SPDY, TCP, and the Single Connection Throttle

Mike Belshe
mbelshe@google.com
04/01/11

A New Protocol? What for?

Speed.

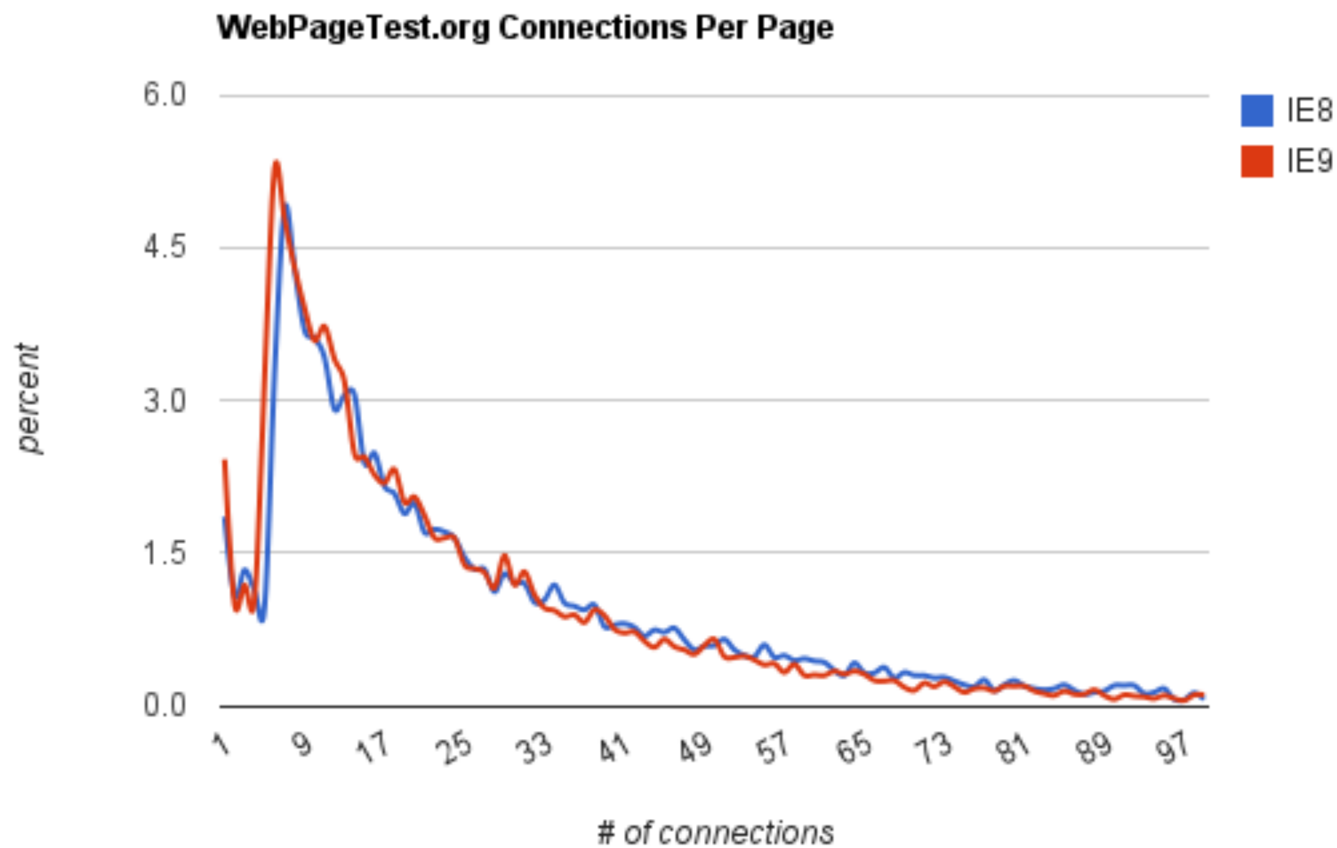
State of the Web Page

- 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

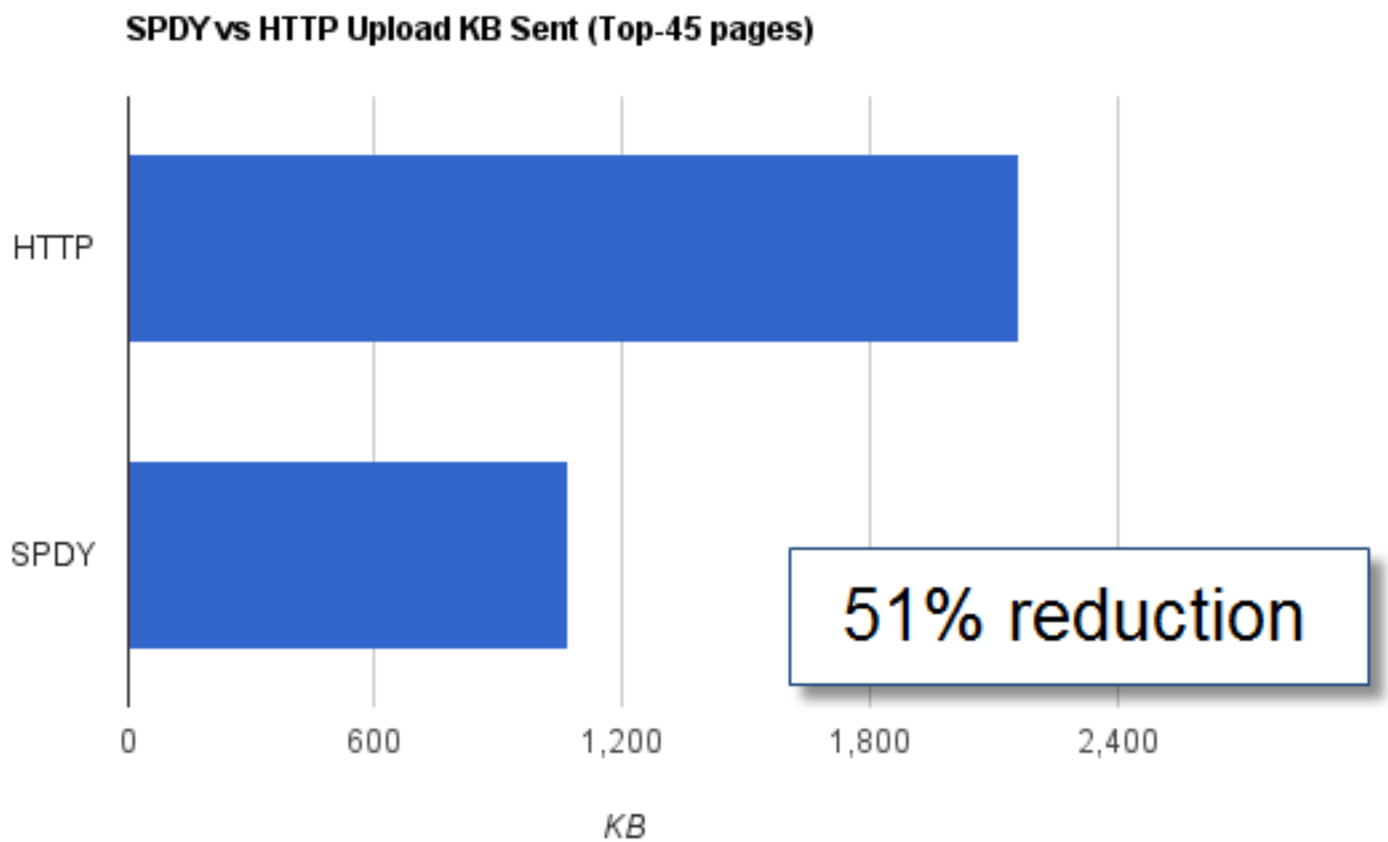
HTTP Connection Use Today



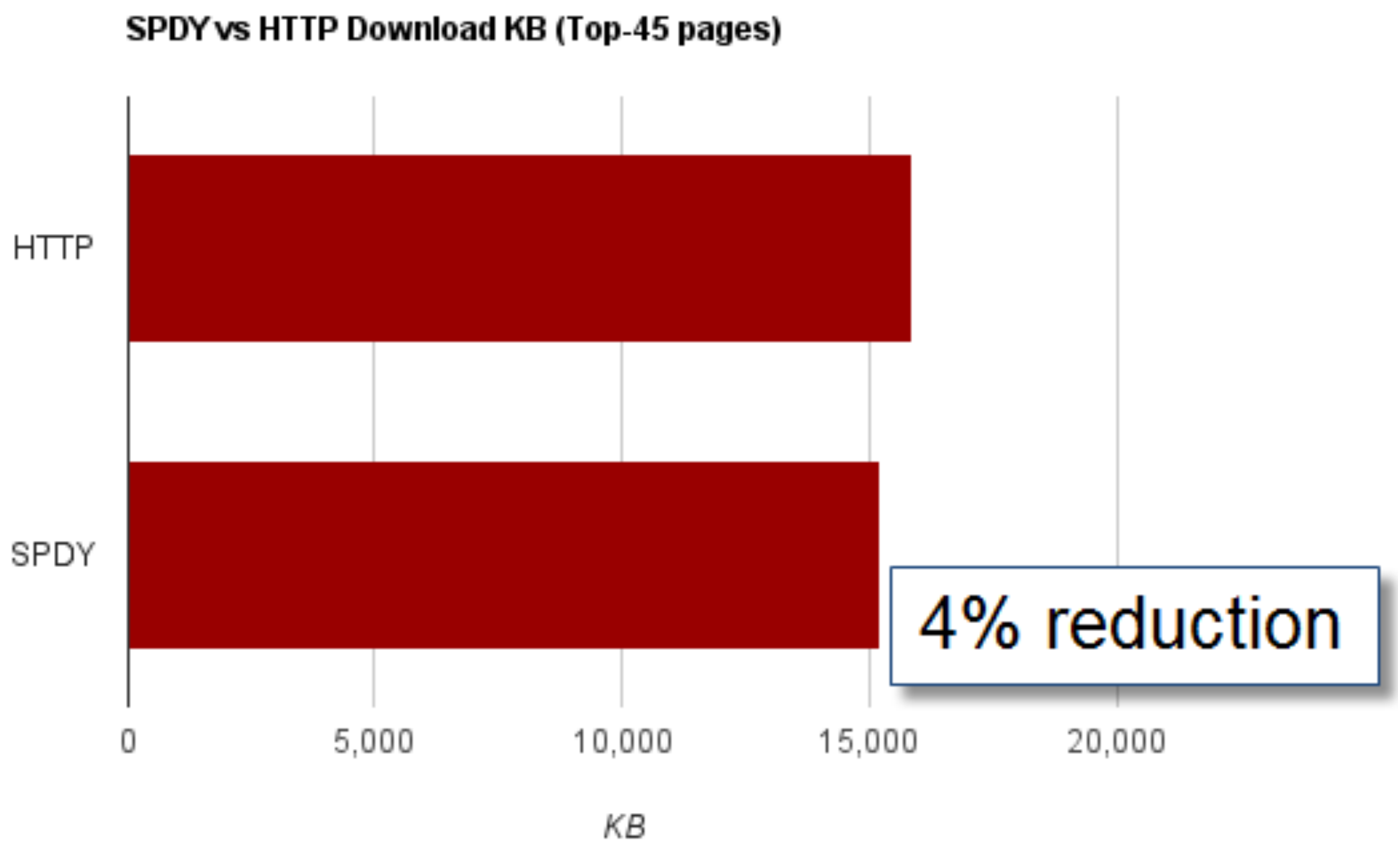
Average: 29 connections per page.

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

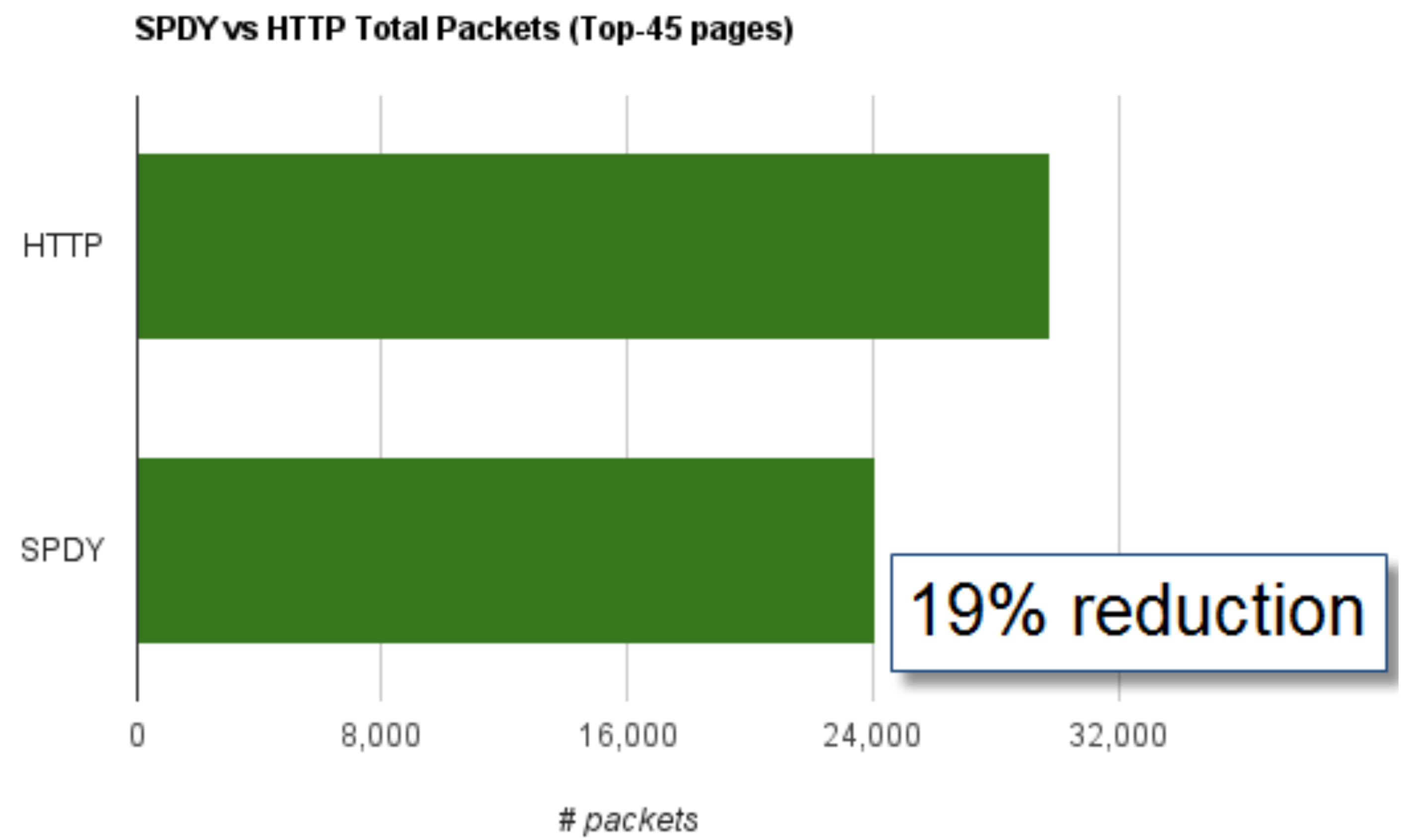
Reducing Upload Bytes



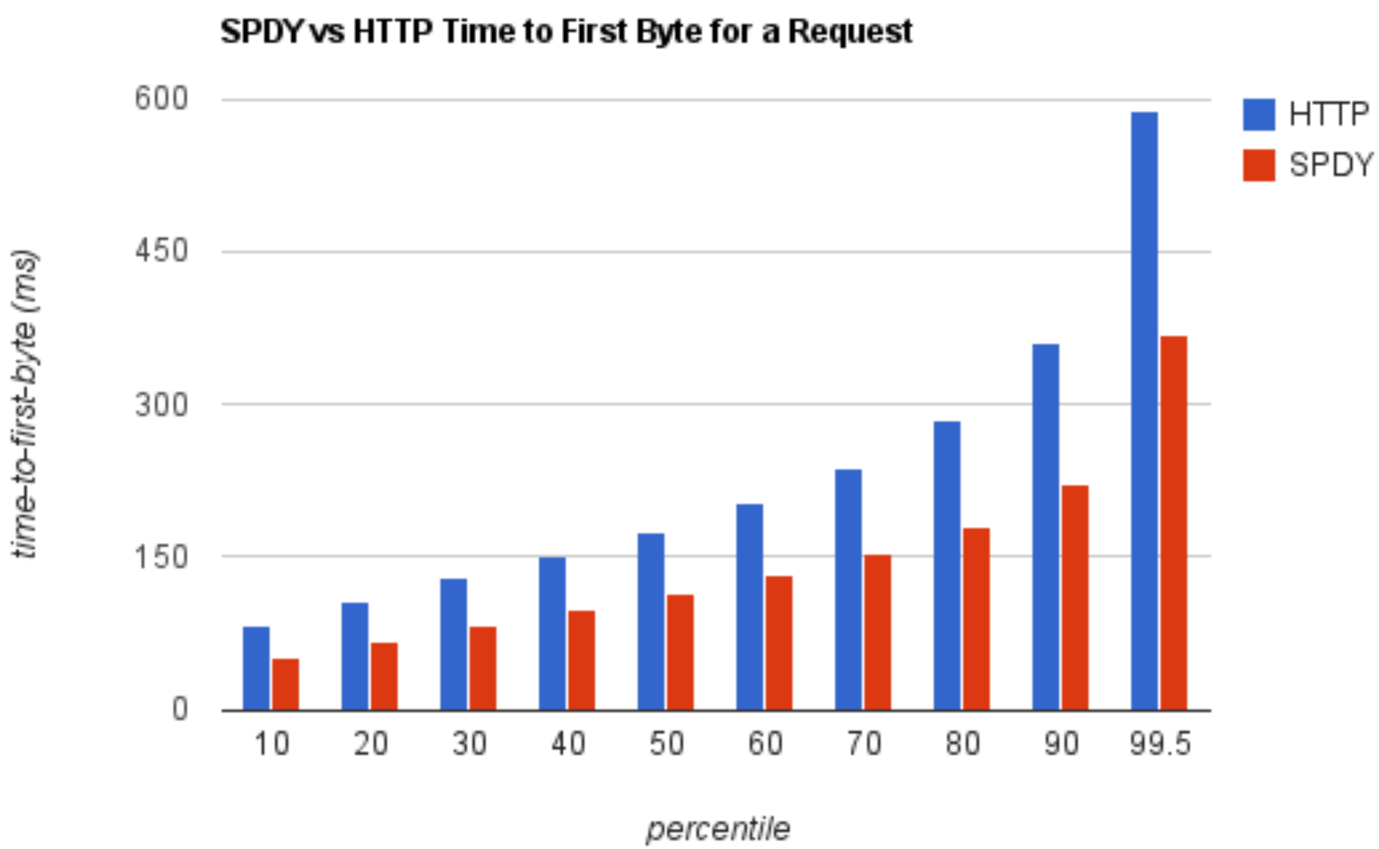
Reducing Download Bytes



Reducing Total Packets



Increasing Parallelism



The Single Connection Throttle

Throttle #1: CWND

Problem:

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

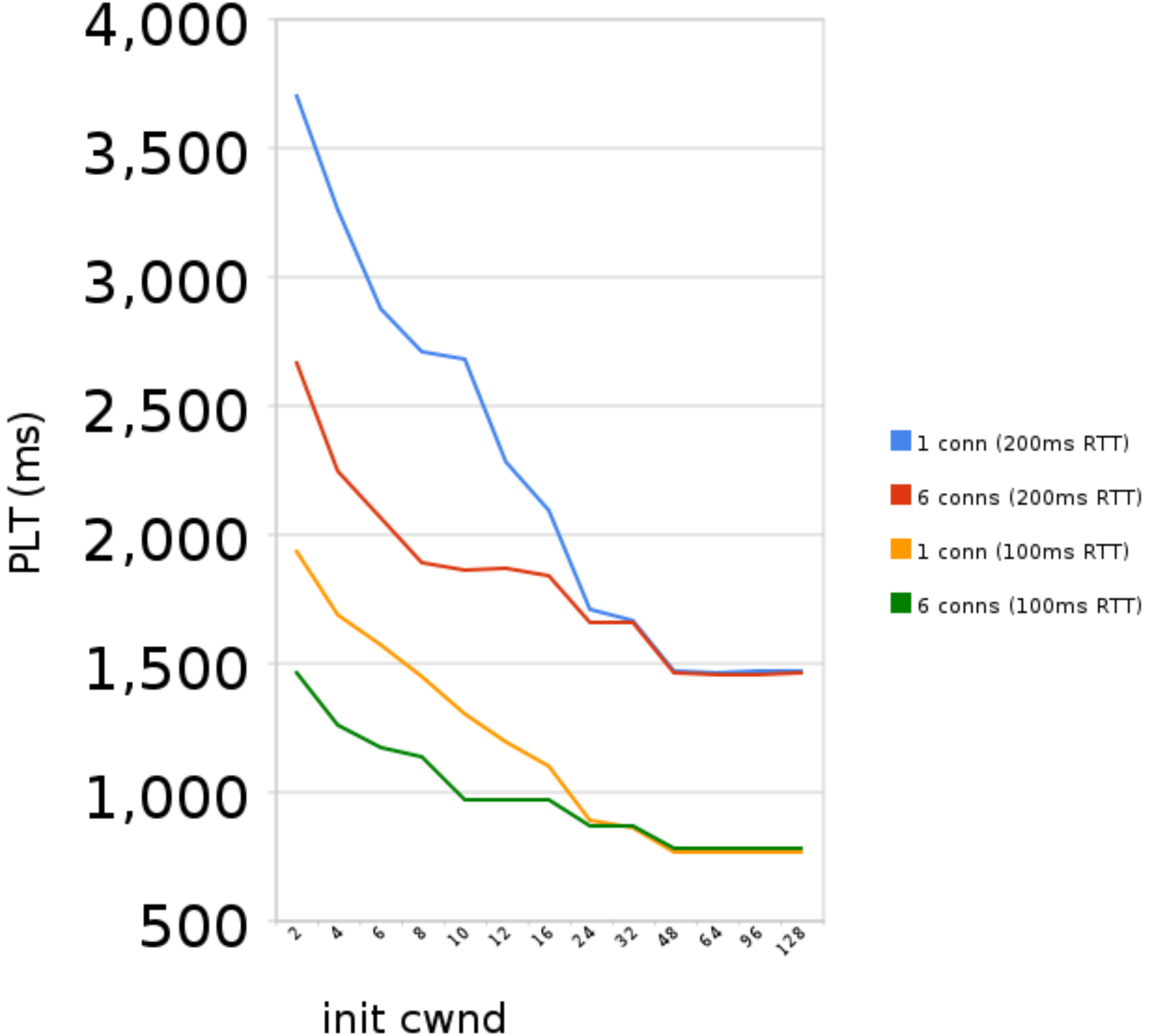
Workaround:

- 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

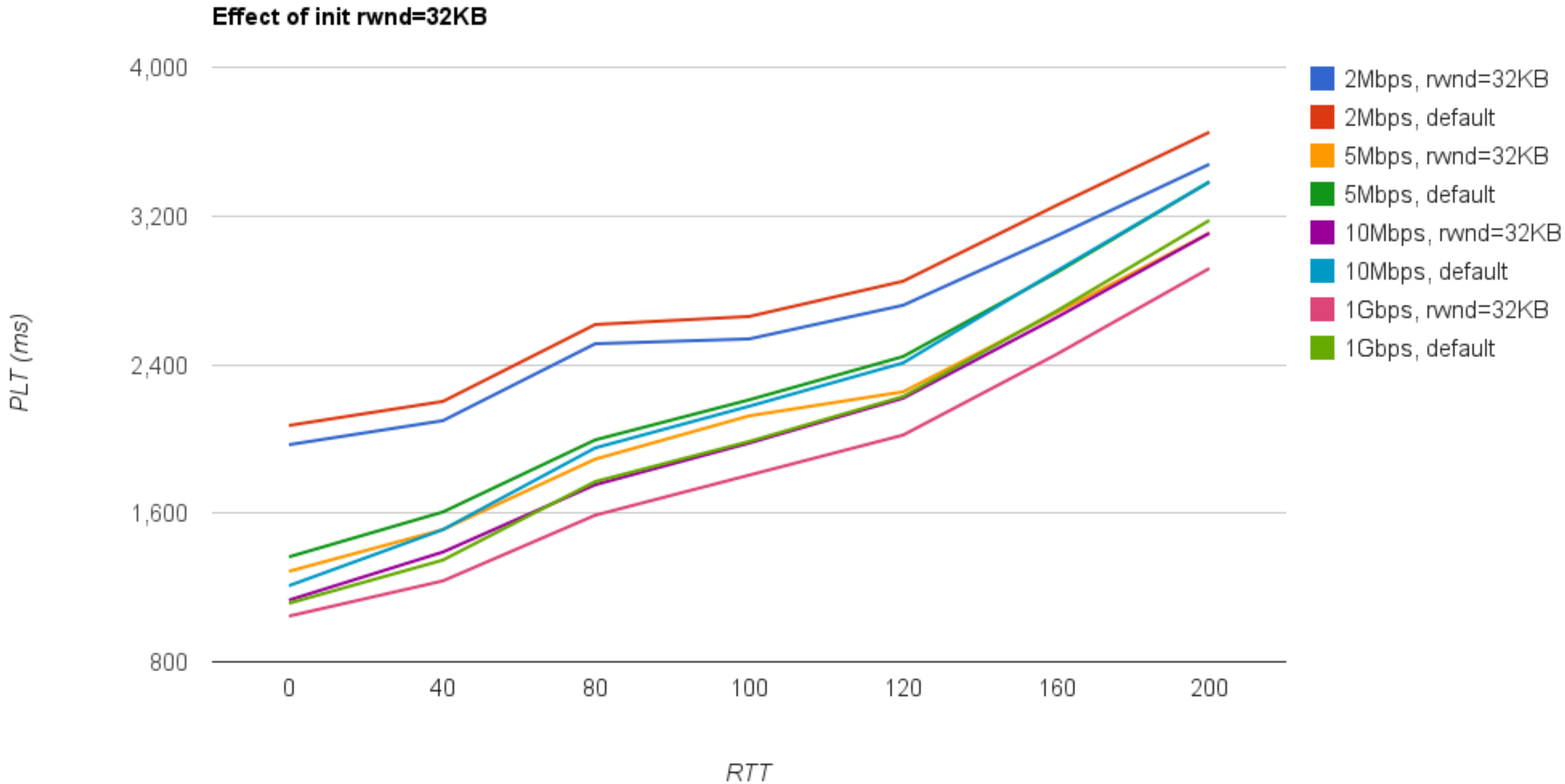
Problem:

- 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



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
```

Throttle #4: Congestion Control

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) == (~1/9th to 1/12th)

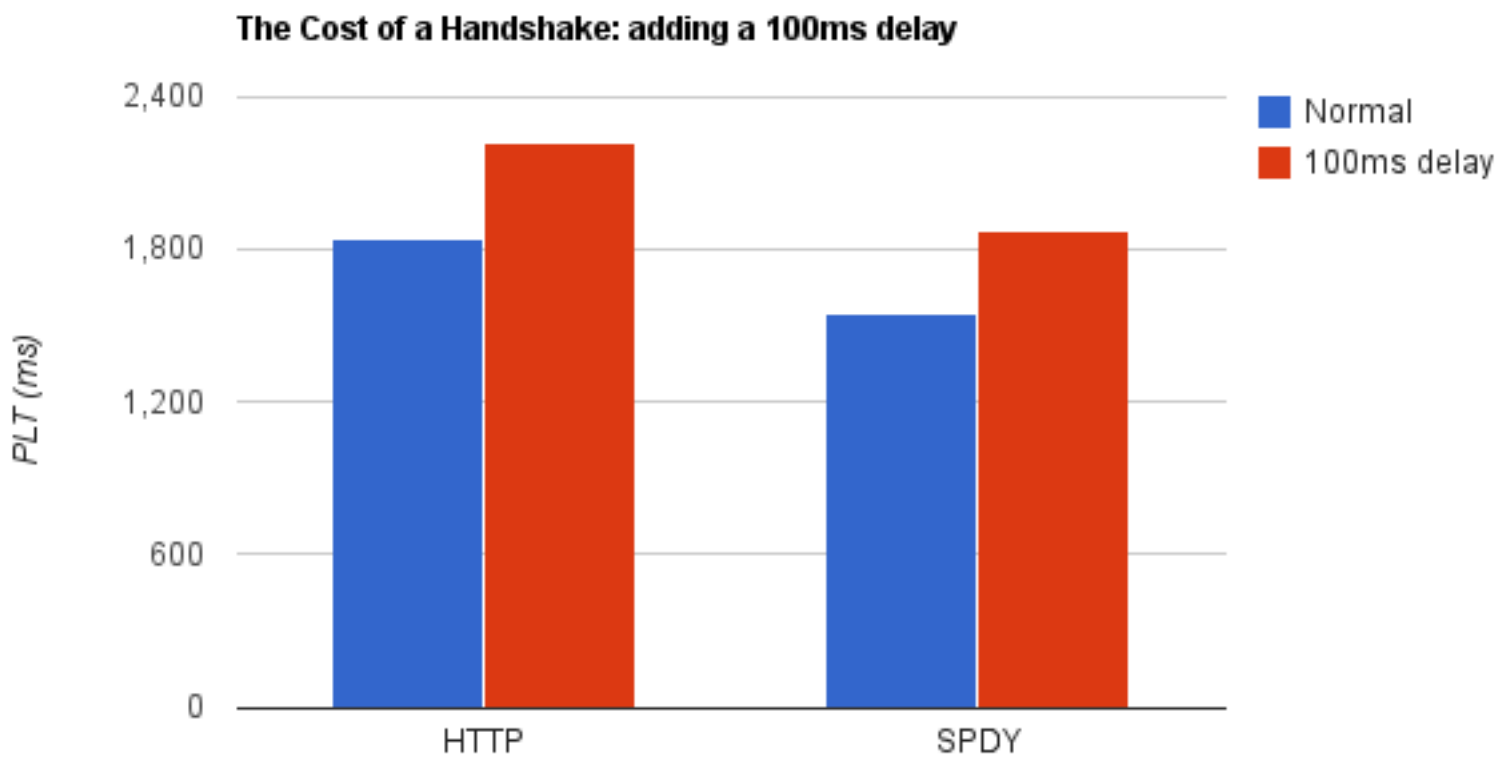
Workaround:

- 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?



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?