



SPDY, err... HTTP 2.0

what is it, how, why, and when?

Make the Web Fast, Google

- Improve end-user perceived latency
- Address the "head of line blocking"
- Not require multiple connections
- Retain the semantics of HTTP/1.1

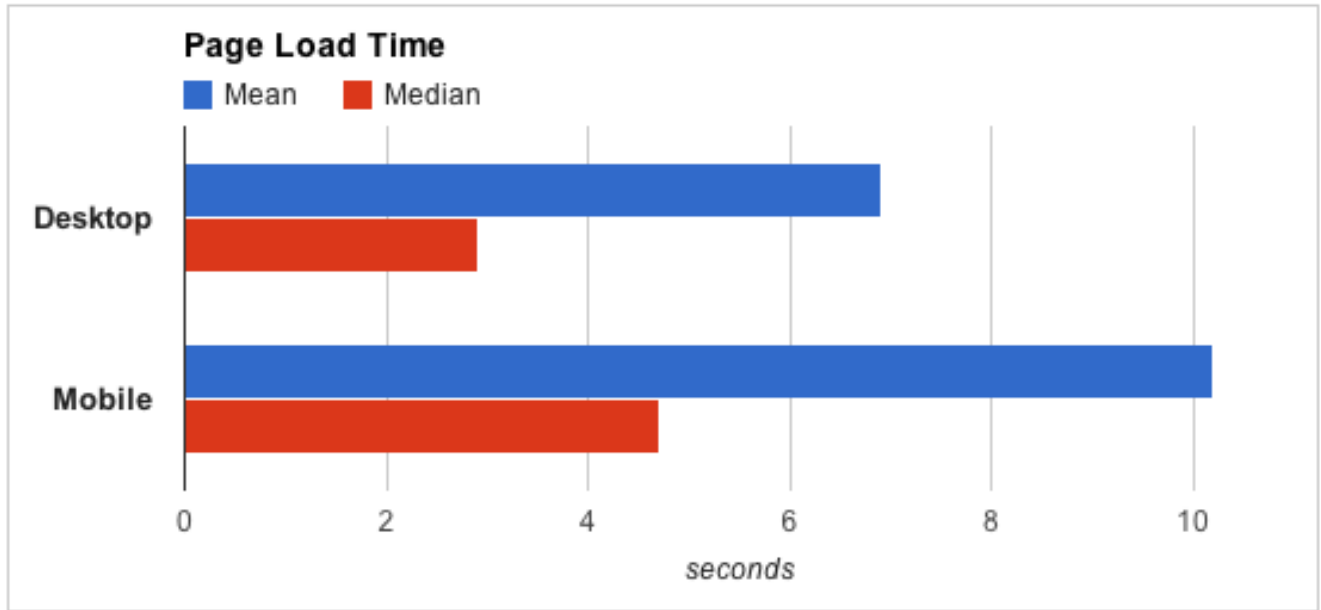
HTTP 2.0 / SPDY goals



Usability Engineering 101

Delay	User reaction
0 - 100 ms	Instant
100 - 300 ms	<i>Feels sluggish</i>
300 - 1000 ms	Machine is working...
1 s+	Mental context switch
10 s+	I'll come back later...





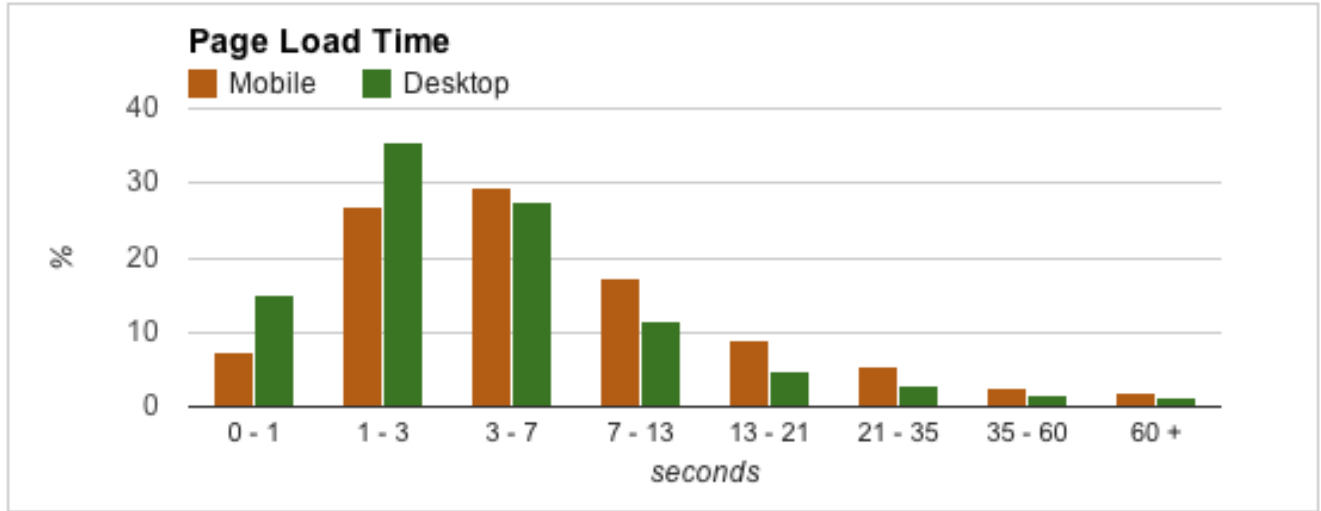
Desktop

Median: ~2.7s
 Mean: ~6.9s

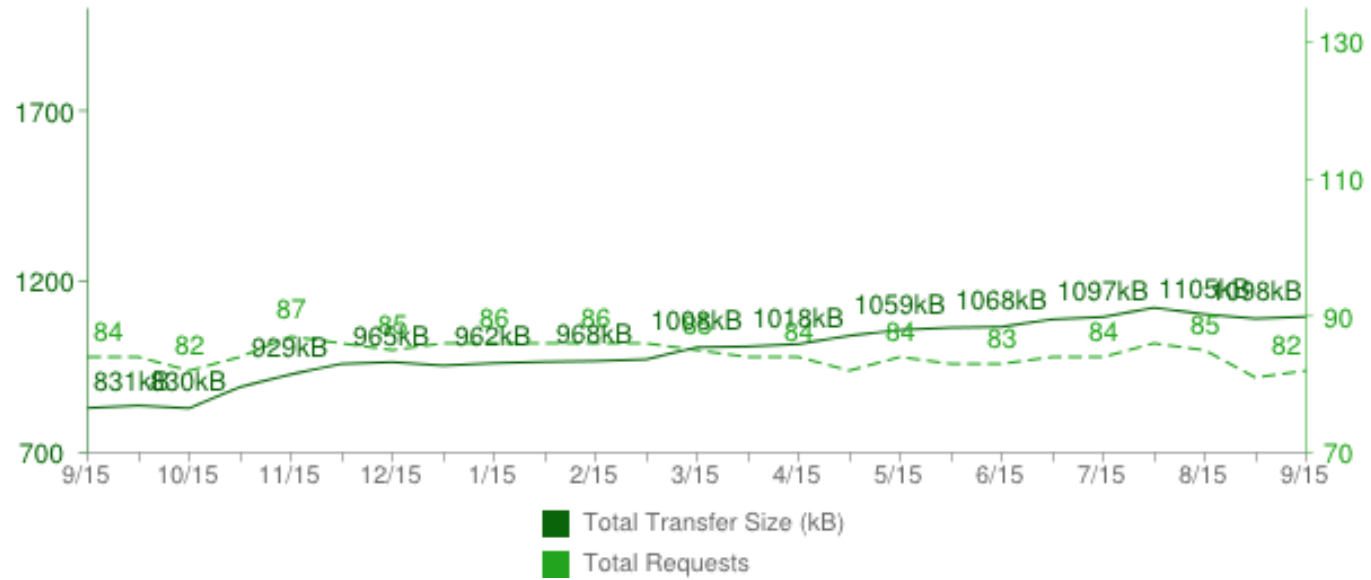
Mobile *

Median: ~4.8s
 Mean: ~10.2s

** optimistic*



Total Transfer Size & Total Requests



Content Type	Avg # of Requests	Avg size
HTML	8	44 kB
Images	53	635 kB
Javascript	14	189 kB
CSS	5	35 kB

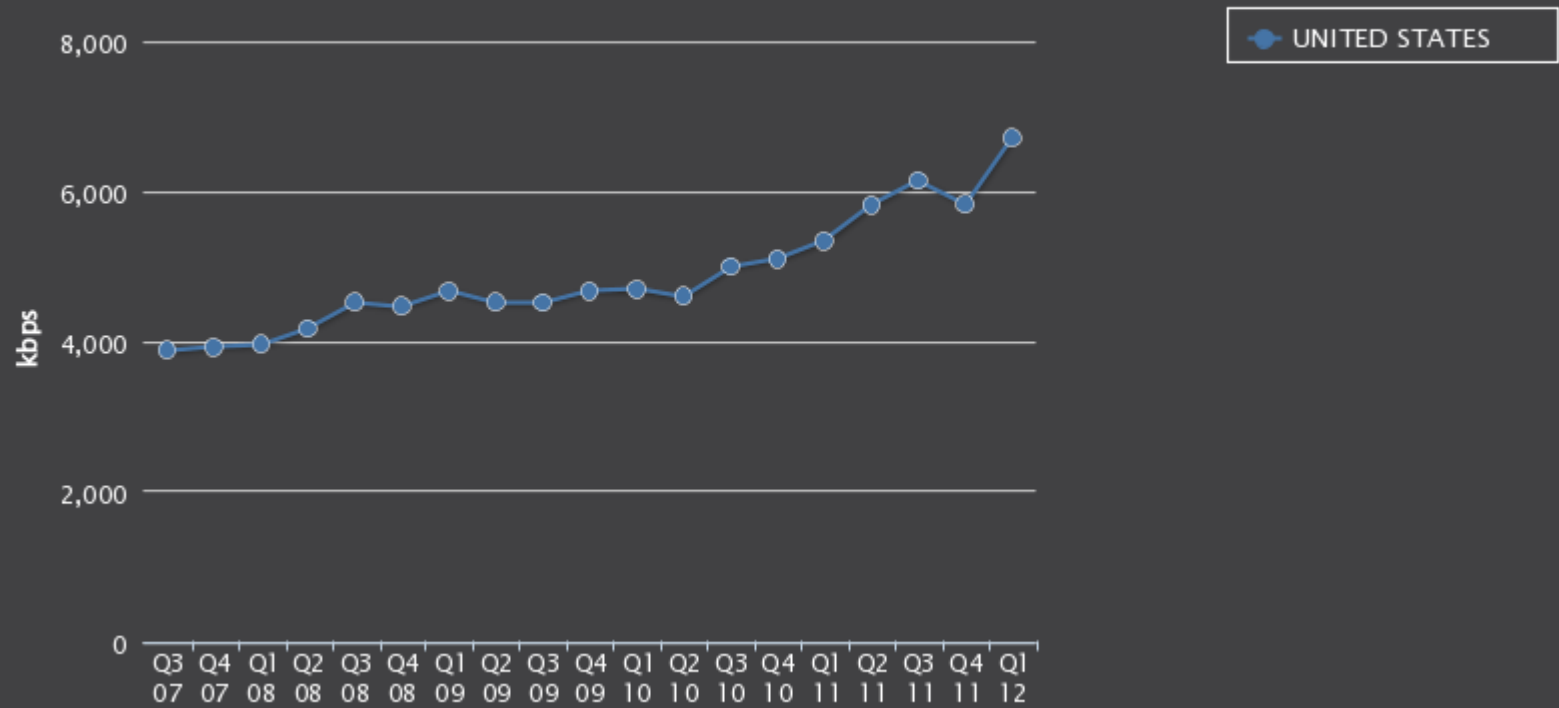




The network will save us?

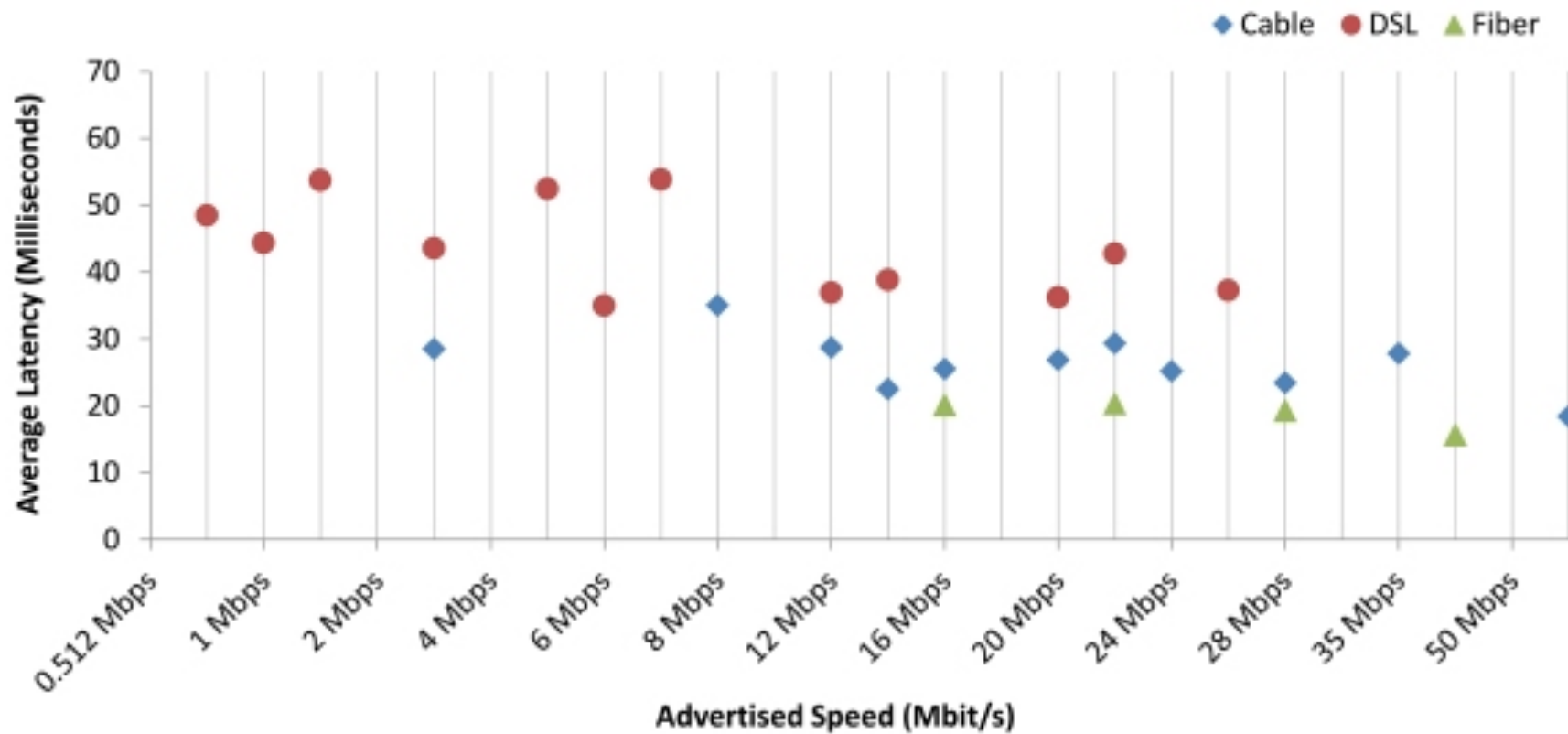
Right, right? Or maybe not...

Connection Speed



*Average US connection in Q1 2012: **6709 kbps***





Fiber-to-the-home services provided **18 ms** round-trip latency on average, while **cable-based** services averaged **26 ms**, and **DSL-based** services averaged **43 ms**. This compares to 2011 figures of 17 ms for fiber, 28 ms for cable and 44 ms for DSL.



Worldwide: ~100ms

US: ~50~60ms

Average RTT to Google in 2012 is...

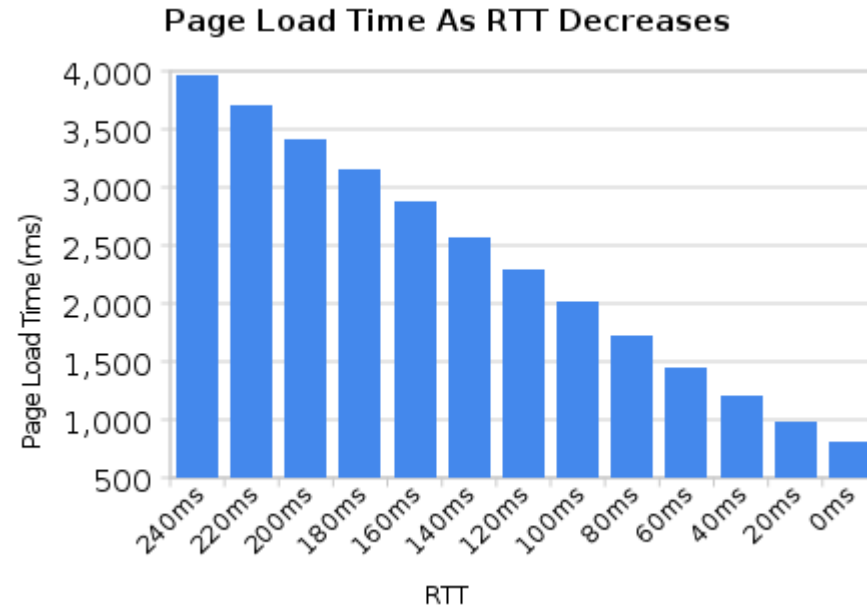
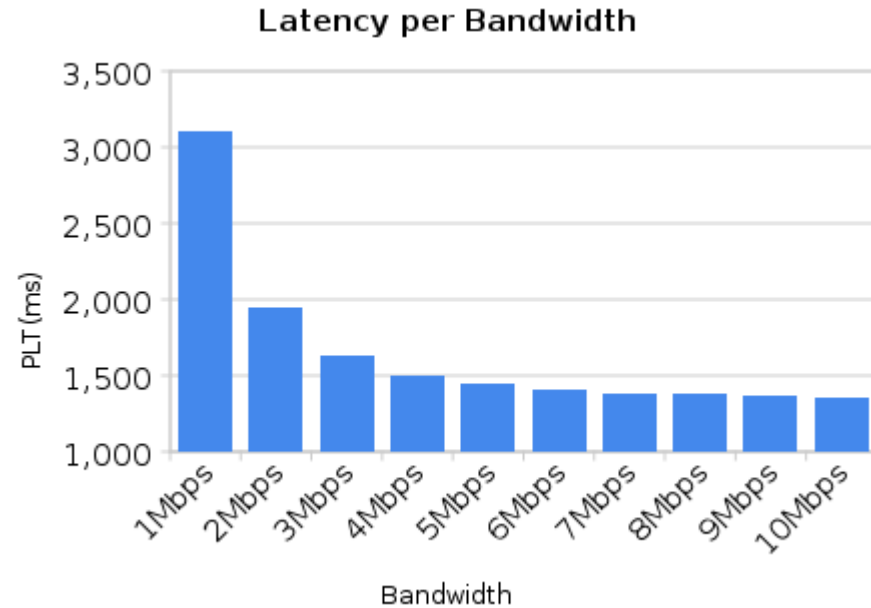




Bandwidth doesn't matter (*much*)

It's the latency, dammit!

PLT: latency vs. bandwidth



Average household in US is running on a **5 mbps+** connection. Ergo, **average consumer in US would not** see an improved PLT by upgrading their connection.

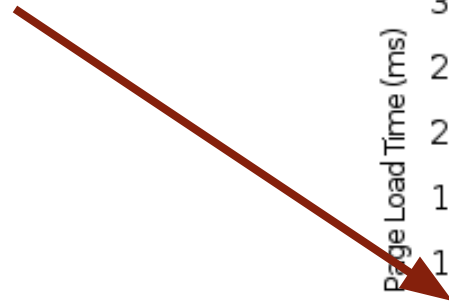


Mobile, oh Mobile...

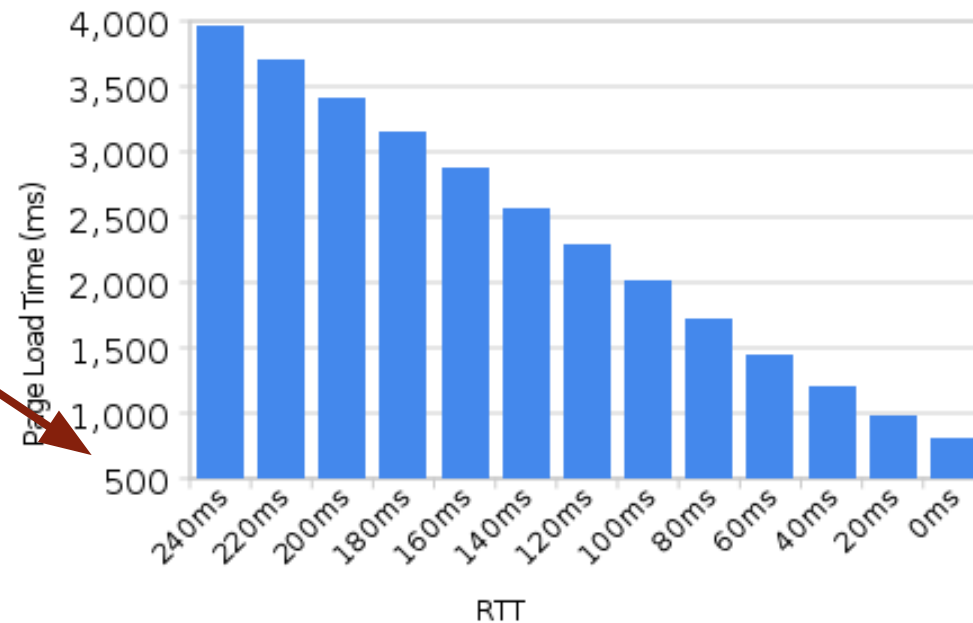
Users of the **Sprint 4G network** can expect to experience average speeds of 3Mbps to 6Mbps download and up to 1.5Mbps upload with an **average latency of 150ms**. On the **Sprint 3G network**, users can expect to experience average speeds of 600Kbps - 1.4Mbps download and 350Kbps - 500Kbps upload with an **average latency of 400ms**.

We stopped at 240ms!

(facepalm meme goes here...)



Page Load Time As RTT Decreases



- **Improving bandwidth is easy... ******

- Still lots of unlit fiber
- 60% of new capacity through upgrades
- "Just lay more cable" ...

- **Improving latency is expensive... impossible?**

- Bounded by the speed of light
- We're already within a small constant factor of the maximum
- Lay **shorter** cables!



\$80M / ms

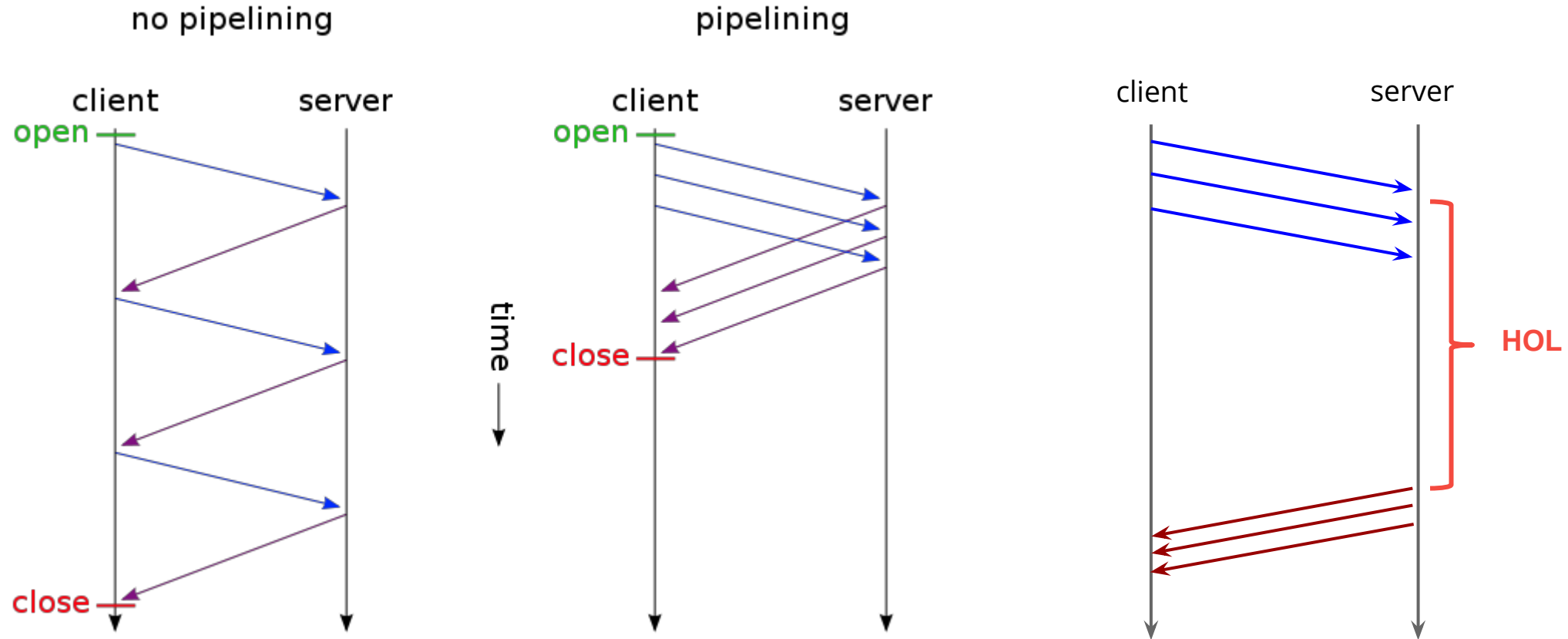




Why is latency the problem?

Remember that HTTP thing... yeah...

HTTP doesn't have multiplexing!



- **No pipelining:** request queuing
- **Pipelining*:** response queuing

- **Head of Line blocking**
 - It's a guessing game...
 - Should I wait, or should I pipeline?



Open multiple TCP connections!!!

Top Desktop			
name	score	PerfTiming	Connections per Hostname
<input type="checkbox"/> Chrome 20 →	12/16	yes	6
<input type="checkbox"/> Firefox 14 →	13/16	yes	6
<input type="checkbox"/> IE 8 →	7/16	no	6
<input type="checkbox"/> IE 9 →	12/16	yes	6
<input type="checkbox"/> Opera 12 →	10/16	no	6
<input type="checkbox"/> RockMelt 0.9 →	13/16	yes	6
<input type="checkbox"/> Safari 5.1 →	12/16	no	6

Top Mobile			
name	score	PerfTiming	Connections per Hostname
<input type="checkbox"/> Android 2.3 →	8/16	no	9
<input type="checkbox"/> Android 4 →	13/16	yes	6
<input type="checkbox"/> Blackberry 7 →	11/16	no	5
<input type="checkbox"/> Chrome Mobile 16 →	13/16	yes	6
<input type="checkbox"/> IEMobile 9 →	11/16	yes	6
<input type="checkbox"/> iPhone 4 →	10/16	no	4
<input type="checkbox"/> iPhone 5 →	10/16	no	6
<input type="checkbox"/> Nokia 950 →			
<input type="checkbox"/> Opera Mobile 12 →	11/16	no	8

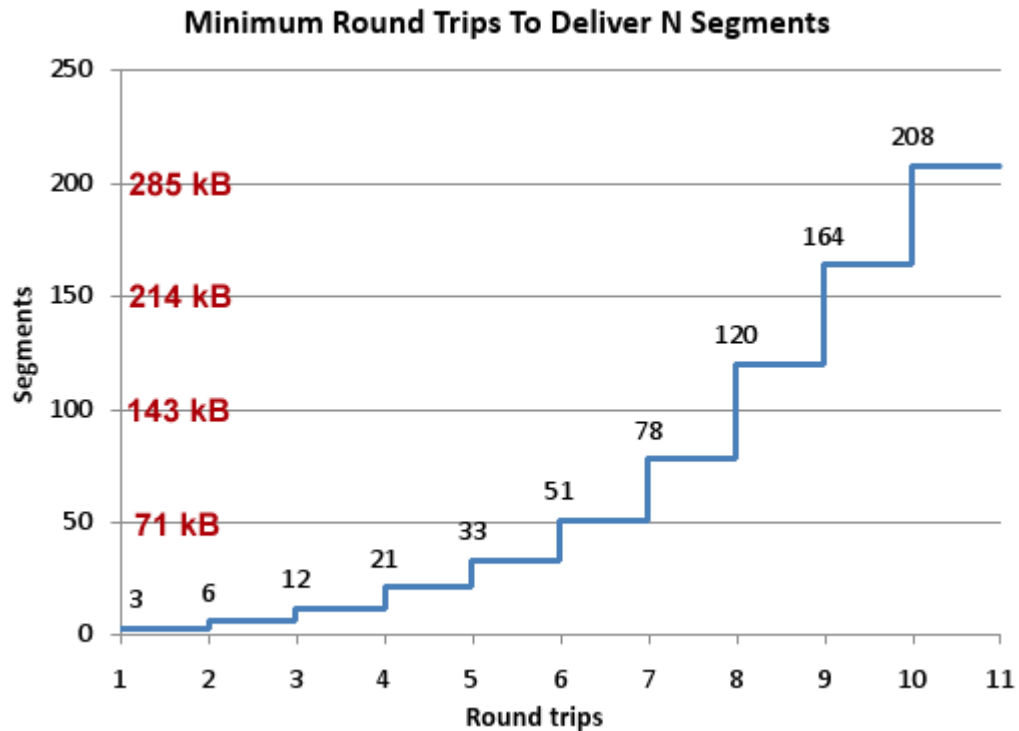
- **6 connections per host** on Desktop
- **6 connections per host** on Mobile (recent builds)

So what, what's the big deal?



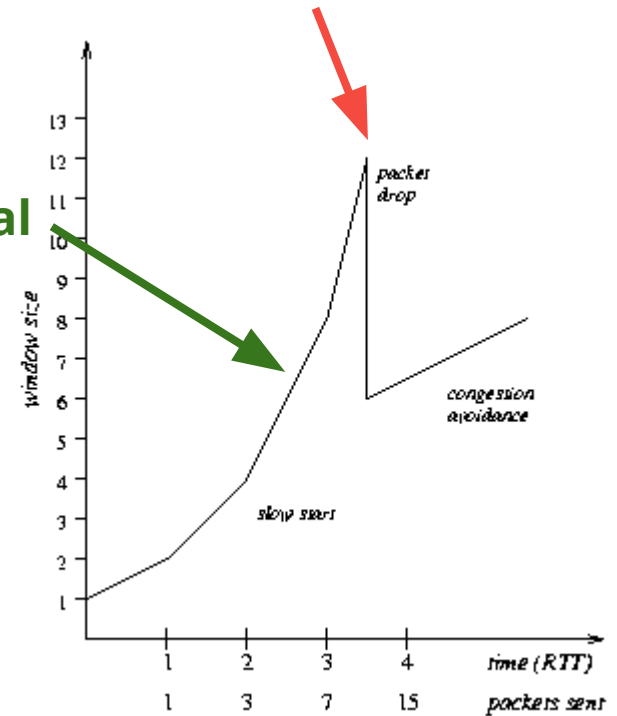
TCP Congestion Control & Avoidance...

- TCP is designed to probe the network to figure out the available capacity
- **TCP Slow Start** - feature, not a bug



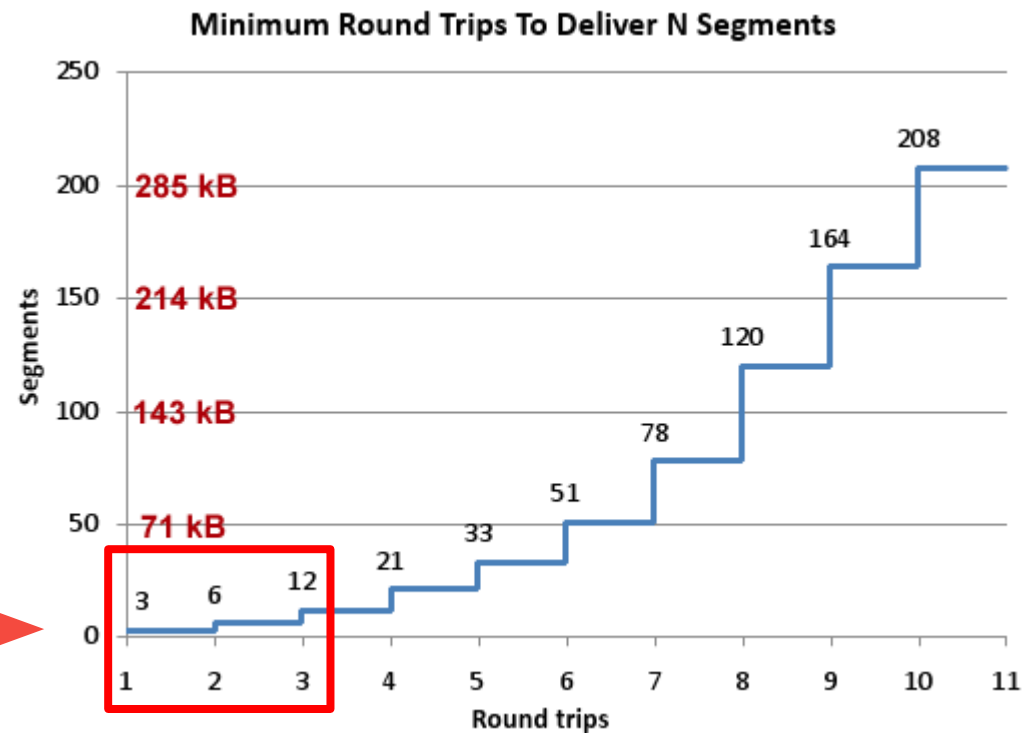
Exponential growth

Packet Loss



HTTP Archive says...

- 1098kb, 82 requests, ~30 hosts... ~14kb per request!
- Most HTTP traffic is composed of small, bursty, TCP flows



← Where we want to be

You are here →

1-3 RTT's



An Argument for Increasing TCP's Initial Congestion Window

Nandita Dukkkipati Tiziana Refice Yuchung Cheng Jerry Chu Natalia Sutin
Amit Agarwal Tom Herbert Arvind Jain
Google Inc.
{nanditad, tiziana, ycheng, hkchu, nsutin, aagarwal, therbert, arvind}@google.com

ABSTRACT

TCP flows start with an initial congestion window of at most three segments or about 4KB of data. Because most Web transactions are short-lived, the initial congestion window is

for standard Ethernet MTUs (approximately 4KB) [5]. The majority of connections on the Web are short-lived and finish before exiting the slow start phase, making TCP's initial congestion window (*init_cwnd*) a crucial parameter in deter-

a c
can
dra
valu
cha
I
tion
larg
ben

of network bandwidth, round-trip time (RTT), bandwidth-delay product (BDP) and nature of applications. We show

initial
short

TCP's
per a
obally
width
(Kbps)
appli-
load of

Update CWND from 3 to 10 segments, or ~14960 bytes

Default size on *Linux 2.6.33+* - double check yours!

Web pages. Popular Web browsers, including IE8 [2], Fire-





Let's talk about SPDY

err... HTTP 2.0!

SPDY is HTTP 2.0... *sort of*...

- HTTPBis Working Group met in Vancouver in late July
- Adopted **SPDY v2 as starting point** for HTTP 2.0

HTTP 2.0 Charter

1. **Done** Call for Proposals for HTTP/2.0
2. **Oct 2012** First WG draft of HTTP/2.0, based upon draft-mbelshe-httpbis-spdy-00
3. **Apr 2014** Working Group Last call for HTTP/2.0
4. **Nov 2014** Submit HTTP/2.0 to IESG for consideration as a Proposed Standard



*It's important to understand that SPDY isn't being adopted as HTTP/2.0; rather, that it's the **starting point** of our discussion, to avoid a laborious start from scratch.*

- Mark Nottingham (chair)



It is expected that HTTP/2.0 will...

- Substantially and measurably improve end-user perceived latency over HTTP/1.1
- Address the "head of line blocking" problem in HTTP
- Not require multiple connections to a server to enable parallelism, thus improving its use on mobile devices
- Retain the semantics of HTTP/1.1, including (but not limited to)
 - HTTP methods
 - Status Codes
 - URIs
 - Header fields
- Clearly define how HTTP/2.0 interacts with HTTP/1.x
 - especially in intermediaries (both 2->1 and 1->2)
- Clearly identify any new extensibility points and policy for their appropriate use

Make things better

Build on HTTP 1.1

Be extensible



A litany of problems.. and "workarounds"...

1. **Concatenating files**

- JavaScript, CSS
- Less modular, large bundles

2. **Spriting images**

- What a pain...

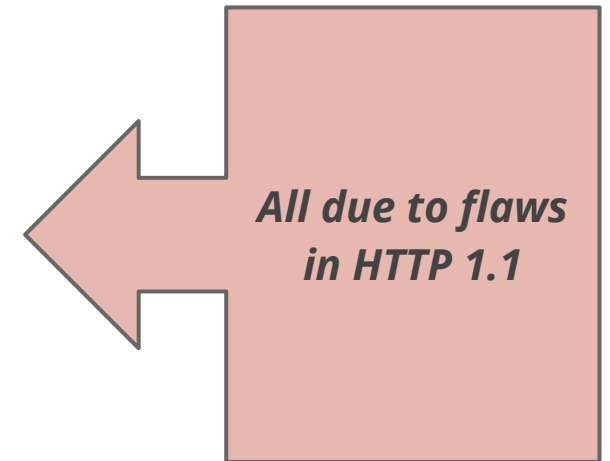
3. **Domain sharding**

- Congestion control who? 30+ parallel requests --- *Yeehaw!!!*

4. **Resource inlining**

- TCP connections are expensive!

5. ...





So, what's a developer to do?

Fix HTTP 1.1! Use SPDY in the meantime...

*... we're not replacing all of HTTP — the methods, status codes, and most of the headers you use today will be the same. Instead, we're **re-defining how it gets used “on the wire” so it's more efficient**, and so that it is more gentle to the Internet itself*

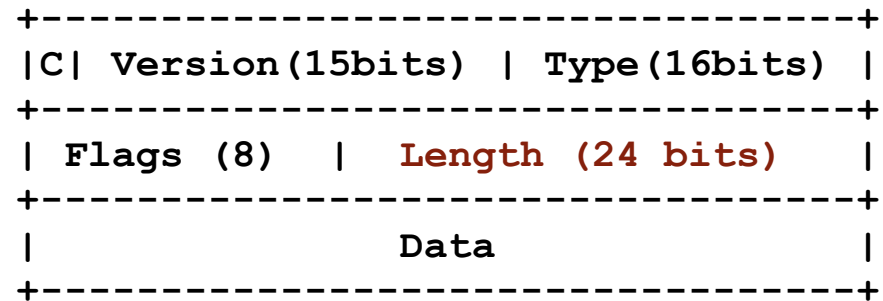
- Mark Nottingham (chair)



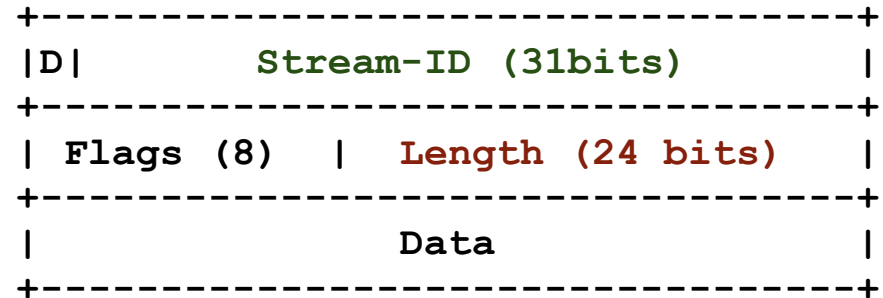
SPDY in a Nutshell

- One TCP connection
- Request = Stream
- Streams are multiplexed
- Streams are prioritized
- Binary framing
- Length-prefixed
- Control frames
- Data frames

Control Frame:

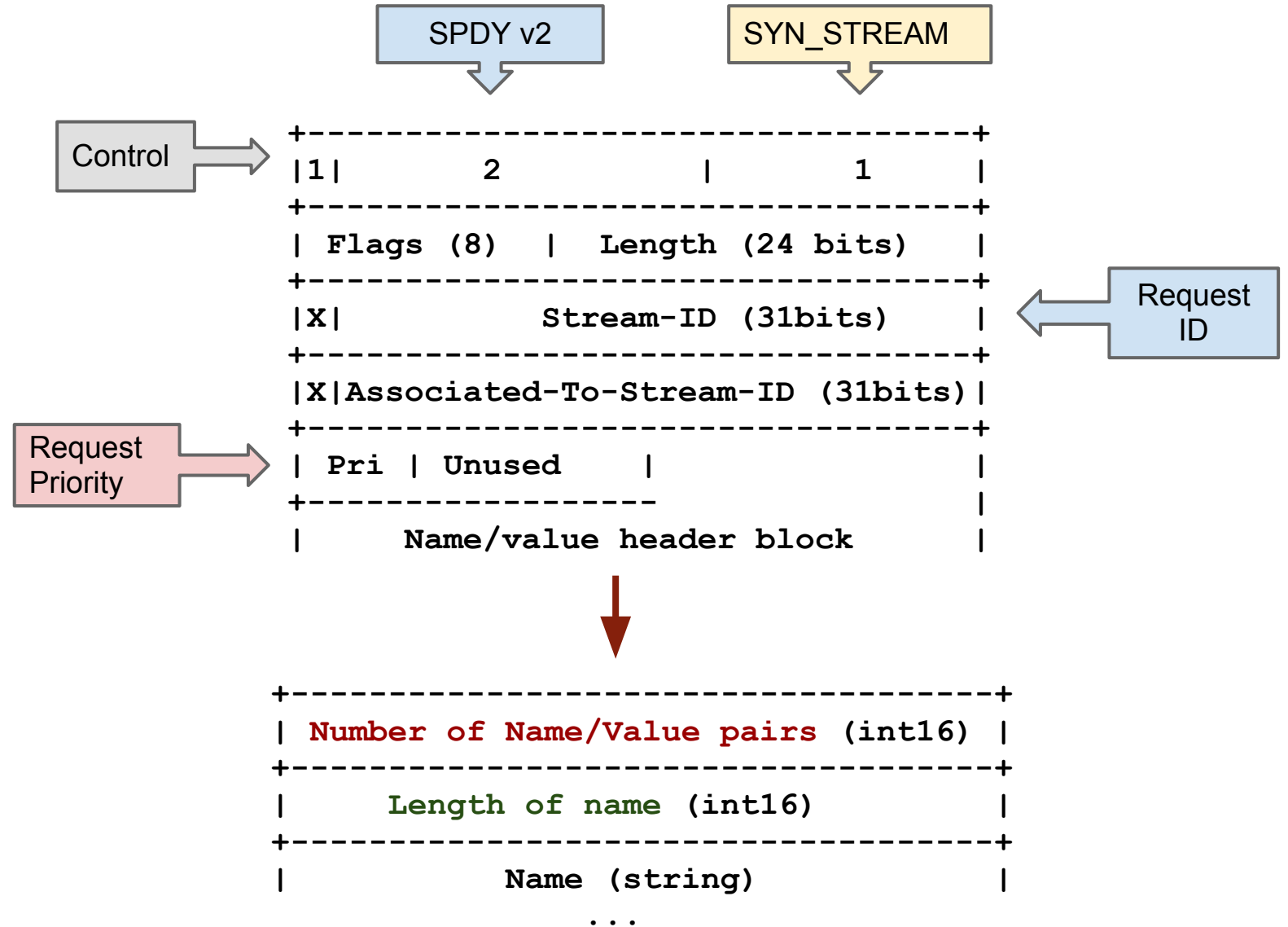


Data Frame:



SYN_STREAM

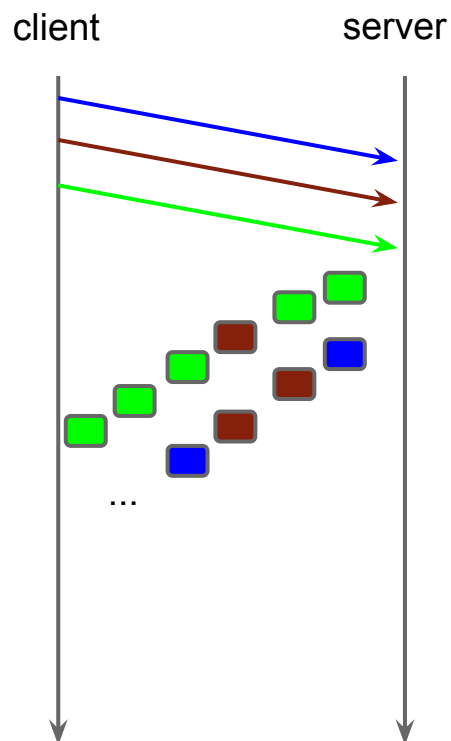
- Server SID: even
- Client SID: odd
- Associated-To: push *
- Priority: higher, better
- Length prefixed headers



*** Much of this may (will, probably) change



SPDY in action



- Full request & response multiplexing
- Mechanism for request prioritization
- Many small files? No problem
- Higher TCP window size
- More efficient use of server resources
- TCP Fast-retransmit for faster recovery

Anti-patterns

- Domain sharding
 - *Now we need to unshard - doh!*



Speaking of HTTP Headers...

```
curl -vv -d '{"msg": "oh hai"}' http://www.igvita.com/api
```

```
> POST /api HTTP/1.1
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0)
libcurl/7.24.0 OpenSSL/0.9.8r zlib/1.2.5
> Host: www.igvita.com
> Accept: */*
> Content-Length: 16
> Content-Type: application/x-www-form-urlencoded

< HTTP/1.1 204
< Server: nginx/1.0.11
< Content-Type: text/html; charset=utf-8
< Via: HTTP/1.1 GWA
< Date: Thu, 20 Sep 2012 05:41:30 GMT
< Expires: Thu, 20 Sep 2012 05:41:30 GMT
< Cache-Control: max-age=0, no-cache
....
```

- Average request / response header overhead: **800 bytes**
- No compression for headers in HTTP!
- Huge overhead
- **Solution:** compress the headers!
 - gzip all the headers
 - header registry
 - connection-level vs. request-level
- **Complication:** intermediate proxies **



SPDY Server Push

Premise: server can push resources to client

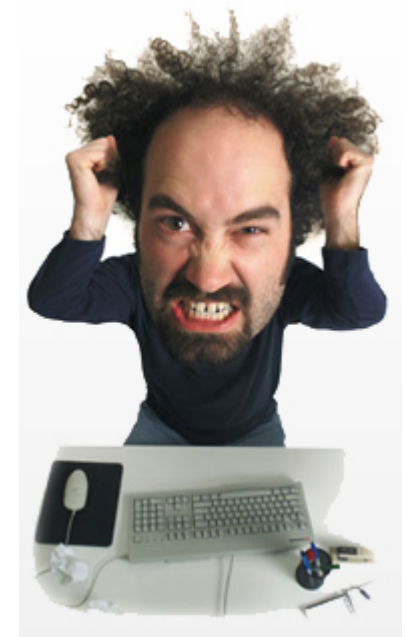
- **Concern: *but I don't want the data! Stop it!***
 - Client can cancel SYN_STREAM if it doesn't the resource
- Resource goes into browsers cache (no client API)

Newsflash: we are already using "server push"

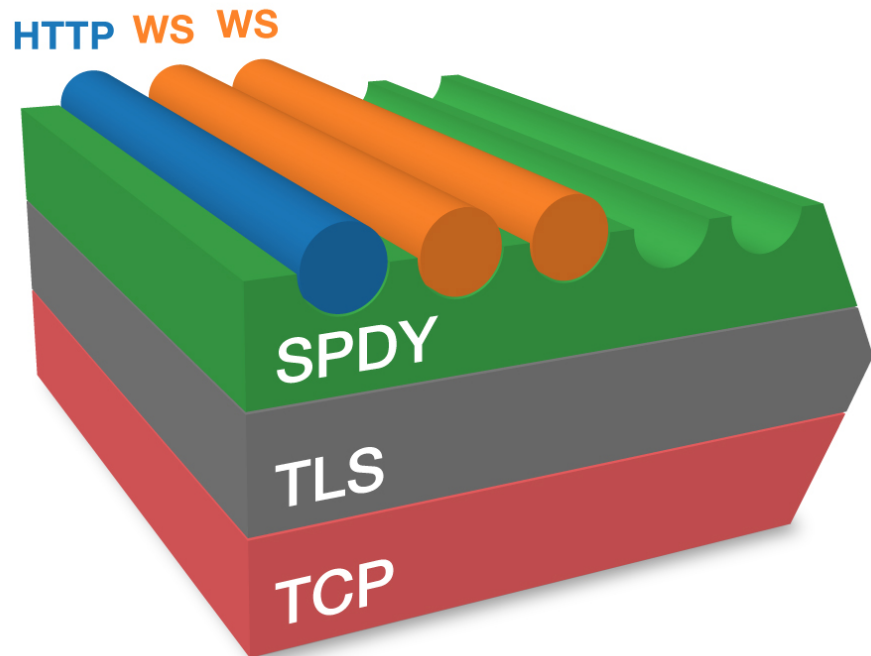
- Today, we call it "inlining"
- Inlining works for unique resources, bloats pages otherwise

Advanced use case: forward proxy (ala Amazon's Silk)

- Proxy has full knowledge of your cache, can intelligently push data to the client



Encrypt all the things!!!



SPDY runs over TLS

- Philosophical reasons
- Political reasons
- **Pragmatic + deployment reasons - Bing!**

Observation: intermediate proxies get in the way

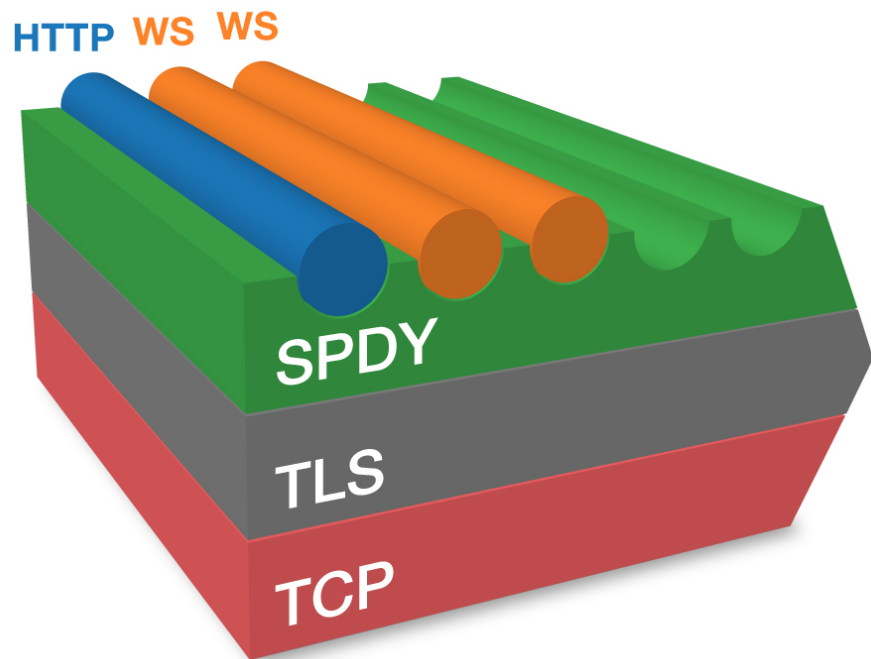
- Some do it intentionally, many unintentionally
- *Ex: Antivirus / Packet Inspection / QoS / ...*

SDHC / WebSocket: No TLS works.. in 80-90% of cases

- 10% of the time things fail for no discernable reason
- In practice, any large WS deployments run as WSS



But isn't TLS *slow*?



CPU

"On our production frontend machines, **SSL/TLS accounts for less than 1% of the CPU load**, less than 10KB of memory per connection and less than 2% of network overhead."

- Adam Langley (Google)

Latency

- [TLS Next Protocol Negotiation](#)
 - Protocol negotiation as part of TLS handshake
- TLS False Start
 - reduce the number of RTTS for full handshake from two to one
- TLS Fast Start
 - reduce the RTT to zero
- Session resume, ...



Who supports SPDY?

- **Chrome**, since forever..
 - Chrome on Android + iOS
- **Firefox 13+**
- Next stable release of **Opera**



Server

- mod_spdy (Apache)
- nginx
- Jetty, Netty
- node-spdy
- ...

3rd parties

- Twitter
- Wordpress
- Facebook*

- Akamai
- Contendo
- F5 SPDY Gateway
- Strangeloop
- ...

All Google properties

- Search, GMail, Docs
- GAE + SSL users
- ...



SPDY FAQ

- **Q: Do I need to modify my site to work with SPDY / HTTP 2.0?**
- A: No. But you can optimize for it.

- **Q: How do I optimize the code for my site or app?**
- A: "Unshard", stop worrying about silly things (like spriting, etc).

- **Q: Any server optimizations?**
- A: Yes!
 - CWND = 10
 - Check your SSL certificate chain (length)
 - TLS resume, terminate SSL close and early
 - Disable slow start on idle

- **Q: Sounds complicated, are there drop-in solutions?**
- A: Yes! mod_spdy, nginx, GAE, ...

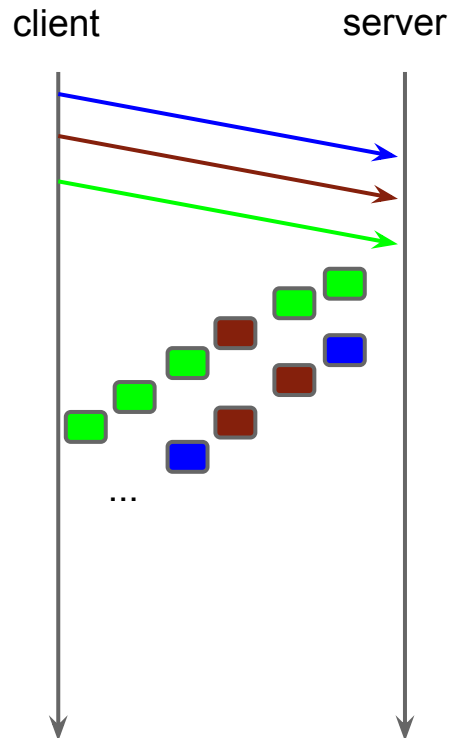




But wait, there is a gotcha!

there is always a gotcha...

~~HTTP Head of line blocking...~~ TCP Head of line blocking



- TCP: in-order, reliable delivery...
 - *what if a packet is lost?*
- **~1~2% packet loss rate**
 - CWND's get chopped
 - Fast-retransmit helps, but..
 - SPDY stalls
- High RTT links are a problem too
 - Traffic shaping
 - ISP's remove dynamic window scaling

Something to think about...





Can haz SPDY?

Apache, nginx, Jetty, node.js, ...

Apache + SPDY



Project Home [Wiki](#) [Issues](#) [Source](#)

Summary [People](#)

Project Information

+108 Recommend this on Google

★ Starred by 259 users
[Project feeds](#)

Code license
[Apache License 2.0](#)

Labels
apache, SPDY

Members
[bmcqu...@google.com](#),
[mdste...@google.com](#),
[ls...@google.com](#)
3 committers

Featured

Wiki pages
[ConfigOptions](#)
[GettingStarted](#)
[Show all »](#)

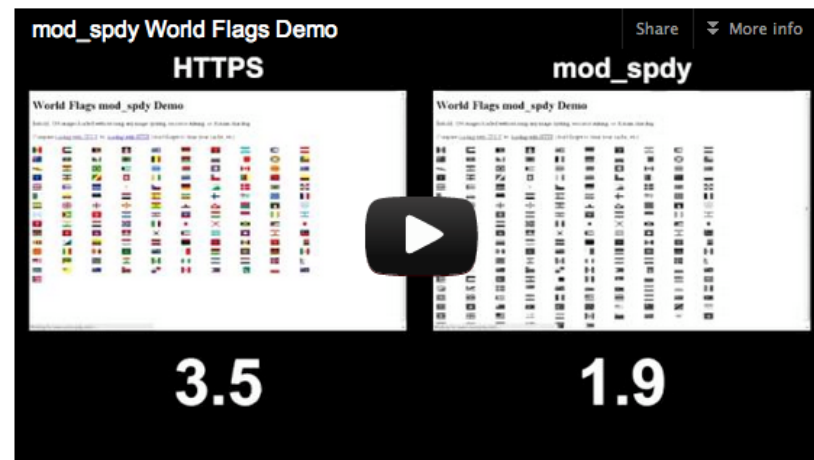
Links

Groups
[mod-spdy-discuss](#)

mod_spdy

mod_spdy is a SPDY module for Apache 2.2 that allows your web server to take advantage of SPDY features like stream multiplexing and header compression. This is the open source home for mod_spdy. You can also download [Debian and RPM packages](#) or [compile mod-spdy from source](#).

Potential speedup from mod_spdy



- mod_spdy is an open-source Apache module
- drop in support for SPDY

Installing mod_spdy in your Apache server

1

```
$ sudo dpkg -i mod-spdy-*.deb  
$ sudo apt-get -f install  
$ sudo a2enmod spdy  
  
$ sudo service apache2 restart
```

2

Profit

- Configure mod_proxy + mod_spdy: <https://gist.github.com/3817065>
 - Enable SPDY for any backend app-server
 - SPDY connection is terminated by Apache, and Apache speaks HTTP to your app server



Building nginx with SPDY support

1

```
$ wget http://openssl.org/source/openssl-1.0.1c.tar.gz
$ tar -xvf openssl-1.0.1c.tar.gz

$ wget http://nginx.org/download/nginx-1.3.4.tar.gz
$ tar xvfz nginx-1.3.4.tar.gz
$ cd nginx-1.3.4

$ wget http://nginx.org/patches/spdy/patch.spdy.txt
$ patch -p0 < patch.spdy.txt
```

2

```
$ ./configure ... --with-openssl='/software/openssl/openssl-1.0.1c'
$ make
$ make install
```

3

Profit



node.js + SPDY

1

```
var spdy = require('spdy'),
    fs = require('fs');

var options = {
  key: fs.readFileSync(__dirname + '/keys/spdy-key.pem'),
  cert: fs.readFileSync(__dirname + '/keys/spdy-cert.pem'),
  ca: fs.readFileSync(__dirname + '/keys/spdy-csr.pem')
};

var server = spdy.createServer(options, function(req, res) {
  res.writeHead(200);
  res.end('hello world!');
});

server.listen(443);
```

2

Profit



Jetty + SPDY

I <3 Java :-)

- 1 Copy X pages of maven XML configs
- 2 Add NPN jar to your classpath
- 3 Wrap HTTP requests in SPDY, or copy copius amounts of XML...
- ...
- N Profit



Am I SPDY?

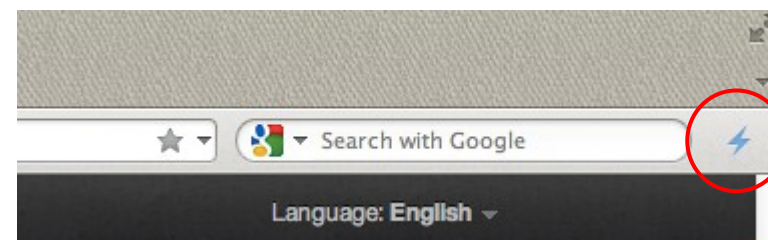
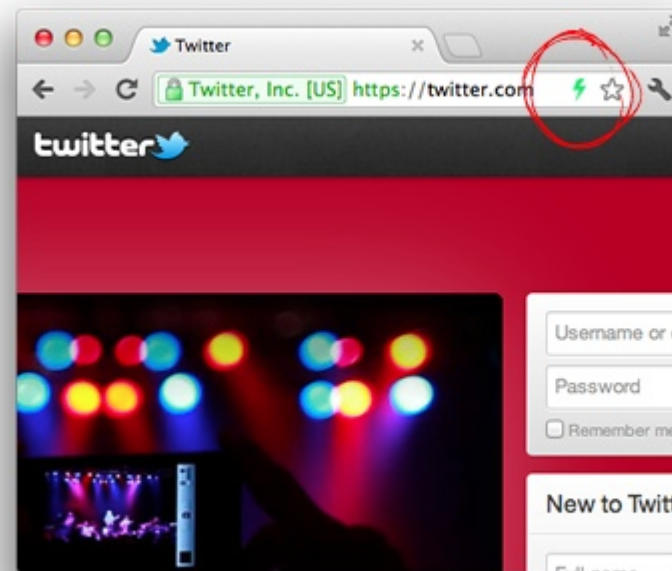
How do I know, how do I debug?

SPDY indicator(s)

- [Chrome SPDY indicator](#)
- [Firefox indicator](#)
- [Opera indicator](#)

In Chrome console:

```
> window.chrome.loadTimes()
< ▼ Object
  commitLoadTime: 1350252136.934823
  finishDocumentLoadTime: 1350252137.397209
  finishLoadTime: 1350252137.529396
  firstPaintAfterLoadTime: 1350252137.611959
  firstPaintTime: 1350252137.523084
  navigationType: "Other"
  npnNegotiatedProtocol: "spdy/3"
  requestTime: 0
  startLoadTime: 1350252135.83449
  wasAlternateProtocolAvailable: false
  wasFetchedViaSpdy: true
  wasNpnNegotiated: true
  __proto__: Object
```



chrome://net-internals#spdy

Capturing network events (185) Stop Reset

Capture
Export
Import
Proxy
Events
Timeline
DNS
Sockets
SPDY
HTTP Pipelining
HTTP Cache
Tests
HSTS
Prerender

SPDY Status

- SPDY Enabled: true
- Use Alternate Protocol: true
- Force SPDY Always: false
- Force SPDY Over SSL: true
- Next Protocols: http/1.1,spdy/2,spdy/3

SPDY sessions

[View live SPDY sessions](#)

Host	Proxy	ID	Protocol Negotiated	Active streams	Unclaimed pushed	Max	Initiated	Pushed	Pushed and claimed
0.docs.google.com:443	direct://	305272	spdy/3	1	0	100	80	0	0
clients4.google.com:443 apis.google.com:443 cbks0.google.com:443 clients1.google.com:443 clients2.google.com:443 docs.google.com:443 drive.google.com:443 encrypted-tbn0.gstatic.com:443 encrypted-tbn1.gstatic.com:443 encrypted-tbn2.gstatic.com:443 encrypted-tbn3.gstatic.com:443 khms0.google.com:443 khms1.google.com:443 maps-api-ssl.google.com:443	direct://	280013	spdy/3	0	0	100	3471	0	0



Try it @ <https://spdy.io/> - open the link, then head to net-internals & click on stream-id

HTTP 2.0 will ...

- Improve end-user perceived latency
- Address the "head of line blocking"
- Not require multiple connections
- Retain the semantics of HTTP/1.1

In the meantime, SPDY is here (FF, Opera, Chrome) ...

- Apache, nginx, node.js, Jetty, ...
- Drop in modules, no modifications needed to your site
- You **can** optimize your site for SPDY / HTTP 2.0



Thanks! Questions?