// Server sends its own
// ws of 6.
SYNACK <- w=5840, ws=6
Problem:

- Congestion detection decreases the send rate.
- But congestion signals can be erroneous.
- Applied to the connection, not the path:
  - 1 connection: single packet loss cuts send rate by N (typically 0.5/0.7).
  - 6 connections: single packet loss cuts send rate by $1/6*(1/N) = \approx (1/9\text{th to } 1/12\text{th})$

Workaround:

- Use more client connections.
Too Obsessed With 1 Connection?

- Could we use 2? 3?
  - Sure, but it neutralizes many of our benefits.
- Disadvantages of multiple connections:
  - Sharing state across connections is hard.
  - Server farms would be required to do sticky load balancing
  - Compression worsens (we use stateful compression)
  - Prioritization becomes impossible
  - Server push difficult
- But it shouldn't be this hard...
How Much Does A Handshake Cost?

The Cost of a Handshake: adding a 100ms delay

<table>
<thead>
<tr>
<th>Protocol</th>
<th>PLT (ms)</th>
<th>Normal</th>
<th>100ms delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td></td>
<td>1,800</td>
<td>2,400</td>
</tr>
<tr>
<td>SPDY</td>
<td></td>
<td>1,200</td>
<td>1,800</td>
</tr>
</tbody>
</table>
What's Next?

- Before SPDY, we could blame the app layer (HTTP).
- With SPDY, we're on the verge of proving that the transport is the new bottleneck.
- TCP needs to address 2 performance obstacles:
  - Data in initial handshake.
  - Single connection taxes.
- TCP needs to address security
  - Both Server Auth & Encryption
    - (Sorry I didn't have time to discuss in this talk!)
- How can we iterate on the transport when it is buried in the kernel? Can we auto-update the network stack?