~~SPDY~~, err... HTTP 2.0

for fun and profit...

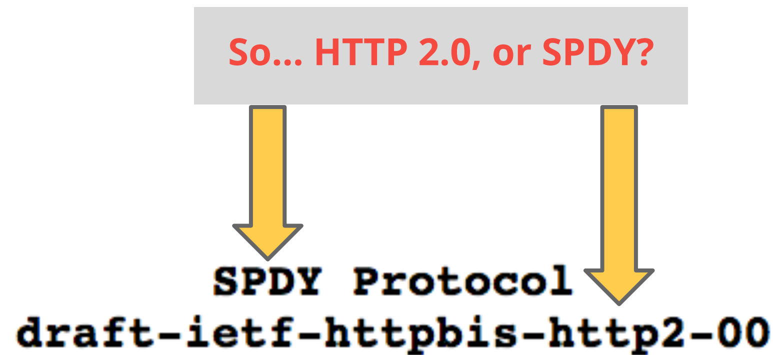
Ilya Grigorik - @igrigorik, gplus.to/igrigorik
Make the Web Fast, Google

tools.ietf.org/html/draft-ietf-httpbis-http2-00

[Docs] [txt|pdf|xml|html] [Tracker] [WG] [Email] [Nits]

Versions: 00

HTTPbis Working Group | M. Belshe
Internet-Draft | Twist
Expires: June 1, 2013 | R. Peon
Google, Inc
M. Thomson, Ed.
Microsoft
A. Melnikov, Ed.
Isode Ltd
November 28, 2012



*"This draft is a work-in-progress, and **does not yet reflect Working Group consensus**... This **first draft uses the SPDY Protocol as a starting point**, as per the Working Group's charter. Future drafts will add, remove and change text, based upon the Working Group's decisions."*

http://tools.ietf.org/html/draft-ietf-httpbis-http2-00

*"a protocol designed for **low-latency transport of content** over the World Wide Web"*

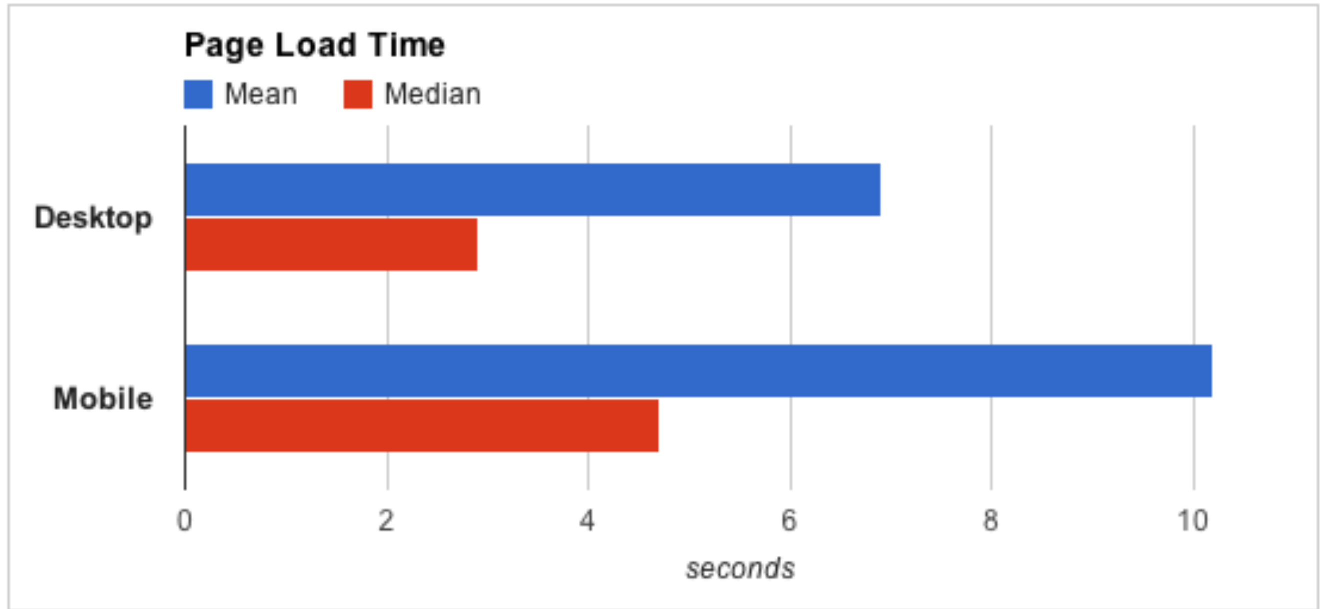
- Improve end-user perceived latency
- Address the "head of line blocking"
- Not require multiple connections
- Retain the semantics of HTTP/1.1

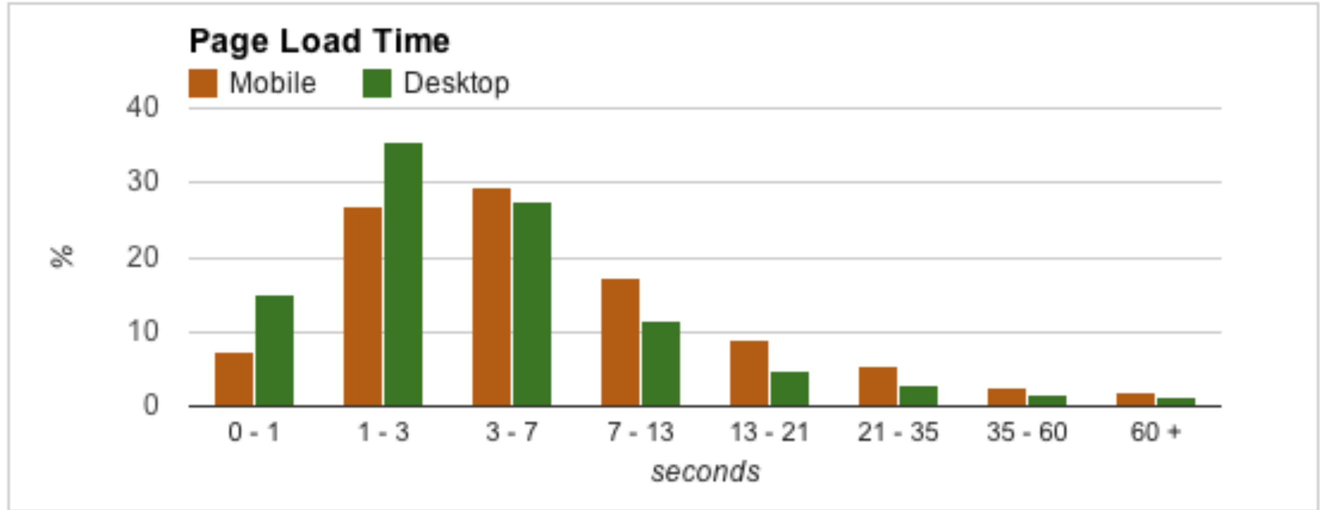
HTTP 2.0 goals

Usability Engineering 101

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

Usability Engineering - Jakob Nielsen, 1993





Desktop

Median: ~2.7s

Mean: ~6.9s

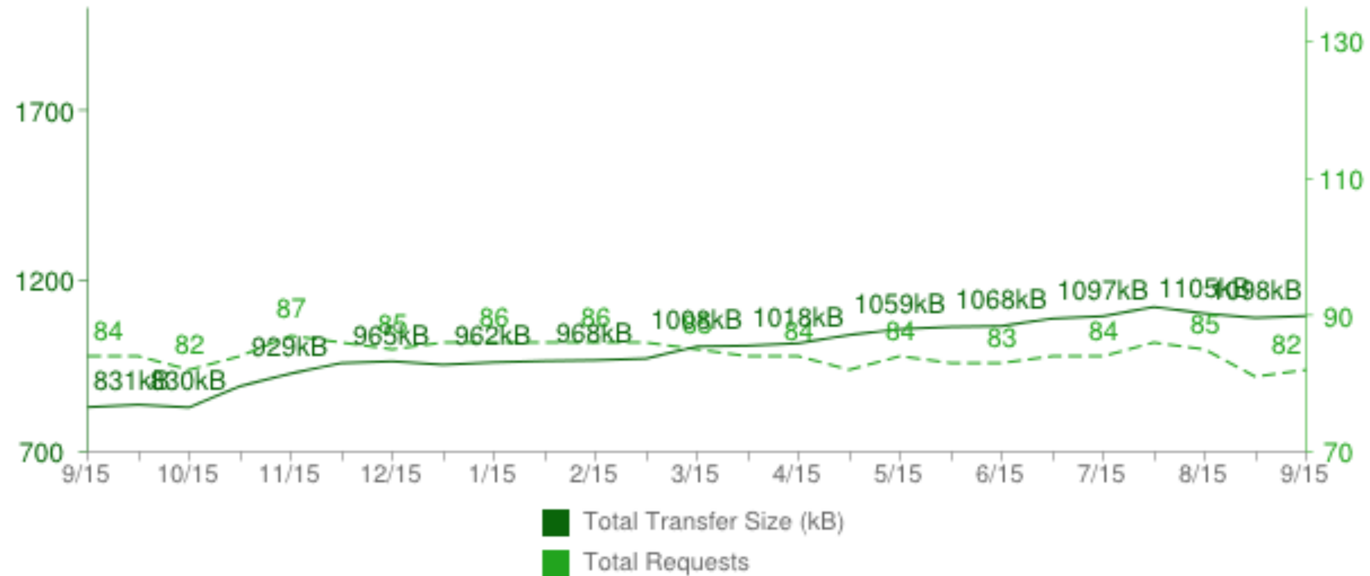
Mobile *

Median: ~4.8s

Mean: ~10.2s

** optimistic*

How Fast Are Websites Around The World? - Google Analytics Blog (April, 2012)

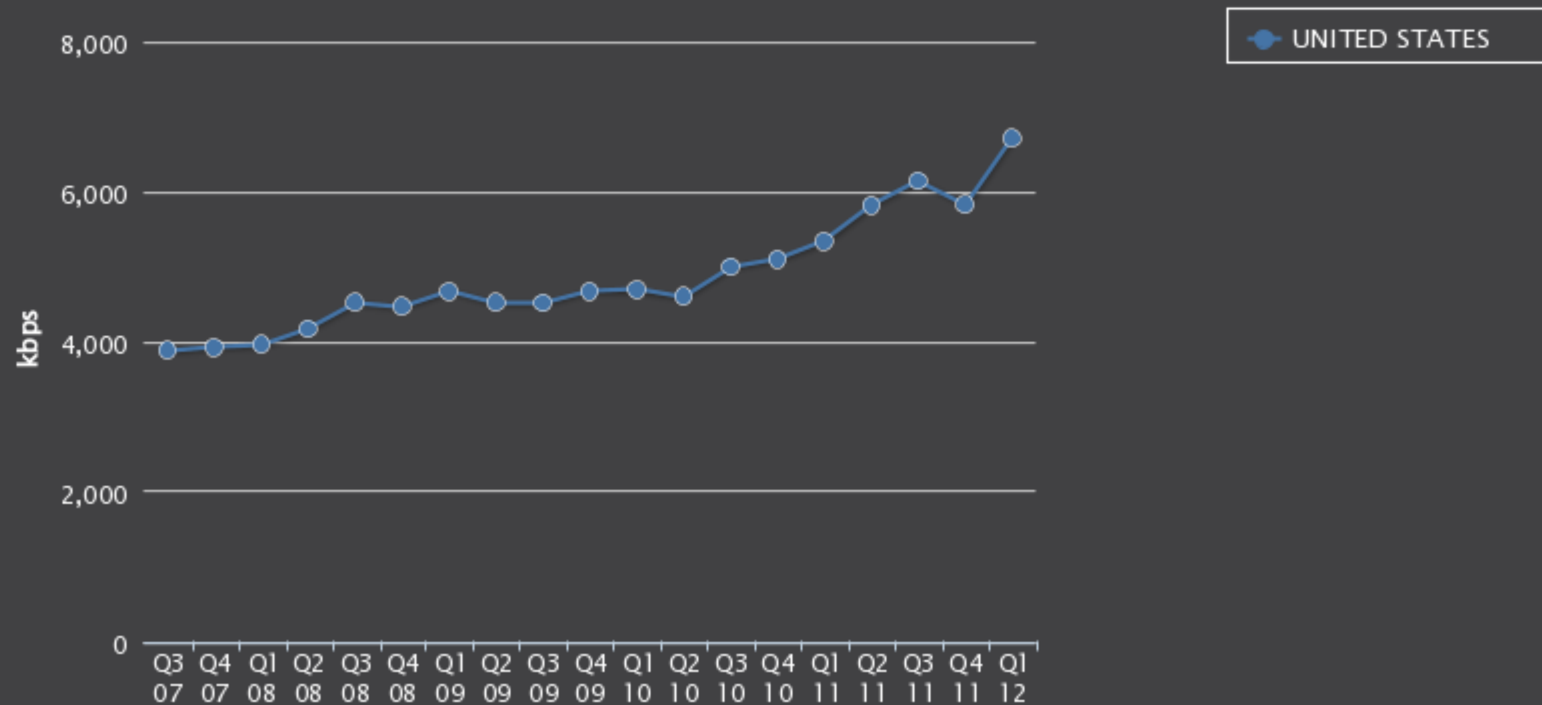


Content Type	Avg # of Requests	Avg size
HTML	8	44 kB
Images	53	635 kB
Javascript	14	189 kB
CSS	5	35 kB

HTTP Archive - Trends (Sept, 2012)

The network will save us?

Right, right? Or maybe not...



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

State of the Internet - Akamai - 2007-2012

Cable DSL Fiber

Average Latency (Milliseconds)

70
60
50
40
30
20
10
0

0.512 Mbps
1 Mbps
2 Mbps
4 Mbps
6 Mbps
8 Mbps
12 Mbps
16 Mbps
20 Mbps
24 Mbps
28 Mbps
35 Mbps
50 Mbps

Advertised Speed (Mbit/s)

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

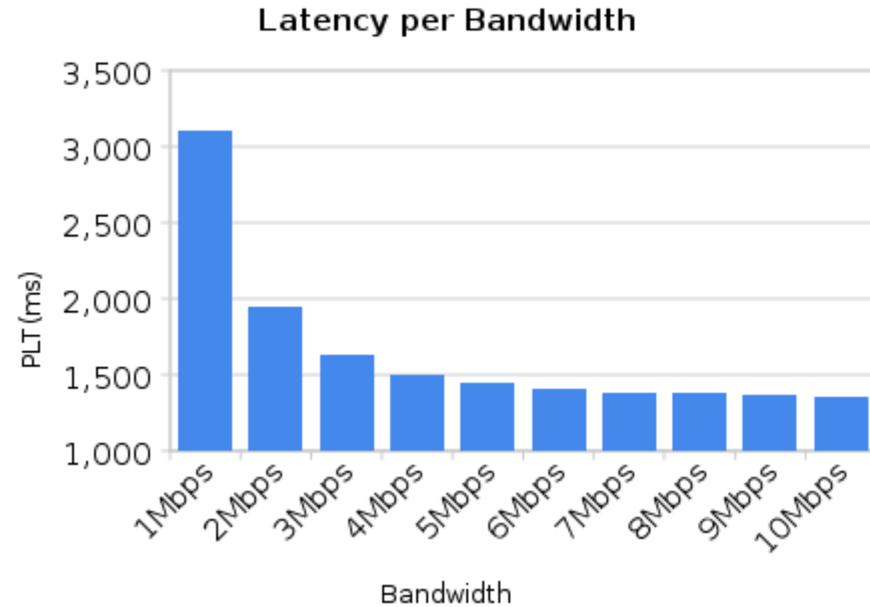
US: ~50~60ms

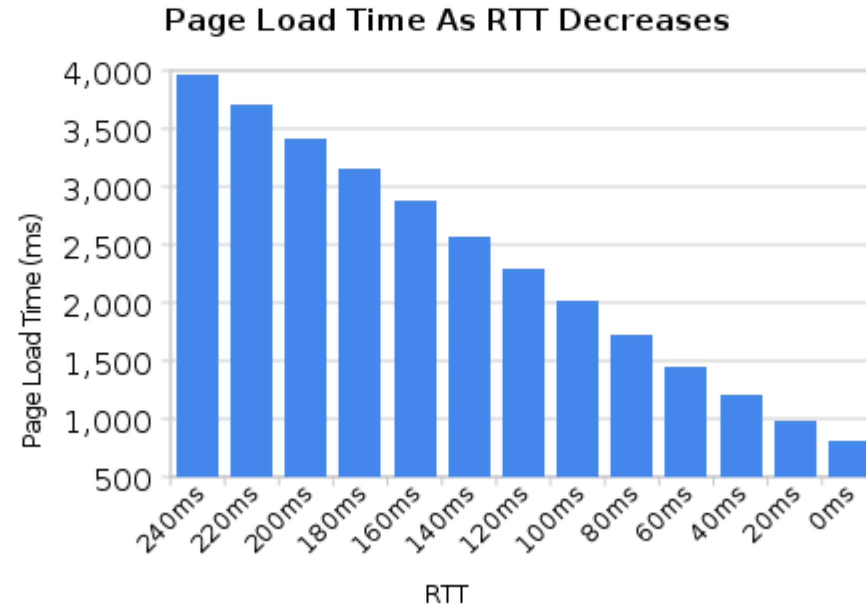
Average RTT to Google in 2012 is...

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.**

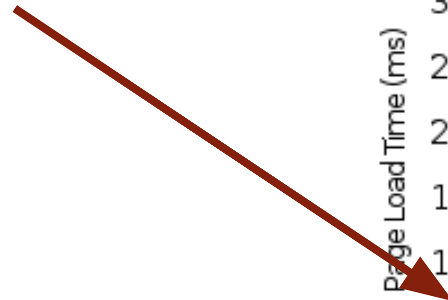
Bandwidth doesn't matter (much) - Google

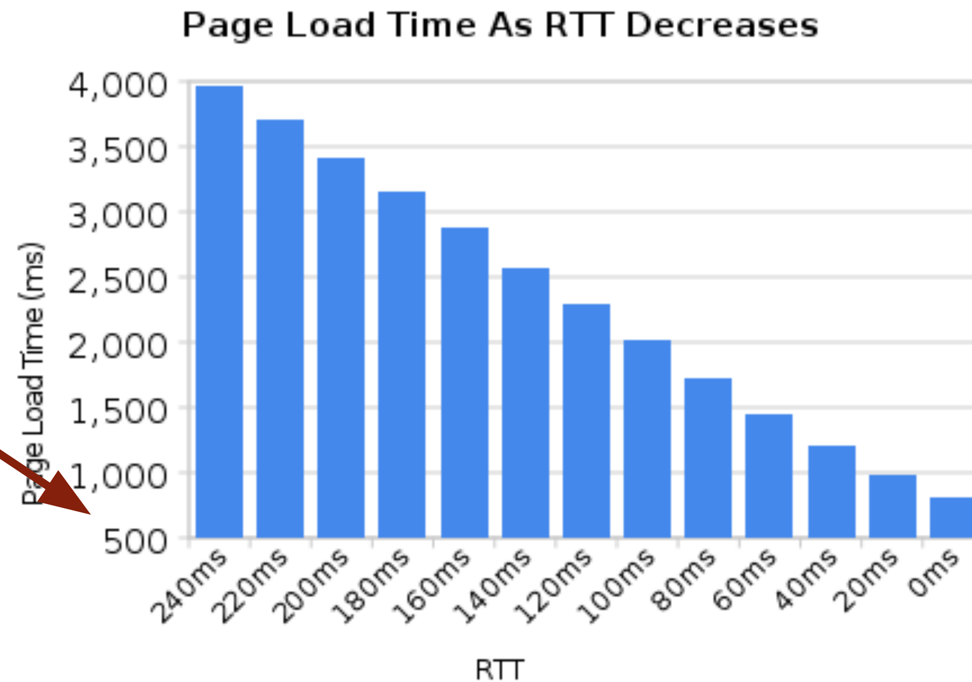
@igrigorik

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 240 ms!





[Virgin Mobile FAQ](#)

@igrigorik

- **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

[Latency is the new Performance Bottleneck](#)

@igrigorik

Why is latency the problem?

Remember that HTTP thing... yeah...

HTTP doesn't have multiplexing!

no pipelining

client server
open
close

time

pipelining

client server
open
close

client server
HOL

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

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

@igrigorik

Open multiple TCP connections!!!

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

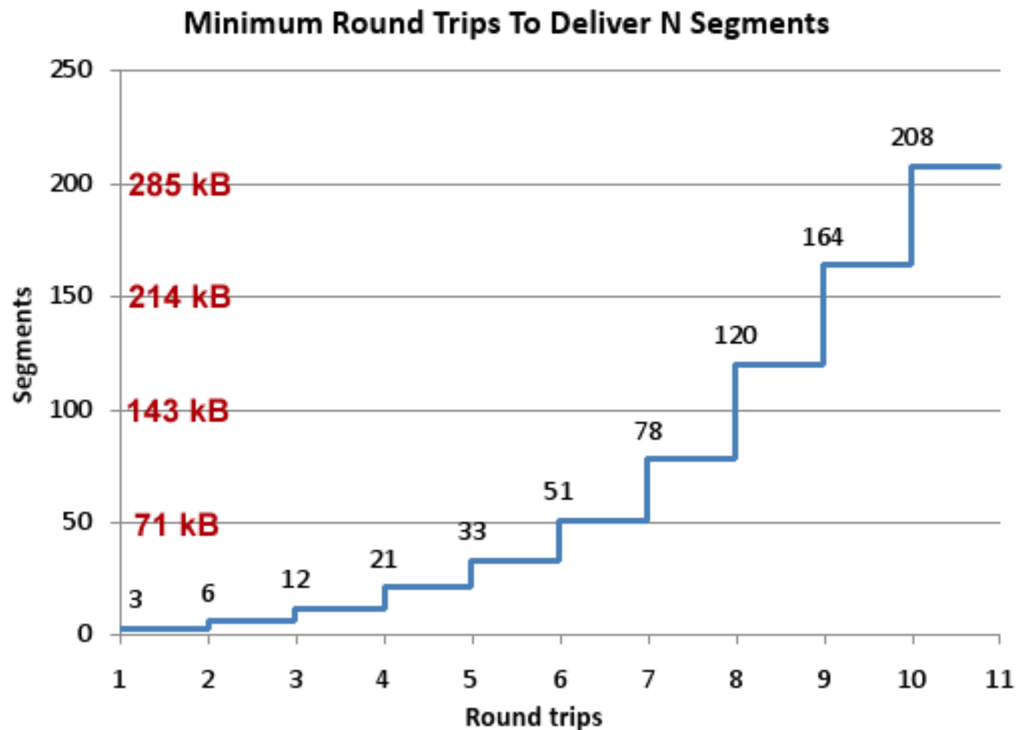
Top Mobile name	score	PerfTiming	Connections per Hostname
Android 2.3 →	8/16	no	9
Android 4 →	13/16	yes	6
Blackberry 7 →	11/16	no	5
Chrome Mobile 16 →	13/16	yes	6
IEMobile 9 →	11/16	yes	6
iPhone 4 →	10/16	no	4
iPhone 5 →	10/16	no	6
Nokia 950 →			
Opera Mobile 12 →	11/16	no	8

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

So what, what's the big deal?

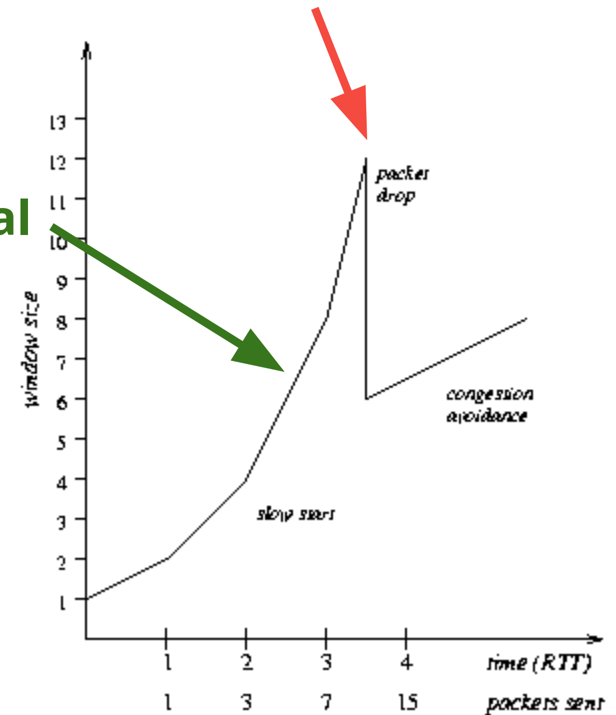
TCP Congestion Control & Avoidance...

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



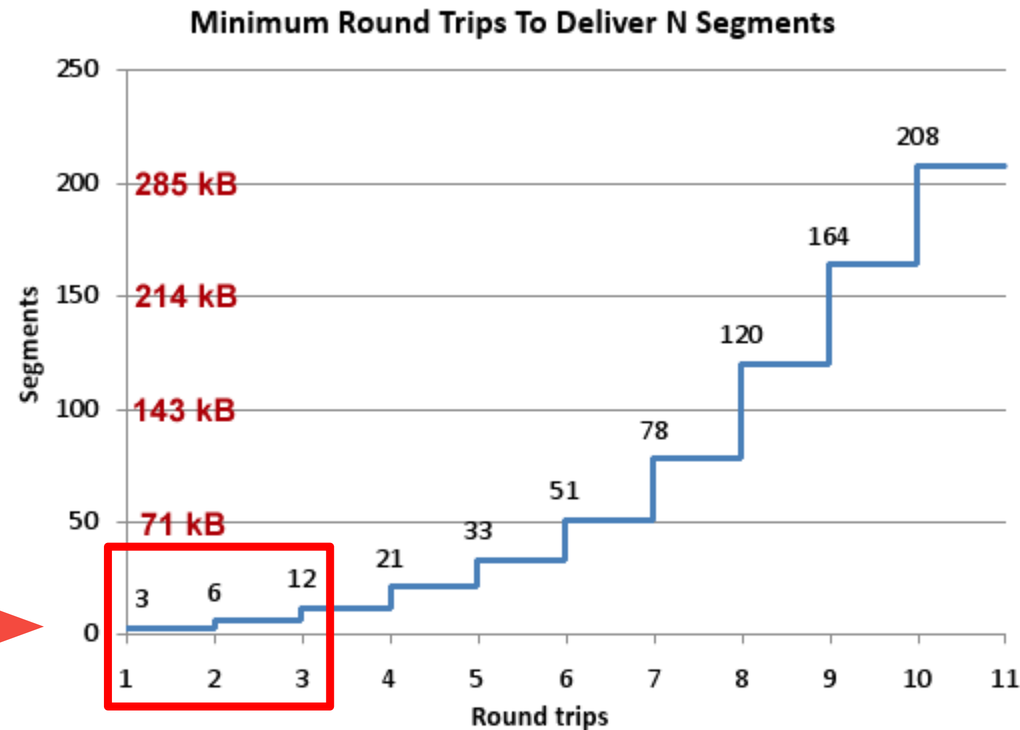
Packet Loss

Exponential growth



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 Dukkipati 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-

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

Default size on Linux 2.6.33+, which kernel are you running?

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

An Argument for Increasing TCP's initial Congestion window

@igrigorik

Let's talk about SPDY

err... HTTP 2.0!

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

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

HTTPbis charter

1. **Done** Call for Proposals for HTTP/2.0
2. **Done** 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

http://lists.w3.org/Archives/Public/ietf-http-wg/2012JulSep/0971.html

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/
- Address the "head of line blocking" problem in HTTP
- Not require multiple connections to a server to enable parallelism, thus improving its use of

Make things better

- 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)

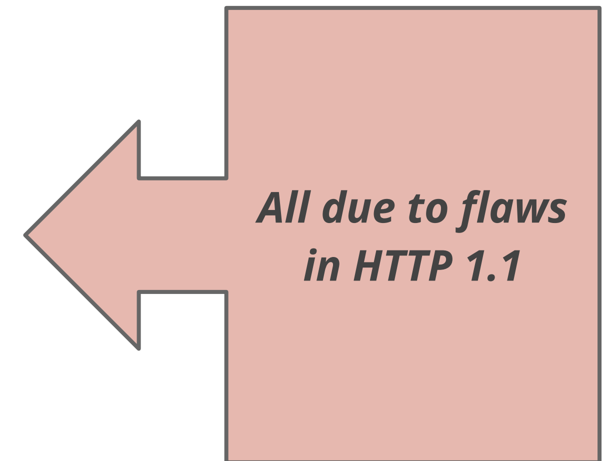
Build on HTTP 1.1

- Clearly identify any new extensibility points and policy for their appropriate use

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:
+----------------------------------+
|C| Version(15bits) | Type(16bits) |
+----------------------------------+
| Flags (8)  |  Length (24 bits)   |
+----------------------------------+
|               Data               |
+----------------------------------+

Data Frame:
+----------------------------------+
|D|       Stream-ID (31bits)       |
+----------------------------------+
| Flags (8)  |  Length (24 bits)   |
+----------------------------------+
|               Data               |
+----------------------------------+
```

Frame layout changed in SPDY v4 to be more consistent

SYN_STREAM

- Server stream ID: **even**
- Client stream ID: **odd**

- Associated-To: *push* *
- Priority: higher, better

- Length prefixed headers

SPDY v2

SYN_STREAM

Control

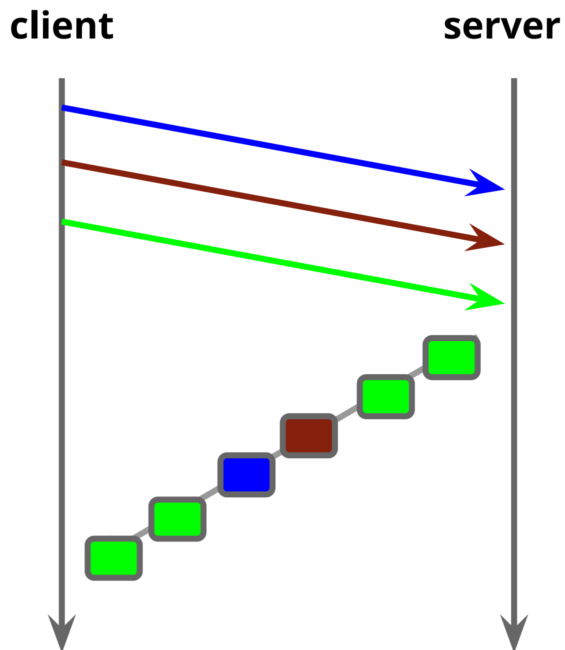
Request ID

Request Priority

```
+----------------------------------+
|1|       2          |       1      |
+----------------------------------+
| Flags (8)  |  Length (24 bits)    |
+----------------------------------+
|X|           Stream-ID (31bits)    |
+----------------------------------+
|X|Associated-To-Stream-ID (31bits)|
+----------------------------------+
| Pri | Unused    |                 |
+------------------                 |
|     Name/value header block       |
```

```
+------------------------------------+
| Number of Name/Value pairs (int16) |
+------------------------------------+
|     Length of name (int16)         |
+------------------------------------+
|           Name (string)            |
                 ...
```

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 (v2,v3)
  - new compressor in v4

- **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)

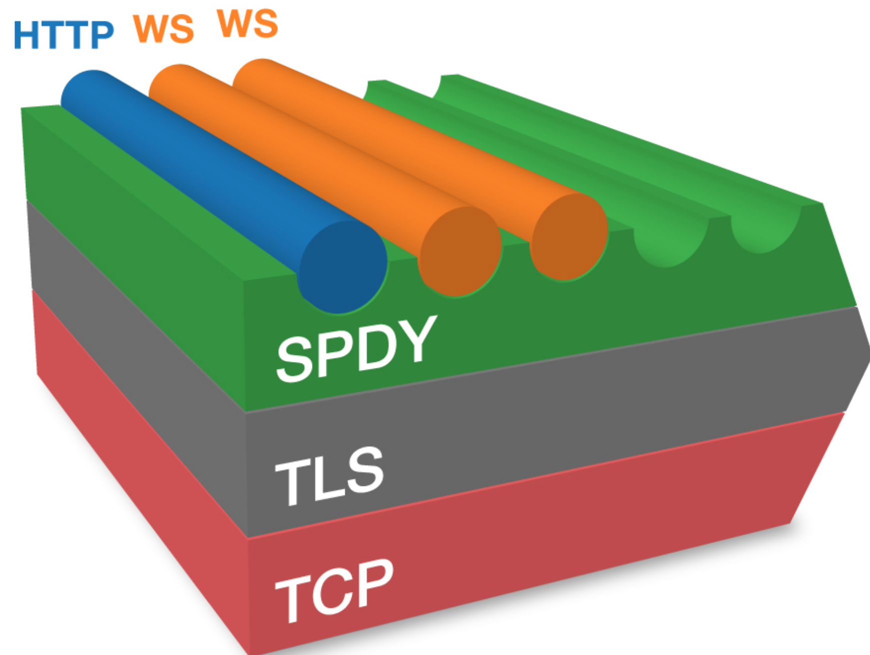
Newsflash: we are already using "server push"

- 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!**

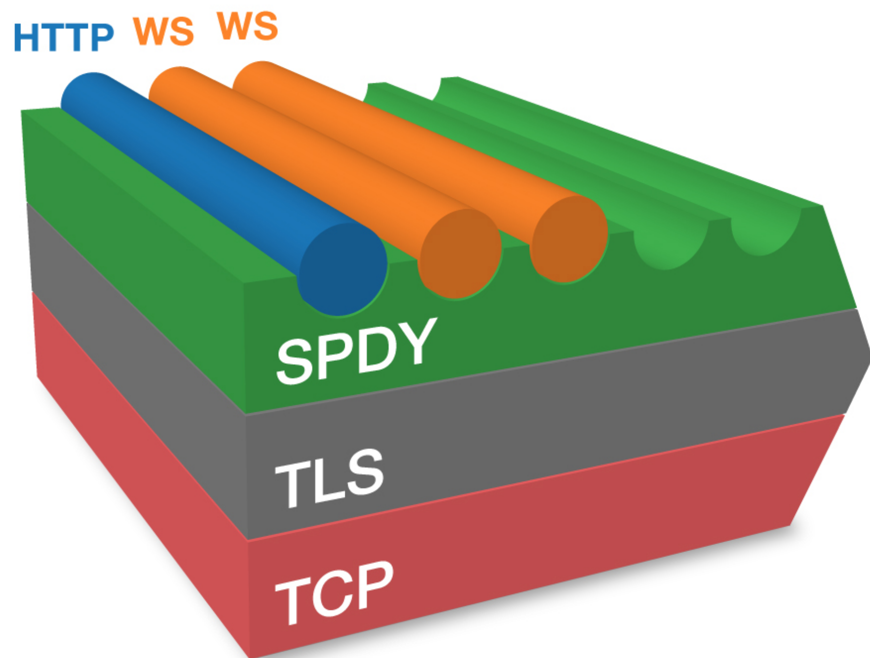
Observation: intermediate proxies get in the way

- 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."*

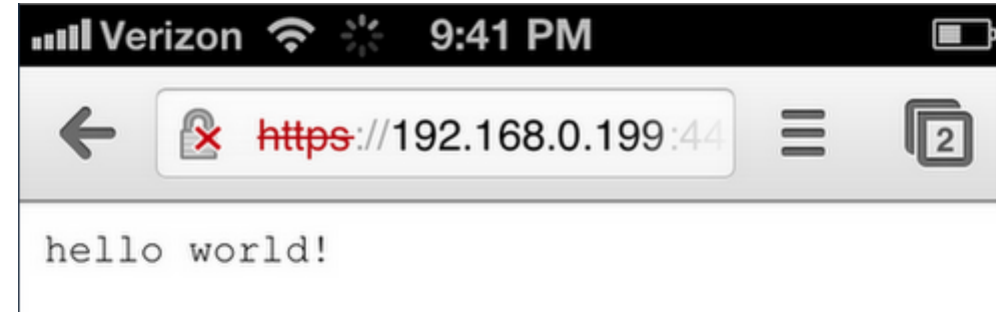
- Adam Langley (Google)

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, ...

SPDY v4 changes and improvements

- *Continuing to experiment and evolve the SPDY spec at Google, and feed our experience back into the HTTPbis discussions.*

- **ETA:** this month... or early next year.
- New **delta-encoding compressor** (ml)
  - Out with gzip. New compressor is faster, better.
  - Allows to create different stream groups.

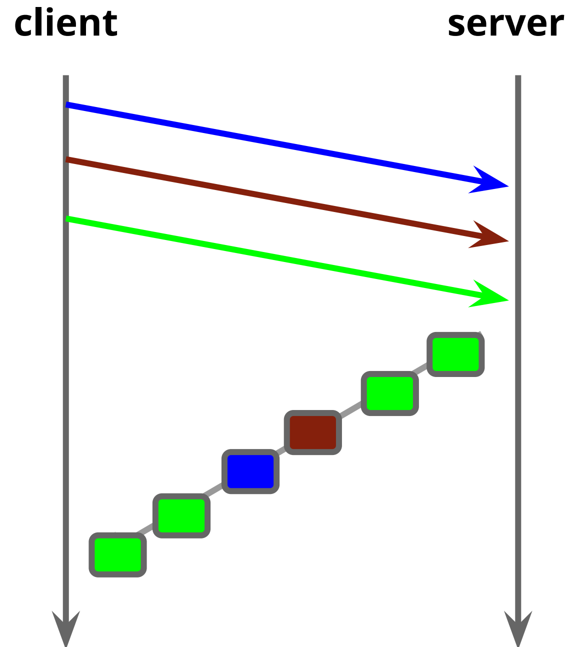
- **Stream dependencies** and priority graph (doc)
  - Out with 0-7 priority, in with stream dependencies
    - *aka, I want X before Y*

- Shuffle wire format to be more consistent
- More clarifications, small fixes, etc.

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, ...

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

http://blog.bubbleideas.com/2012/08/How-to-set-up-SPDY-on-nginx-for-your-rails-app-and-test-it.html

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**

https://github.com/indutny/node-spdy

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

http://www.smartjava.org/content/how-use-spdy-jetty

Am I SPDY?

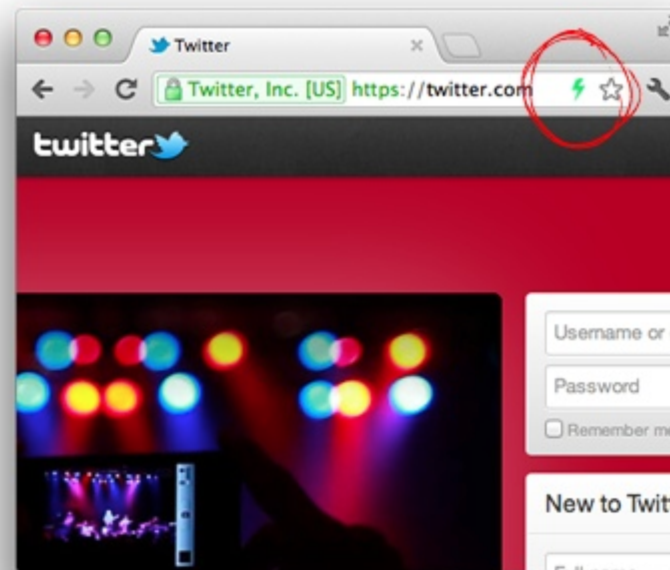
How do I know, how do I debug?

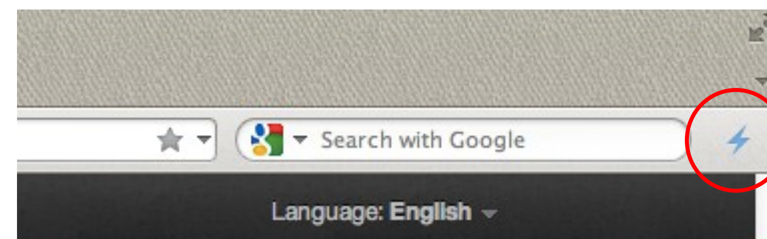
SPDY indicator(s)

- 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 Negotiatied	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

Building a Modern Web Stack

for the Real-time Web...

We're optimized for "Yesterday's Web"

We have an **infrastructure gap** between existing services and the real-time web

What we need:

- Request and Response **streaming should be the default**
- Connections to backend servers should be **persistent**
- Communication with backend servers should be **message-oriented**
- Communication between clients and backends should be **bi-directional**

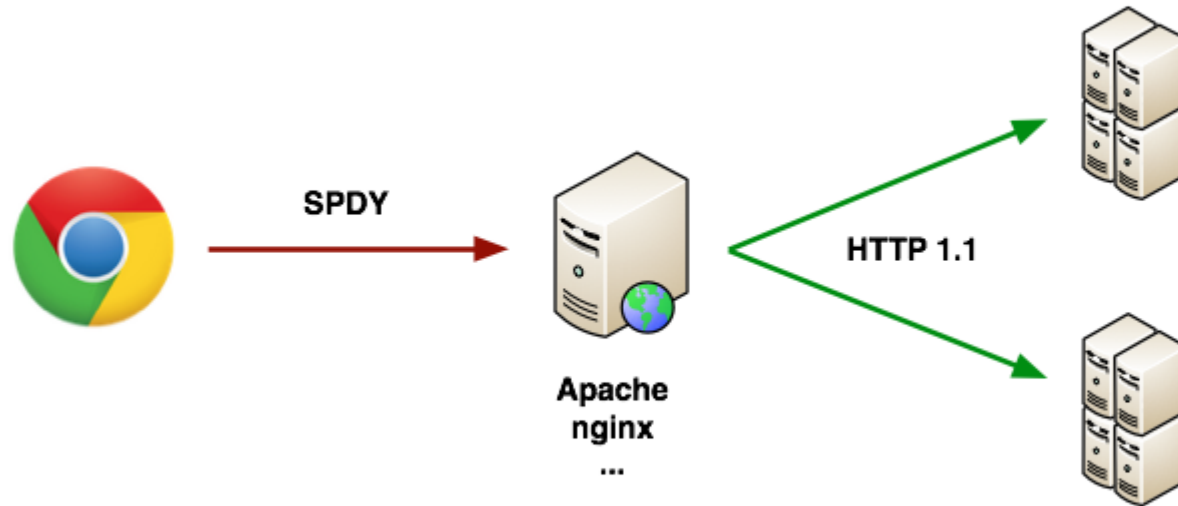
What we have:

- Occasional connection re-use... who uses pipelining in own infrastructure?
- Stream oriented protocols... how many times do you reparse those HTTP headers?
- Multiplexing? Not with HTTP!

... Writing HTTP parsers in 2012 is neither fun nor an interesting problem.

\- yours truly

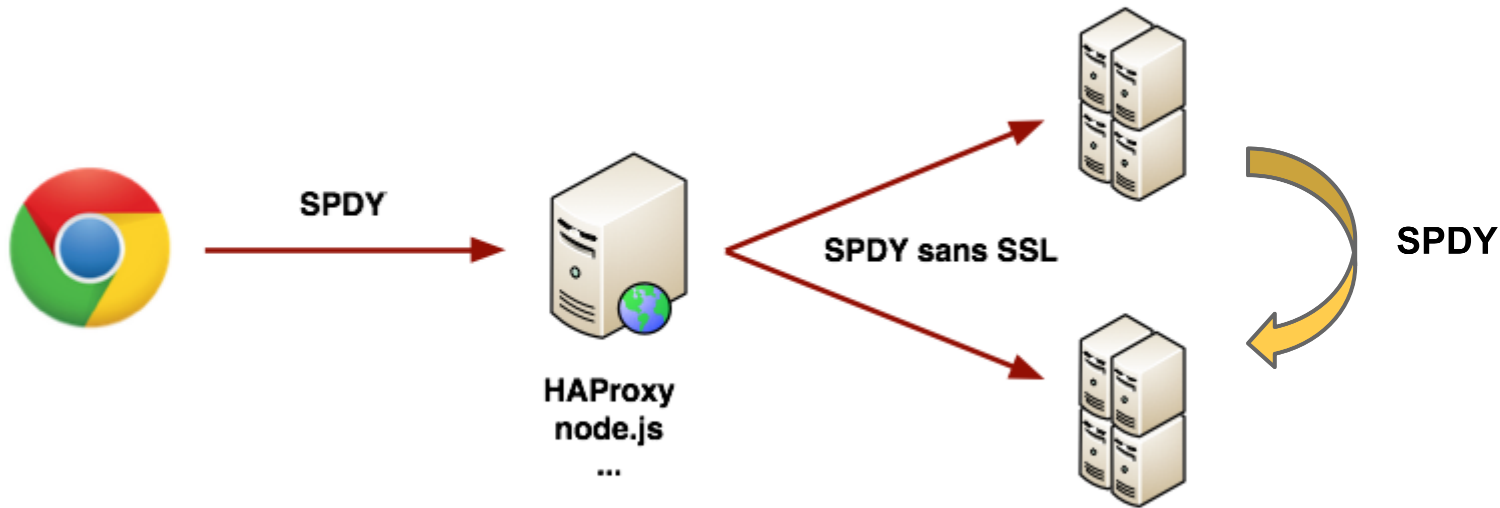
Please, let's not do this...



Yes, it's a convenient "upgrade" path. But, in the process we...

- **Give up multiplexing** on backend servers
- **Give up prioritization** on backend servers
- **Give up header compression** on backend servers
- Forced to (once again) to **reparse the TCP stream**

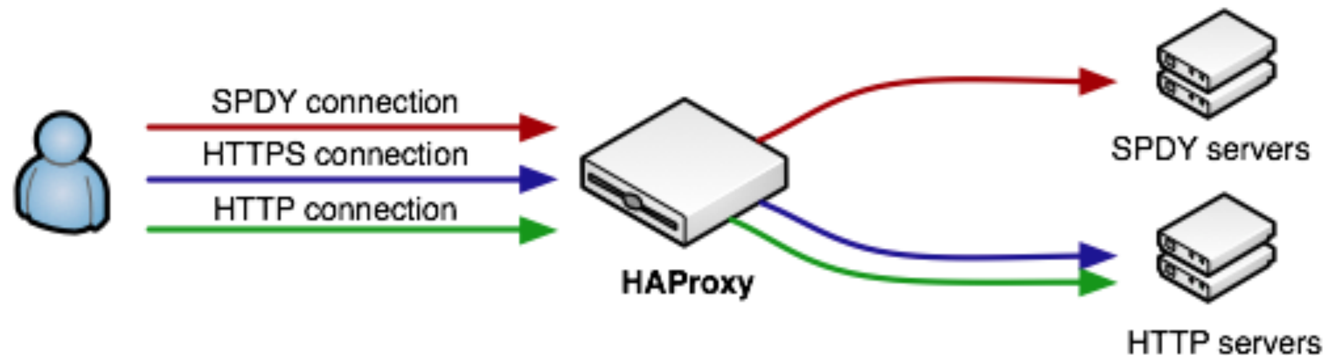
Talk HTTP 2.0, life will be better...



SSL is a deployment constraints on the "World Wild Web"

- **Terminate SSL+NPN**, forward raw SPDY frames
  - Leverage SPDY in own infrastructure!
- HTTP 2.0 **is your new, optimized, RPC layer**
  - Thrift, <insert own>, ..., RPC layer? Bah!

Hands on, with HAProxy




```
frontend secure
  mode tcp

  bind :443 ssl crt ./certs/ha.pem npn spdy/2
  use_backend spdy_cluster if { ssl_npn -i spdy/2 }

  default_backend http_cluster
```

- Terminate SSL, enable NPN
- Forward raw SPDY frames to SPDY servers
- Forward everyone else to HTTP servers

Also possible to do with nginx, spdylay, ...

[Simple SPDY and NPN Negotiation with HAProxy]

If I was to build a new backend system...

- I would **use HTTP 2.0** over any other RPC mechanism
- I would take advantage of **multiplexing, priorities, and server-push**
- I would have a **future-proof backend**

- I would probably use a **message-oriented transport** (ex, ZMQ)
  - I'm tired of building parsers, and it's a waste
  - Message == HTTP 2.0 frame == YES!

What we need:

- Back-end support for HTTP 2.0 / SPDY
  - *I shouldn't have to rewrite my code, or even adopt a new server..*
- (Sane) client implementations for SPDY

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

It will benefit your backend as much, if not more...

- Scratch the <roll own> RPC
- Use HTTP 2.0 / SPDY on the backend

Slides @ **bit.ly/http2-backend**

Thanks! Questions?

Ilya Grigorik - @igrigorik

igvita.com