SPDY, err... HTTP 2.0

what is it, how, why, and when?

Ilya Grigorik - @igrigorik, gplus.to/igrigorik
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
# Usability Engineering 101

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

**Desktop**
- Median: ~2.7s
- Mean: ~6.9s

**Mobile** *
- Median: ~4.8s
- Mean: ~10.2s

* optimistic
### Total Transfer Size & Total Requests

<table>
<thead>
<tr>
<th>Content Type</th>
<th>Avg # of Requests</th>
<th>Avg size</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML</td>
<td>8</td>
<td>44 kB</td>
</tr>
<tr>
<td>Images</td>
<td>53</td>
<td>635 kB</td>
</tr>
<tr>
<td>Javascript</td>
<td>14</td>
<td>189 kB</td>
</tr>
<tr>
<td>CSS</td>
<td>5</td>
<td>35 kB</td>
</tr>
</tbody>
</table>
The network will save us?
Right, right? Or maybe not...
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: \(~100\text{ms}\)
US: \(~50\text{~60ms}\)

Average RTT to Google in 2012 is...
Bandwidth doesn't matter *much*!

It's the latency, dammit!
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**
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...)
• **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*
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!!!

- **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

![Minimum Round Trips To Deliver N Segments](image1.jpg)

- Exponential growth

![Packet Loss](image2.jpg)

- Exponential growth
HTTP Archive says...

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

![Graph showing minimum round trips to deliver N segments.]

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 a critical parameter in determining the initial throughput of TCP. The default setting of the initial congestion window (CWND) on most Linux distributions is 3 segments - double check your setup! The initial congestion window is determined by the network link’s bandwidth-delay product (BDP), memory available for TCP’s retransmission buffer (rmbuf), and a globally determined multiplier (m) that is typically set to 2 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 determining the initial throughput of TCP.

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

Default size on Linux 2.6.33+ - double check yours!
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

[http://lists.w3.org/Archives/Public/ietf-http-wg/2012JulSep/0971.html](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/1.1 using TCP
- Address the "head of line blocking" problem in HTTP
- Not require multiple connections to a server to enable parallelism, thus improving its use of TCP

- 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

@igrigorik
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. ...

---

All due to flaws in HTTP 1.1

@igrigorik
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
**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  
- ...

@igrigorik
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, ...
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

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

```javascript
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);
```

Profit

https://github.com/indutny/node-spdy
Jetty + SPDY

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...
   ...
4. Profit

I <3 Java :-)

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:

```javascript
window.chrome.loadTimes()
```

```
commitLoadTime: 1350252136.934823
finishDocumentLoadTime: 1350252137.397209
finishLoadTime: 1350252137.529396
firstPaintAfterLoadTime: 1350252137.611959
firstPaintTime: 1350252137.520384

navigationType: "Other"
npnNegotiatedProtocol: "spdy/3"
requestTime: 0
startLoadTime: 1350252135.83449
wasAlternateProtocolAvailable: false
wasFetchedViaSpdy: true
wasNpnNegotiated: true

__proto__: Object
```
Try it @ [https://spdy.io/](https://spdy.io/) - open the link, then head to net-internals & click on stream-id

### SPAN 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](https://spdy.io/)

<table>
<thead>
<tr>
<th>Host</th>
<th>Proxy</th>
<th>ID</th>
<th>Protocol Negotiated</th>
<th>Active streams</th>
<th>Unclaimed pushed</th>
<th>Max</th>
<th>Initiated</th>
<th>Pushed</th>
<th>Pushed and claimed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.docs.google.com:443</td>
<td>direct://</td>
<td>305272</td>
<td>spdy/3</td>
<td>1</td>
<td>0</td>
<td>100</td>
<td>80</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>clients4.google.com:443</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>apis.google.com:443</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cbks0.google.com:443</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clients1.google.com:443</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clients2.google.com:443</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>docs.google.com:443</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>drive.google.com:443</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>encrypted-tbn0.gstatic.com:443</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>encrypted-tbn1.gstatic.com:443</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>encrypted-tbn2.gstatic.com:443</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>encrypted-tbn3.gstatic.com:443</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>khtml0.google.com:443</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>khtml1.google.com:443</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maps-apis-ssl.google.com:443</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>direct://</td>
<td>280013</td>
<td>spdy/3</td>
<td></td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>5471</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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?

Ilya Grigorik - @igrigorik
igvita.com

Slides @ bit.ly/http2-spdy