Making the Web Faster

at Google and beyond (with focus on beyond)

Ilya Grigorik - @igrigorik
Perf Engineer, Developer Advocate
Google

- Kernel, Networking, Infrastructure, Chrome, Mobile...
- Research & drive performance web standards (W3C, etc)
- Build open source tools, contribute to existing projects
- Optimize Google, optimize the web... *

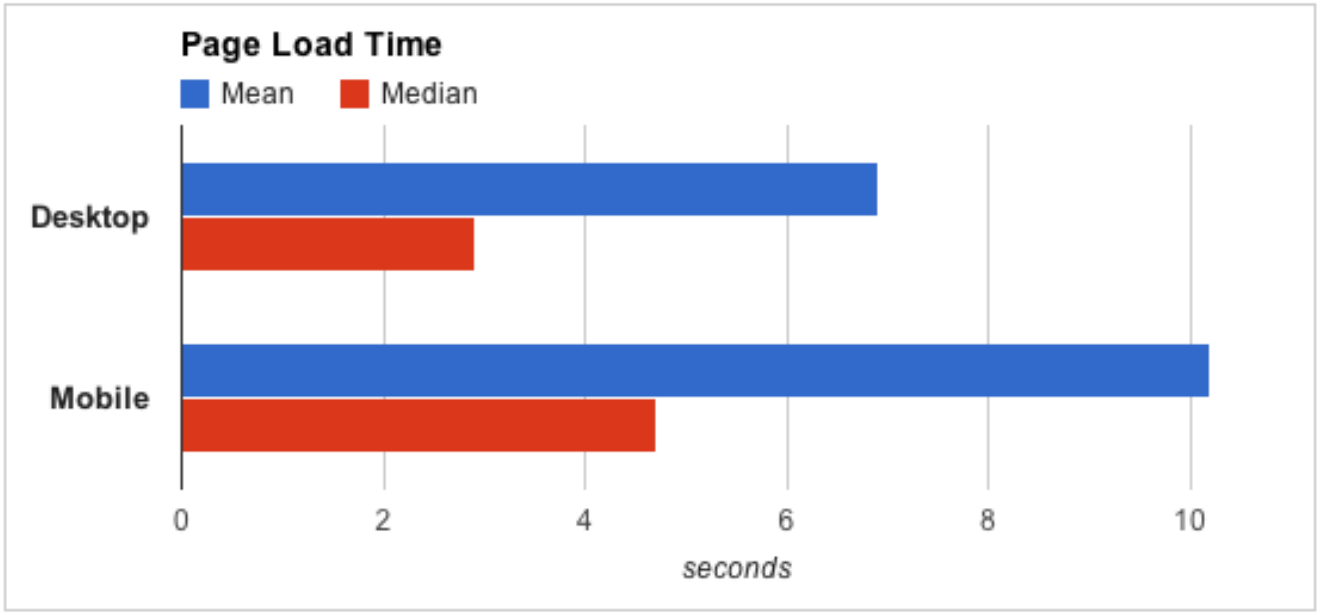
** same thing..*

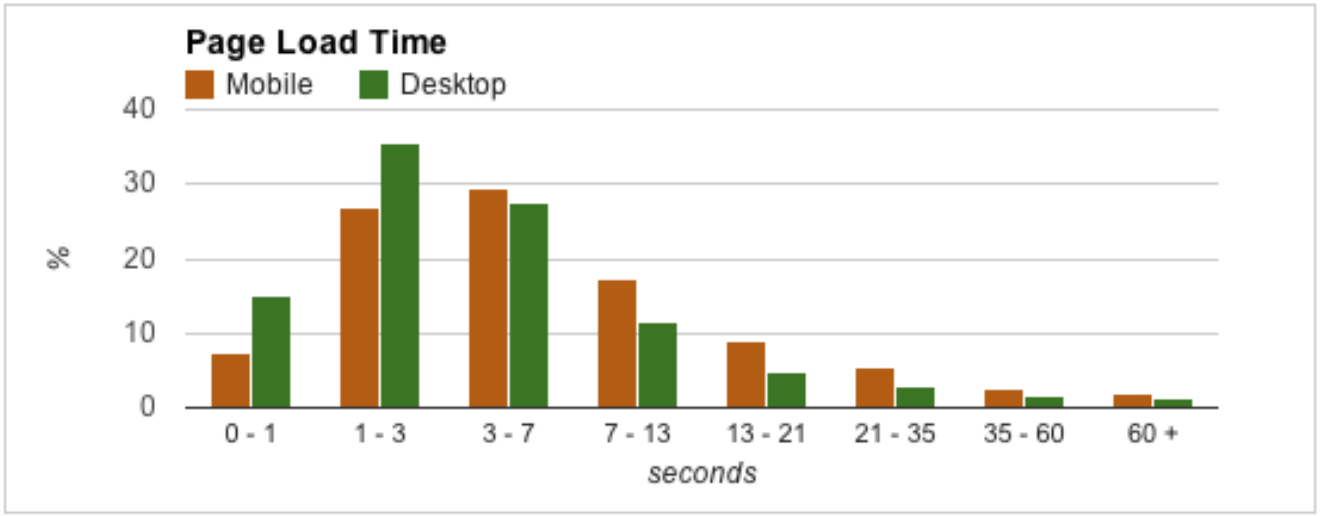
Goal: make the entire web *faster*

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

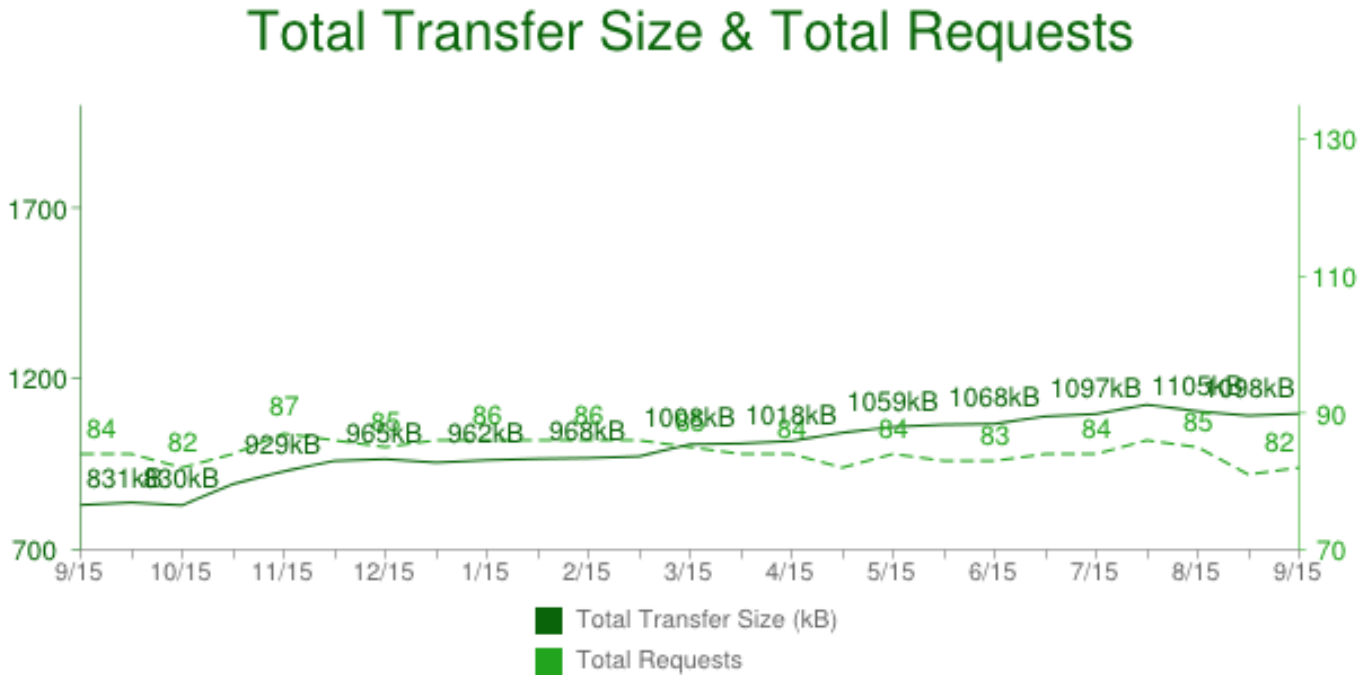
Mobile *

Median: ~4.8s

Mean: ~10.2s

* *optimistic*

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

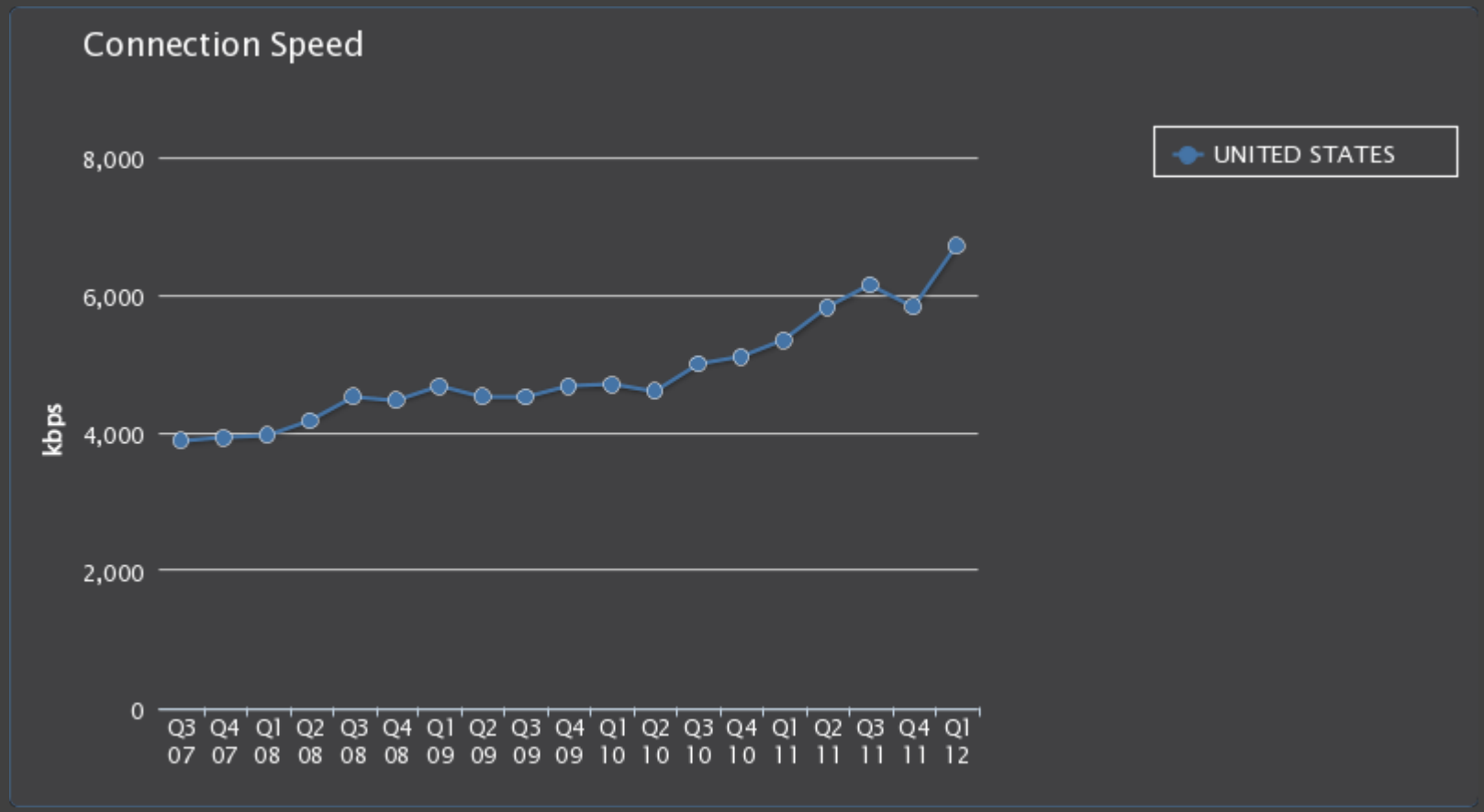


Content Type	Avg # of Requests	Avg size
HTML	8	44 kB
Images	53	635 kB
Javascript	14	189 kB
CSS	5	35 kB

[HTTP Archive - Trends](#) (Sept, 2012)

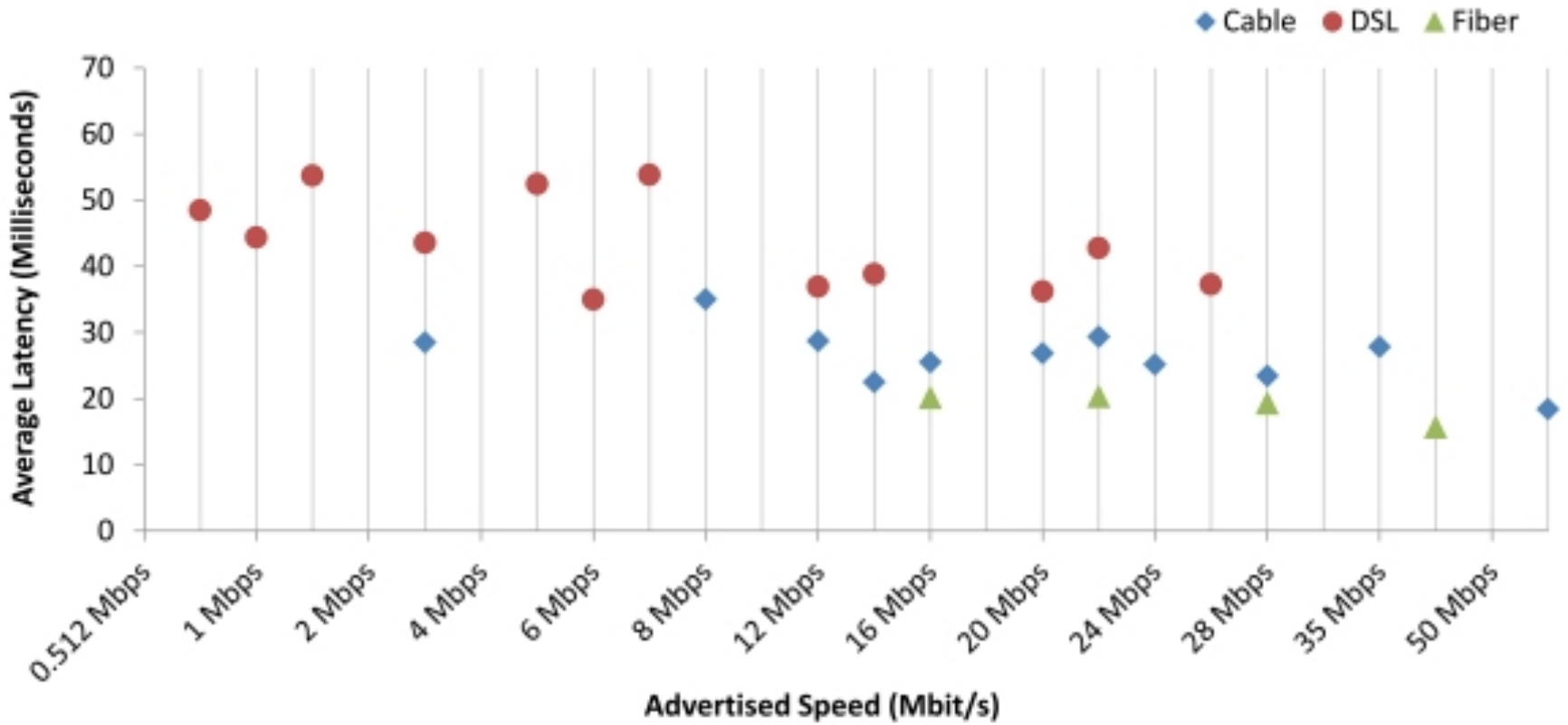
The network will save us?

Right, right? Or maybe not...



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

[State of the Internet](#) - Akamai - 2007-2012



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.*

Measuring Broadband America - July 2012 - FCC

Worldwide: ~100ms

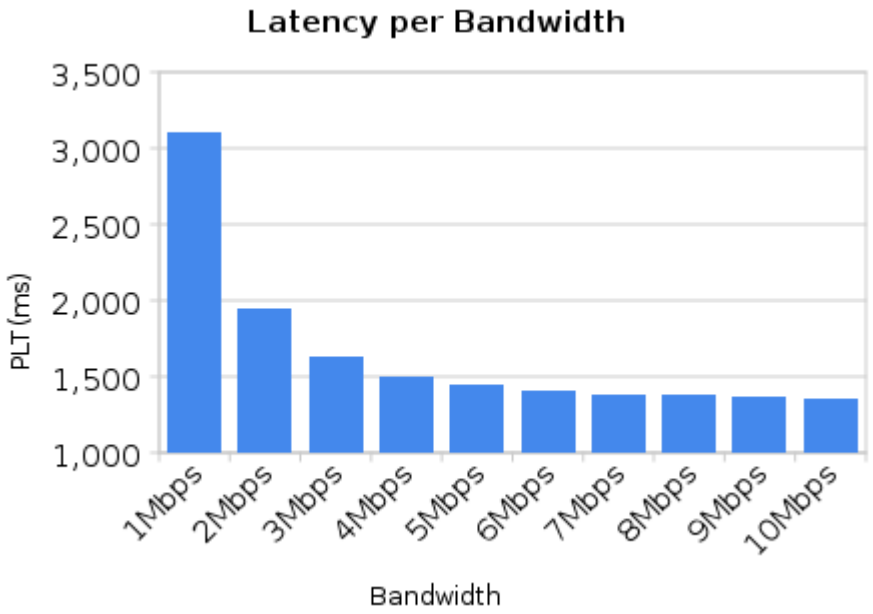
US: ~50~60ms

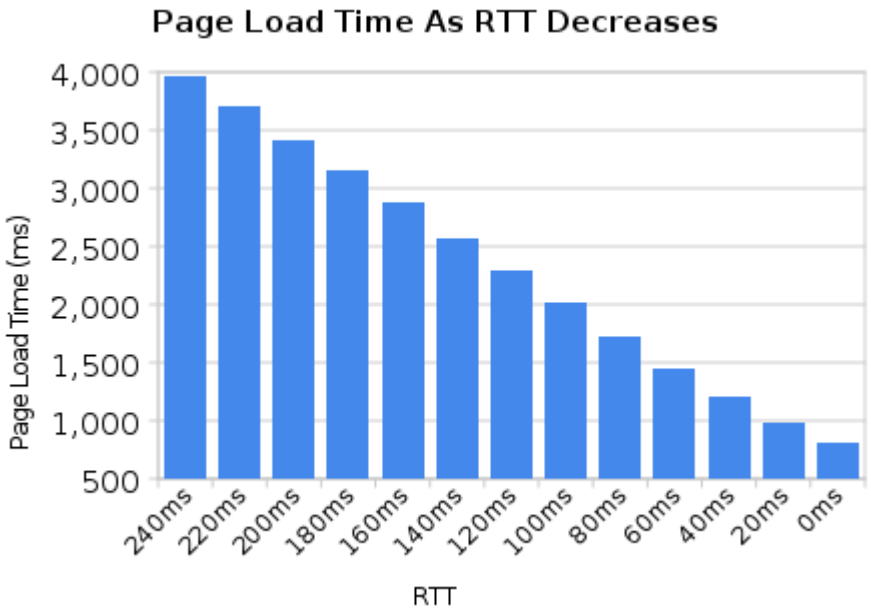
Average RTT to Google in 2012 is...

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.**

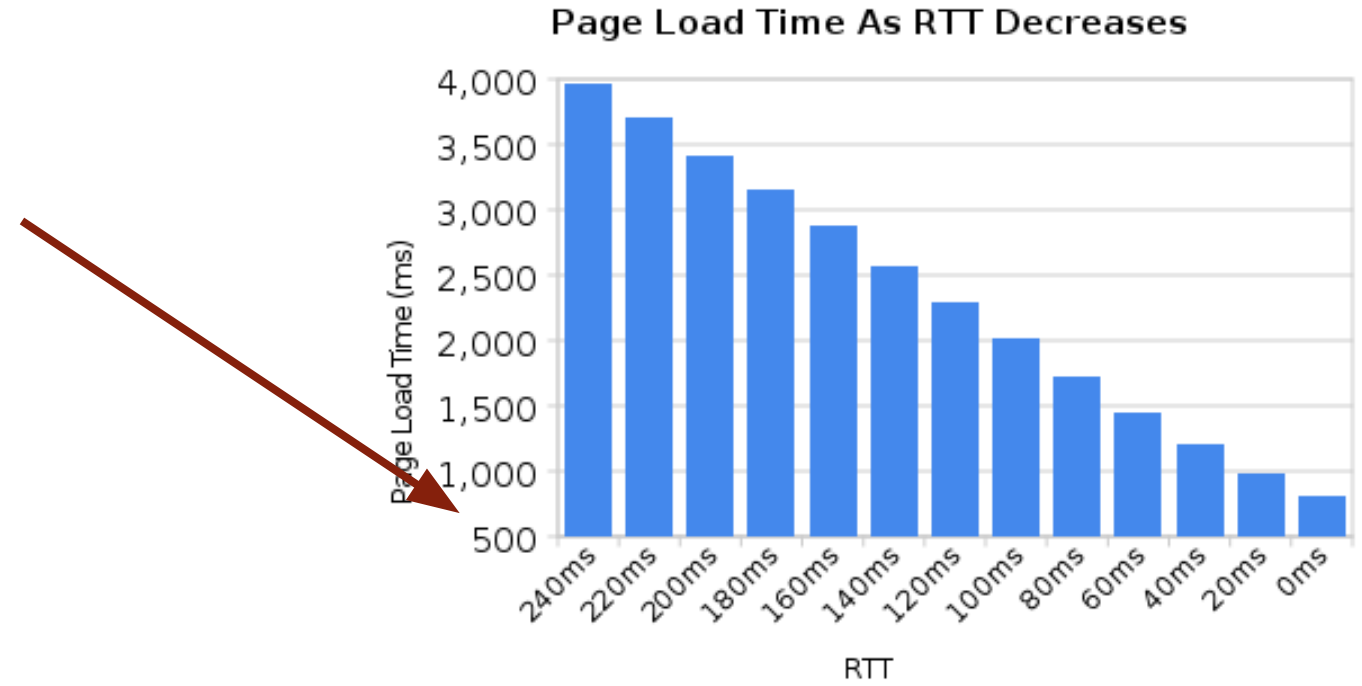
[Bandwidth doesn't matter (much)](#) - Google

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**.*

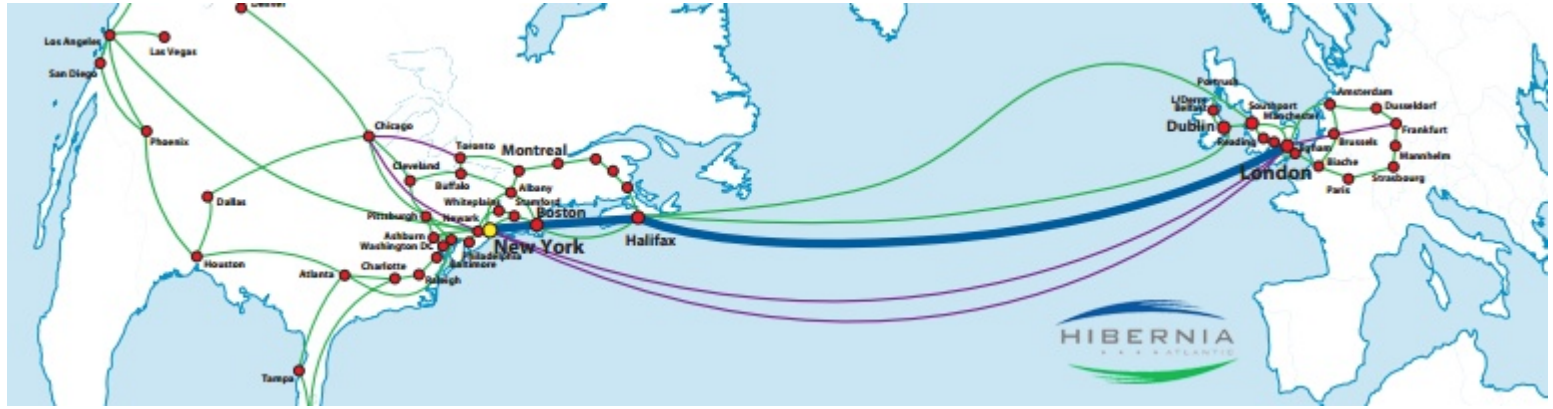
We stopped at 240ms!

(facepalm meme goes here...)



Verizon FAQ

- **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!

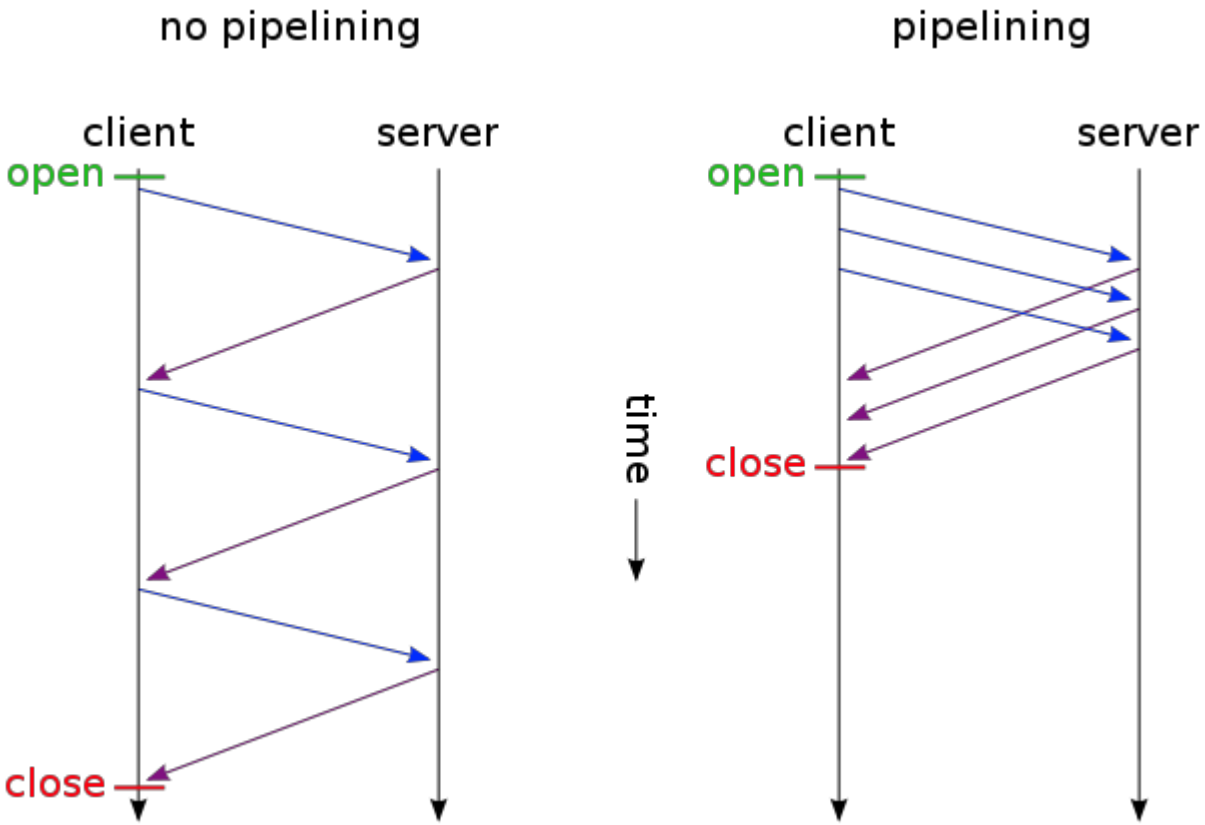


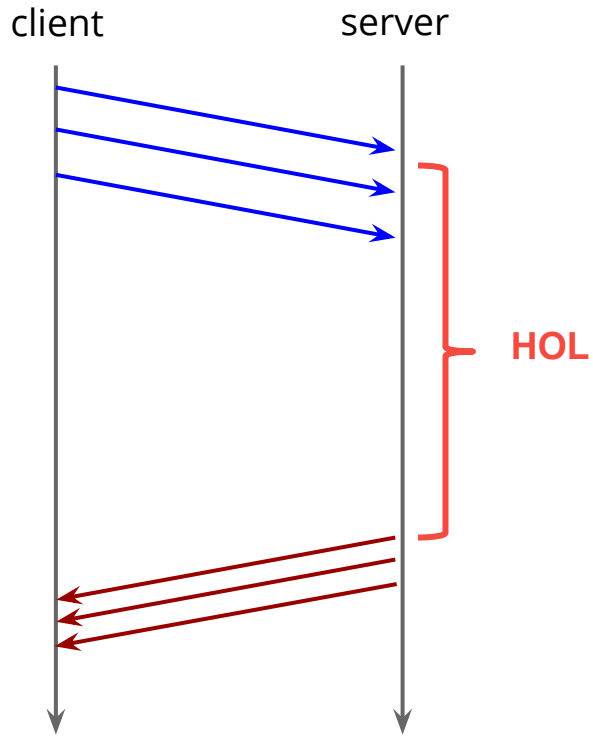
$80M / ms

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
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

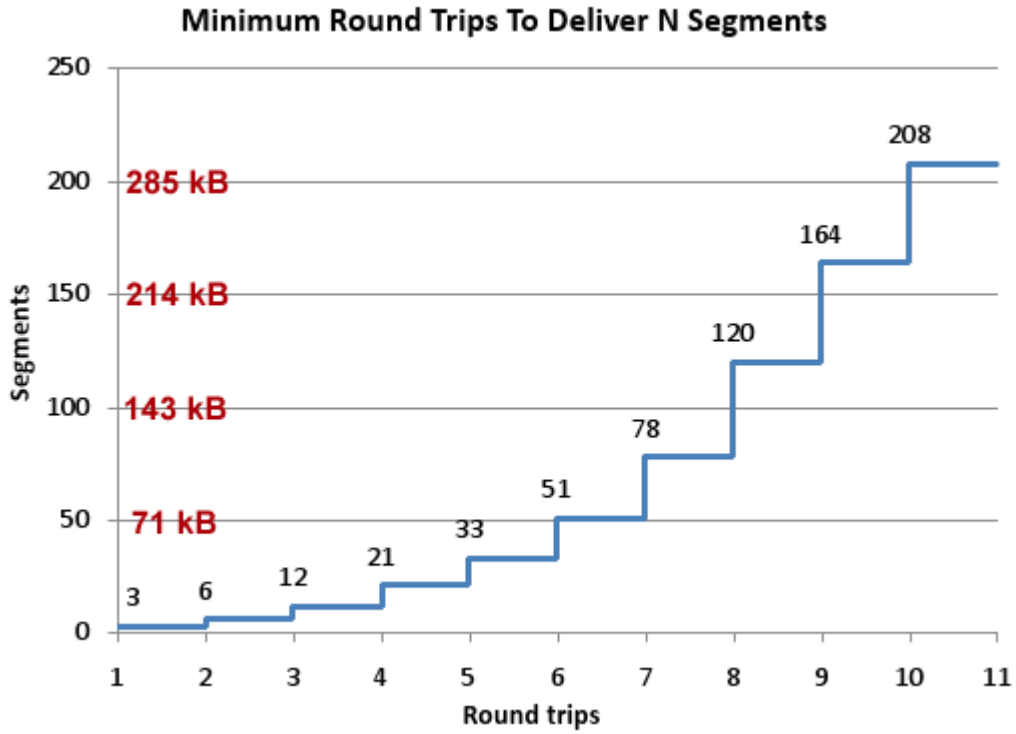
Top Mobile name	score	PerfTiming	Connections per Hostname
Android 2.3 →	8/16	no	9
Android 4 →	13/16	yes	6
Blackberry 7 →	11/16	no	5
Chrome Mobile 16 →	13/16	yes	6
IEMobile 9 →	11/16	yes	6
iPhone 4 →	10/16	no	4
iPhone 5 →	10/16	no	6
Nokia 950 →			
Opera Mobile 12 →	11/16	no	8

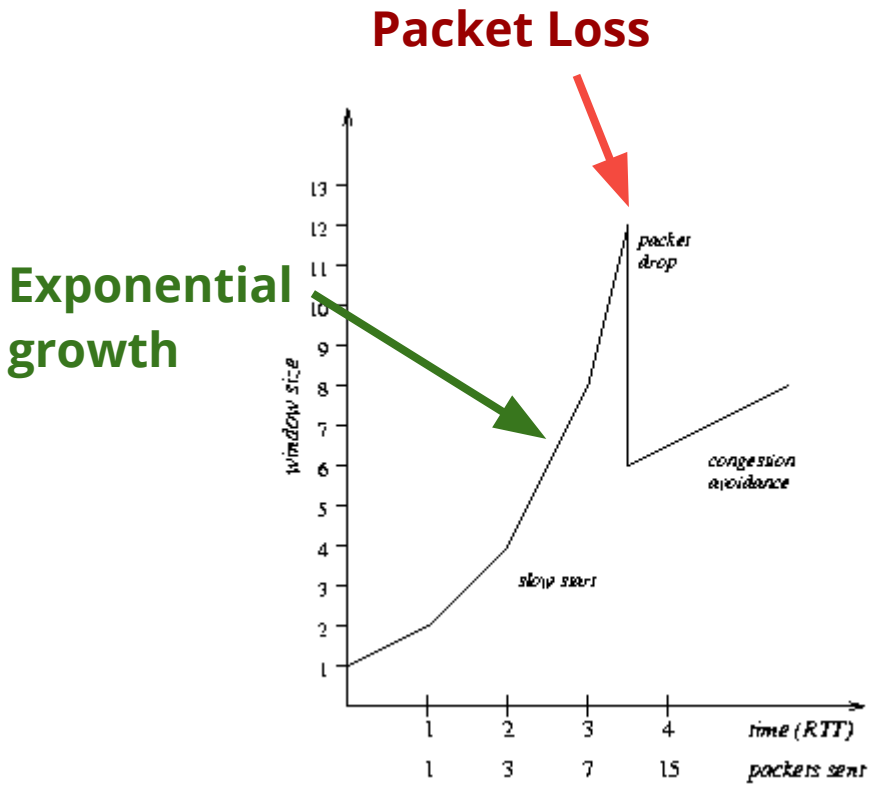
- **6 connections per host** on Desktop
- **6 connections per host** on Mobile (recent builds)

So what, what's the big deal?

TCP Congestion Control & Avoidance...

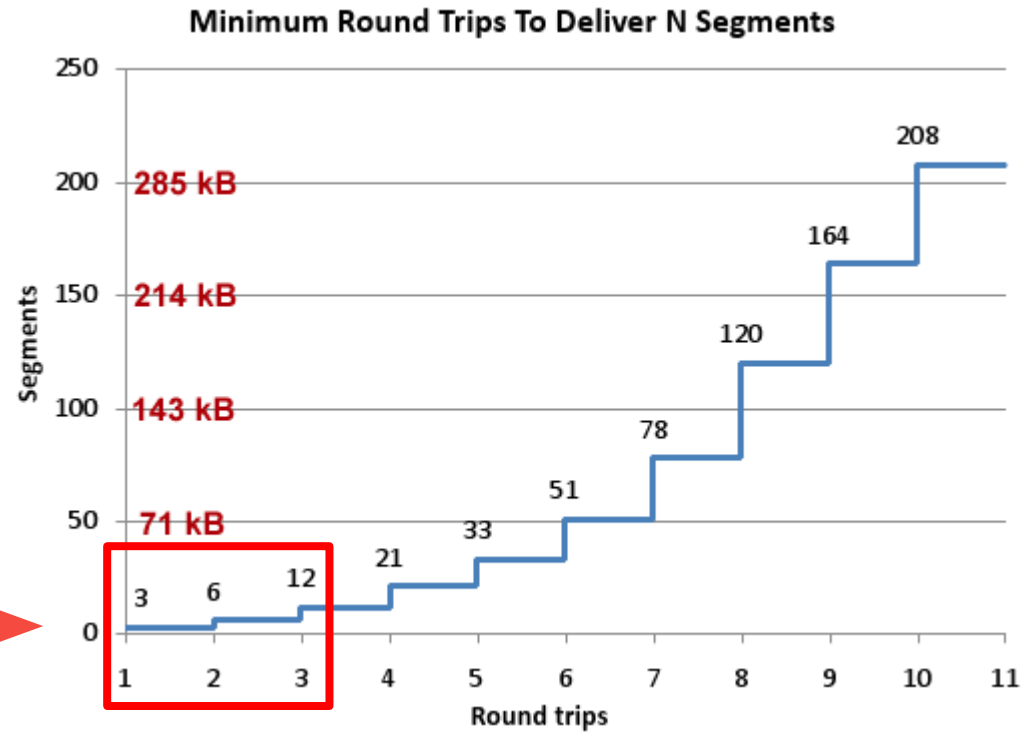
- TCP is designed to probe the network to figure out the available capacity
- **TCP Slow Start** - feature, not a bug





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 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

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-
nitial
short
TCP's
per a
bally
width
Kbps)
appli-
oad of
Web pages. Popular Web browsers, including IE8 [2], Fire-

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

Default size on Linux 2.6.33+ - double check yours!

Let's talk about DNS

A quick, but important detour...

Most DNS servers are...

- Under provisioned
- Not monitored well
- Susceptible to attacks
- ...

- Poor cache hit rate
- Intermittent failures
- DDOS, cache poisoning, ...

*"Operating the Googlebot web crawler, we have observed an **average resolution time of 130 ms** for nameservers that respond. However, a **full 4-6% of requests simply time out**, due to UDP packet loss and servers being unreachable. If we take into account failures such as packet loss, dead nameservers, DNS configuration errors, etc., the **actual average end-to-end resolution time is 300-400 ms**."*

Public DNS: Performance Benefits

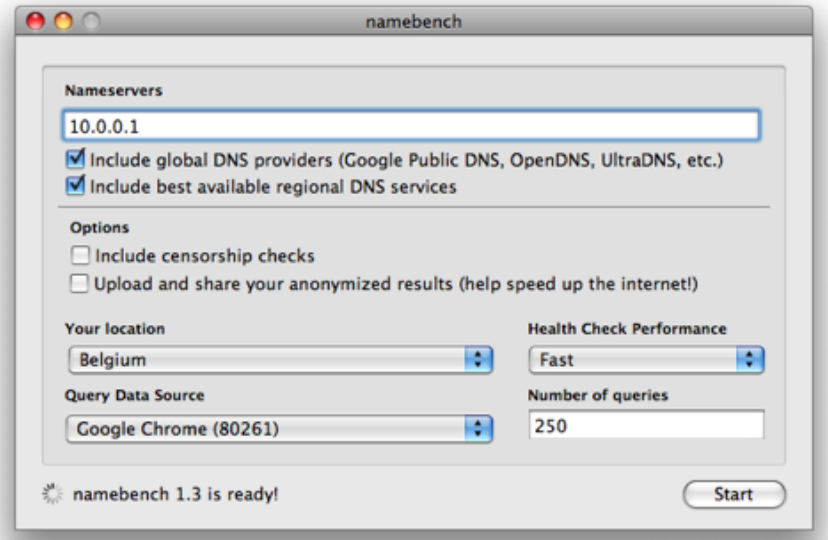
8.8.4.4

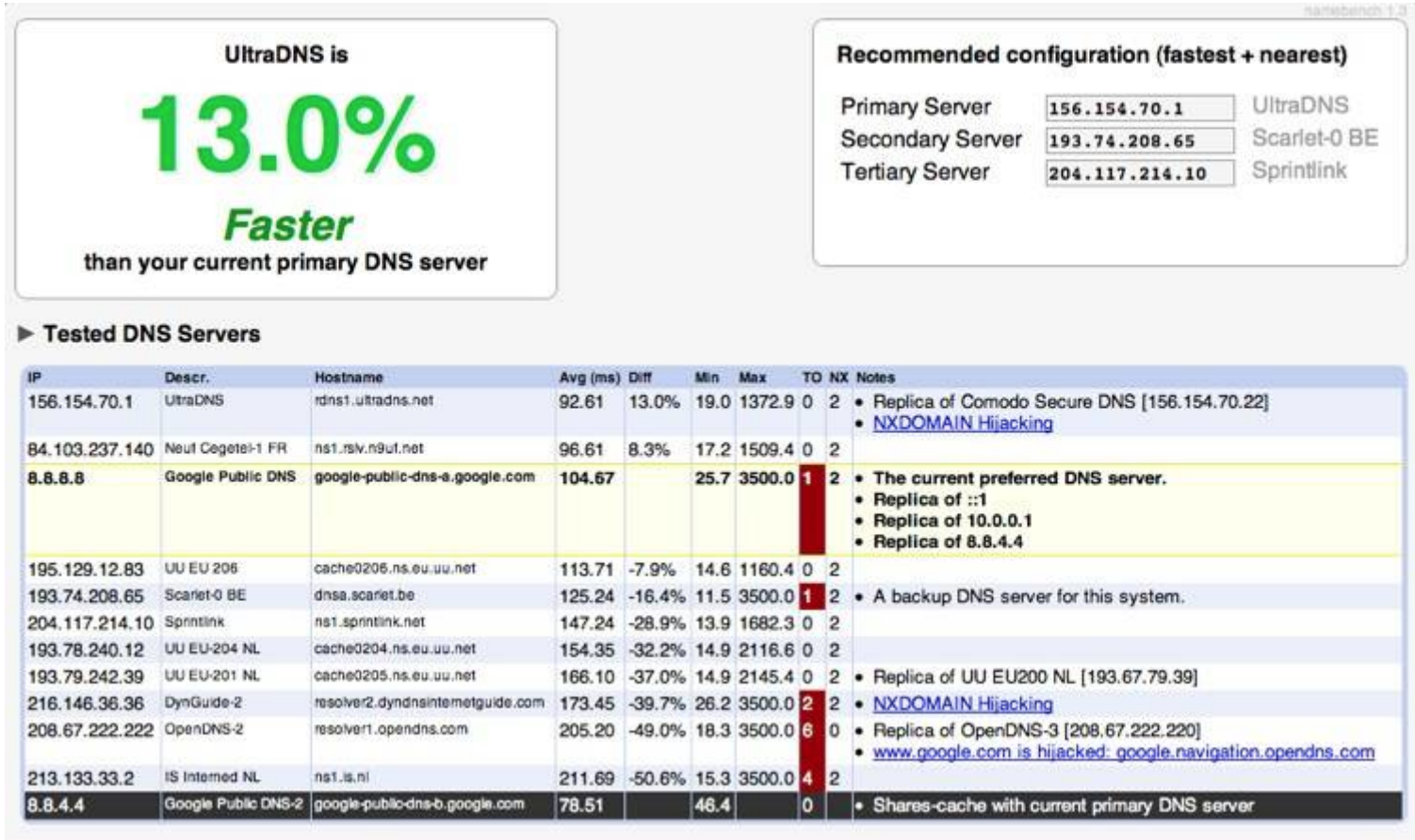
8.8.8.8

Google Public DNS

free, no redirects, etc.

namebench





IP	Descr.	Hostname	Avg (ms)	Diff	Min	Max	TO	NX	Notes
156.154.70.1	UltraDNS	rdns1.ultradns.net	92.61	13.0%	19.0	1372.9	0	2	• Replica of Comodo Secure DNS [156.154.70.22] • NXDOMAIN Hijacking
84.103.237.140	Neuf Cegetel-1 FR	ns1.rslv.n9uf.net	96.61	8.3%	17.2	1509.4	0	2	
8.8.8.8	**Google Public DNS**	**google-public-dns-a.google.com**	**104.67**		**25.7**	**3500.0**	**1**	**2**	• **The current preferred DNS server.** • **Replica of ::1** • **Replica of 10.0.0.1** • **Replica of 8.8.4.4**
195.129.12.83	UU EU 206	cache0206.ns.eu.uu.net	113.71	-7.9%	14.6	1160.4	0	2	
193.74.208.65	Scarlet-0 BE	dnsa.scarlet.be	125.24	-16.4%	11.5	3500.0	1	2	• A backup DNS server for this system.
204.117.214.10	Sprintlink	ns1.sprintlink.net	147.24	-28.9%	13.9	1682.3	0	2	
193.78.240.12	UU EU-204 NL	cache0204.ns.eu.uu.net	154.35	-32.2%	14.9	2116.6	0	2	
193.79.242.39	UU EU-201 NL	cache0205.ns.eu.uu.net	166.10	-37.0%	14.9	2145.4	0	2	• Replica of UU EU200 NL [193.67.79.39]
216.146.36.36	DynGuide-2	resolver2.dyndnsinternetguide.com	173.45	-39.7%	26.2	3500.0	2	2	• NXDOMAIN Hijacking
208.67.222.222	OpenDNS-2	resolver1.opendns.com	205.20	-49.0%	18.3	3500.0	6	0	• Replica of OpenDNS-3 [208.67.222.220] • www.google.com is hijacked: google.navigation.opendns.com
213.133.33.2	IS Internet NL	ns1.is.nl	211.69	-50.6%	15.3	3500.0	4	2	
8.8.4.4	Google Public DNS-2	google-public-dns-b.google.com	78.51		46.4		0		• Shares-cache with current primary DNS server

*"namebench runs a fair and **thorough benchmark using your web browser history, tcpdump output, or standardized datasets** in order to provide an **individualized recommendation**. namebench is completely free and does not modify your system in any way. This project began as a 20% project at Google."*

namebench - Google Code

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

*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 o

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)

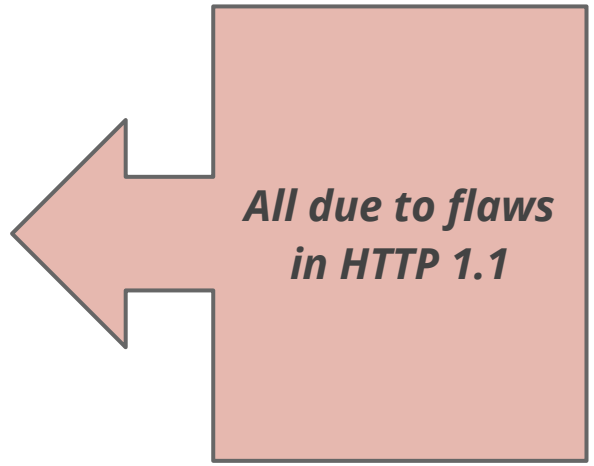
Build on HTTP 1.1

- 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

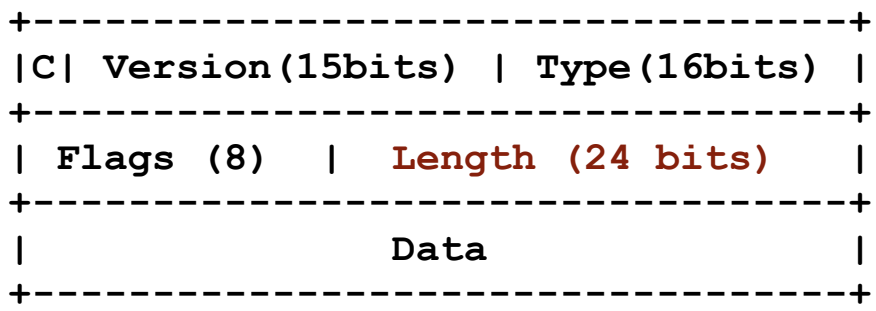
- One TCP connection
- Request = Stream

- Streams are multiplexed
- Streams are prioritized

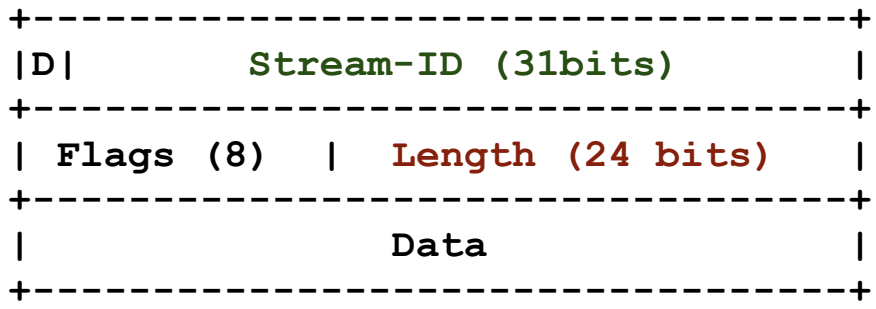
- Binary framing
- Length-prefixed

- Control frames
- Data frames

Control Frame:



Data Frame:



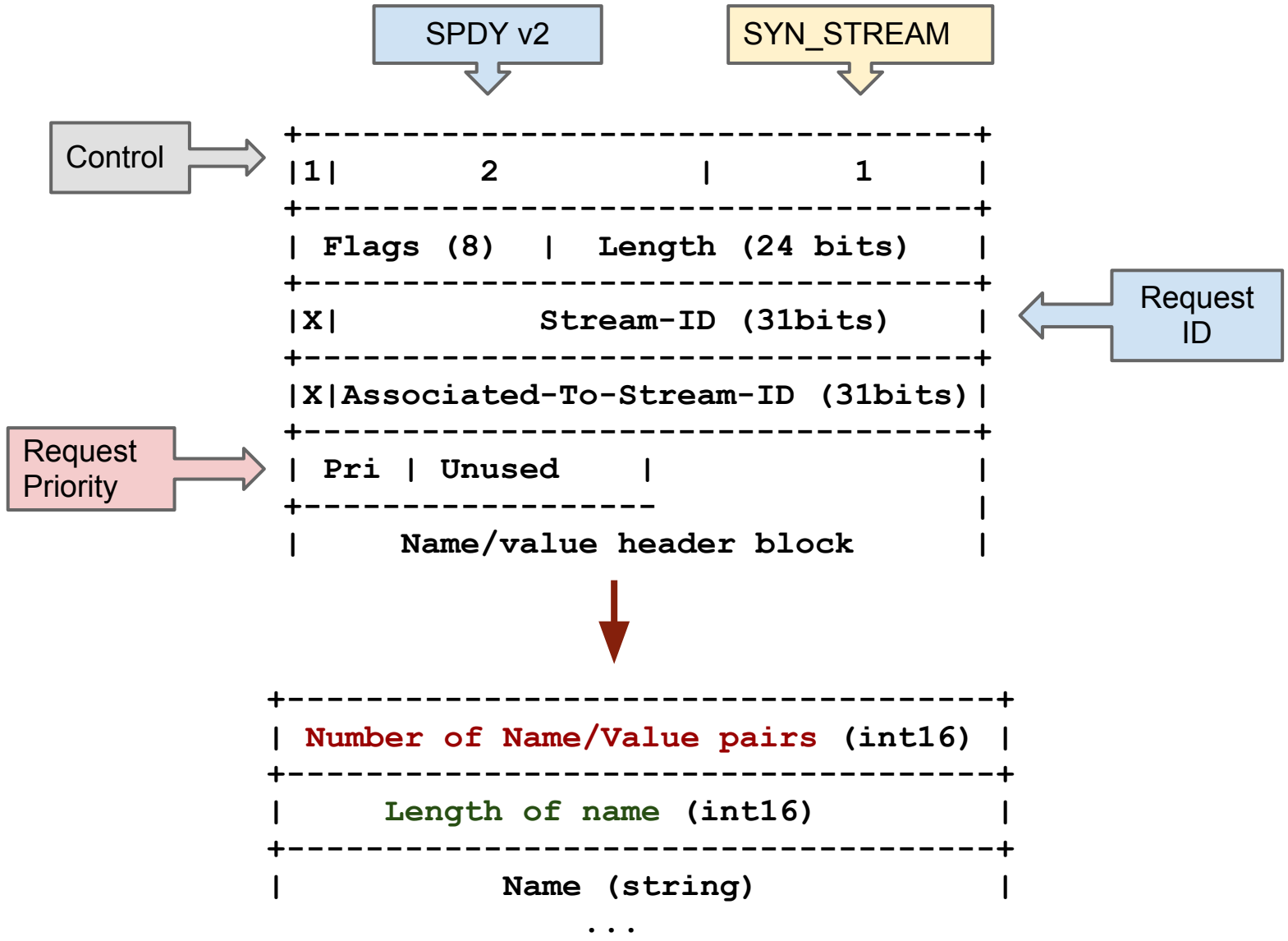
SYN_STREAM

- Server SID: even
- Client SID: odd

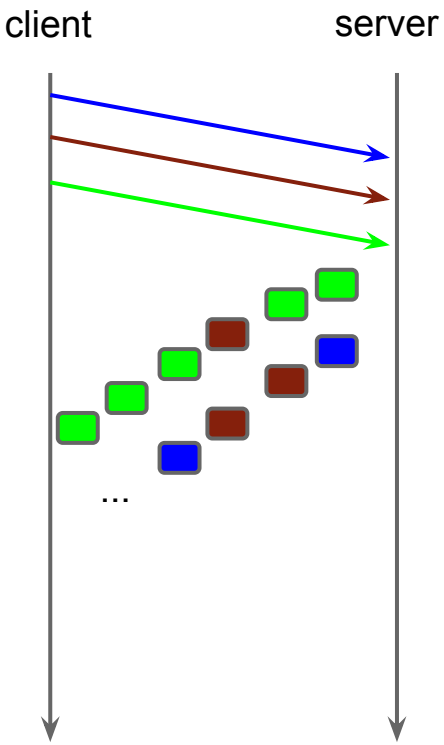
- Associated-To: push *
- Priority: higher, better

- 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)

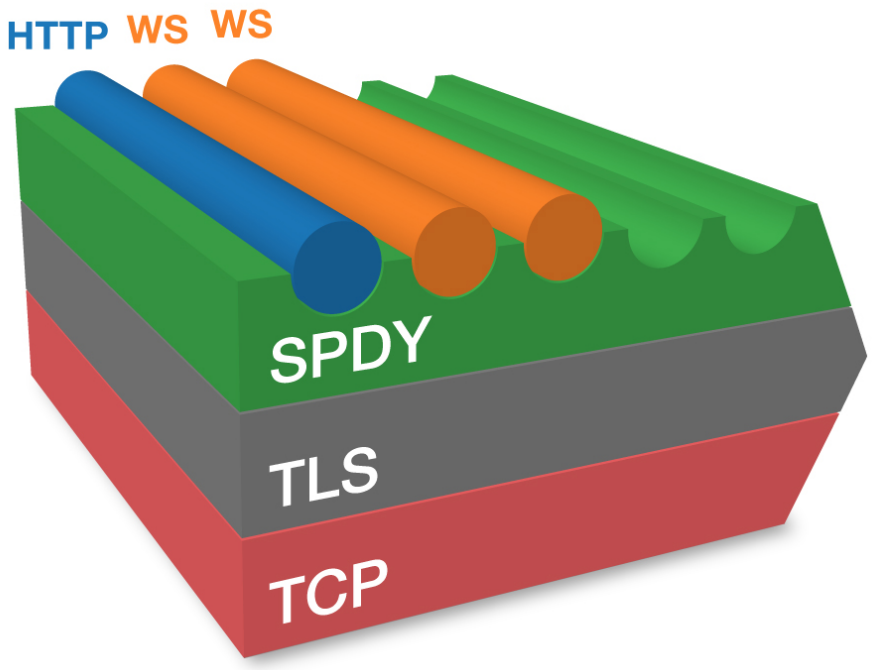
Newsflash: we are already using "server push"

- 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!**

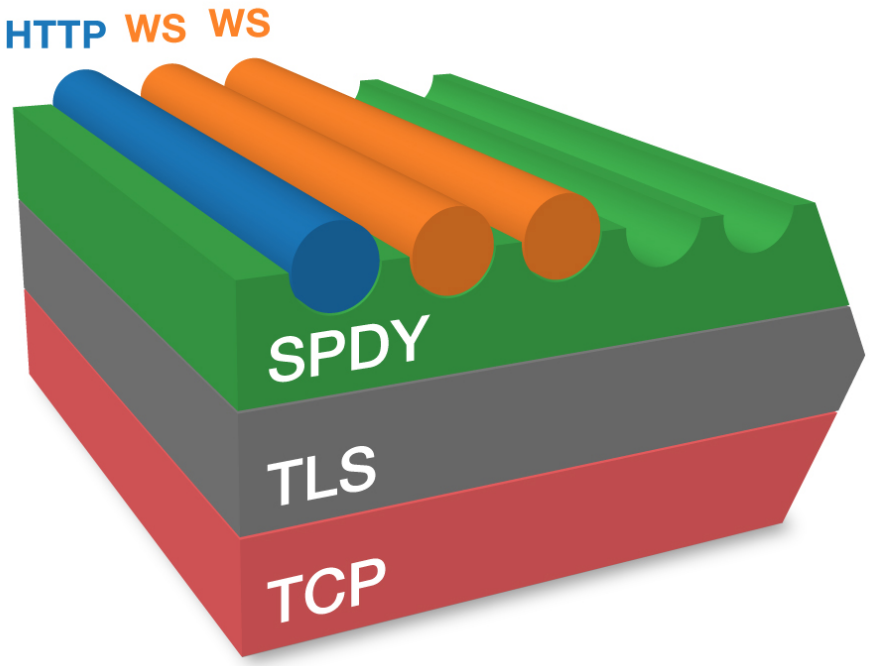
Observation: intermediate proxies get in the way

- 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).

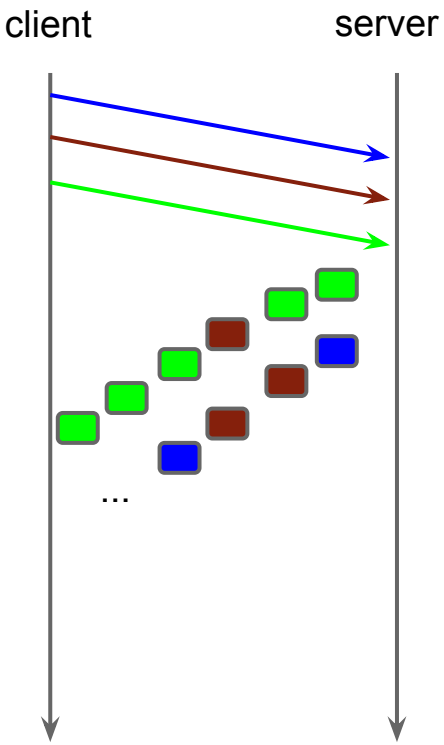
- **Q: Any server optimizations?**
- A: Yes!
  - CWND = 10
  - Check your SSL certificate chain (length)
  - TLS resume, terminate SSL close and early
  - Disable slow start on idle

- **Q: Sounds complicated, can haz lazy?**
- A: 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...

Chrome

(Chrome) Network Stack

An average page has grown to **1059 kB** (over 1MB!) and is now composed of **80+ subresources**.

- **DNS prefetch** - pre-resolve hostnames before we make the request
- **TCP preconnect** - establish connection before we make the request
- **Pooling & re-use** - leverage keep-alive, re-use existing connections (6 per host)
- **Caching** - fastest request is request not made (sizing, validation, eviction, etc)

Ex, Chrome learns subresource domains:

Host for Page	Page Load Count	Subresource Navigations	Subresource PreConnects	Subresource PreResolves	Expected Connects	Subresource Spec
http://www.igvita.com/	3	27	2	0	3.953	http://1-ps.googleusercontent.com/
		3	0	2	0.588	http://fonts.googleapis.com/
		3	0	2	0.588	http://ps.googleusercontent.com/
		8	2	0	1.862	http://www.google-analytics.com/
		9	2	0	1.689	http://www.igvita.com/

(Chrome) Network Stack

- **chrome://predictors** - omnibox predictor stats (check 'Filter zero confidences')
- **chrome://net-internals#sockets** - current socket pool status
- **chrome://net-internals#dns** - Chrome's in-memory DNS cache
- **chrome://histograms/DNS** - histograms of your DNS performance
- **chrome://dns** - startup prefetch list and subresource host cache

```
enum ResolutionMotivation {
    MOUSE_OVER_MOTIVATED,       // Mouse-over link induced resolution.
    PAGE_SCAN_MOTIVATED,        // Scan of rendered page induced resolution.
    LINKED_MAX_MOTIVATED,       // enum demarkation above motivation from links.
    OMNIBOX_MOTIVATED,          // Omni-box suggested resolving this.
    STARTUP_LIST_MOTIVATED,     // Startup list caused this resolution.
    EARLY_LOAD_MOTIVATED,       // In some cases we use the prefetcher to warm up the connection
    STATIC_REFERAL_MOTIVATED,   // External database suggested this resolution.
    LEARNED_REFERAL_MOTIVATED,  // Prior navigation taught us this resolution.
    SELF_REFERAL_MOTIVATED,     // Guess about need for a second connection.
    // ...
};
```

chromiumembedded

A simple framework for embedding chromium browser windows in other applications.

Project Home | Downloads | Wiki | Issues | Source

Summary | People

Project Information

+99 Recommend this on Google

Starred by 516 users

Project feeds

Code license

New BSD License

Labels

Chromium, Embedded, Browser, CPlusPlus, Framework

Members

magreenb...@gmail.com

Links

External links

CEF Forum
CEF1 C++ API Docs
CEF3 C++ API Docs
CEF Donations

Introduction

The Chromium Embedded Framework (CEF) is an open source project founded by Marshall Greenblatt in 2008 to develop a Web browser control based on the Google Chromium project. CEF currently supports a range of programming languages and operating systems and can be easily integrated into both new and existing applications. It was designed from the ground up with both performance and ease of use in mind. The base framework includes C and C++ programming interfaces exposed via native libraries that insulate the host application from Chromium and WebKit implementation details. It provides close integration between the browser control and the host application including support for custom plugins, protocols, JavaScript objects and JavaScript extensions. The host application can optionally control resource loading, navigation, context menus, printing and more, while taking advantage of the same performance and HTML5 technologies available in the Google Chrome Web browser.

Numerous individuals and organizations contribute time and resources to support CEF development, but more involvement from the community is always welcome. This includes support for both the core CEF project and external projects that integrate CEF with additional programming languages and frameworks (see the "External Projects" section below). If you are interested in donating time to help with CEF development please see the "Helping Out" section below. If you are interested in donating money to support general CEF development and infrastructure efforts please visit the CEF Donations page.

Binary Distributions

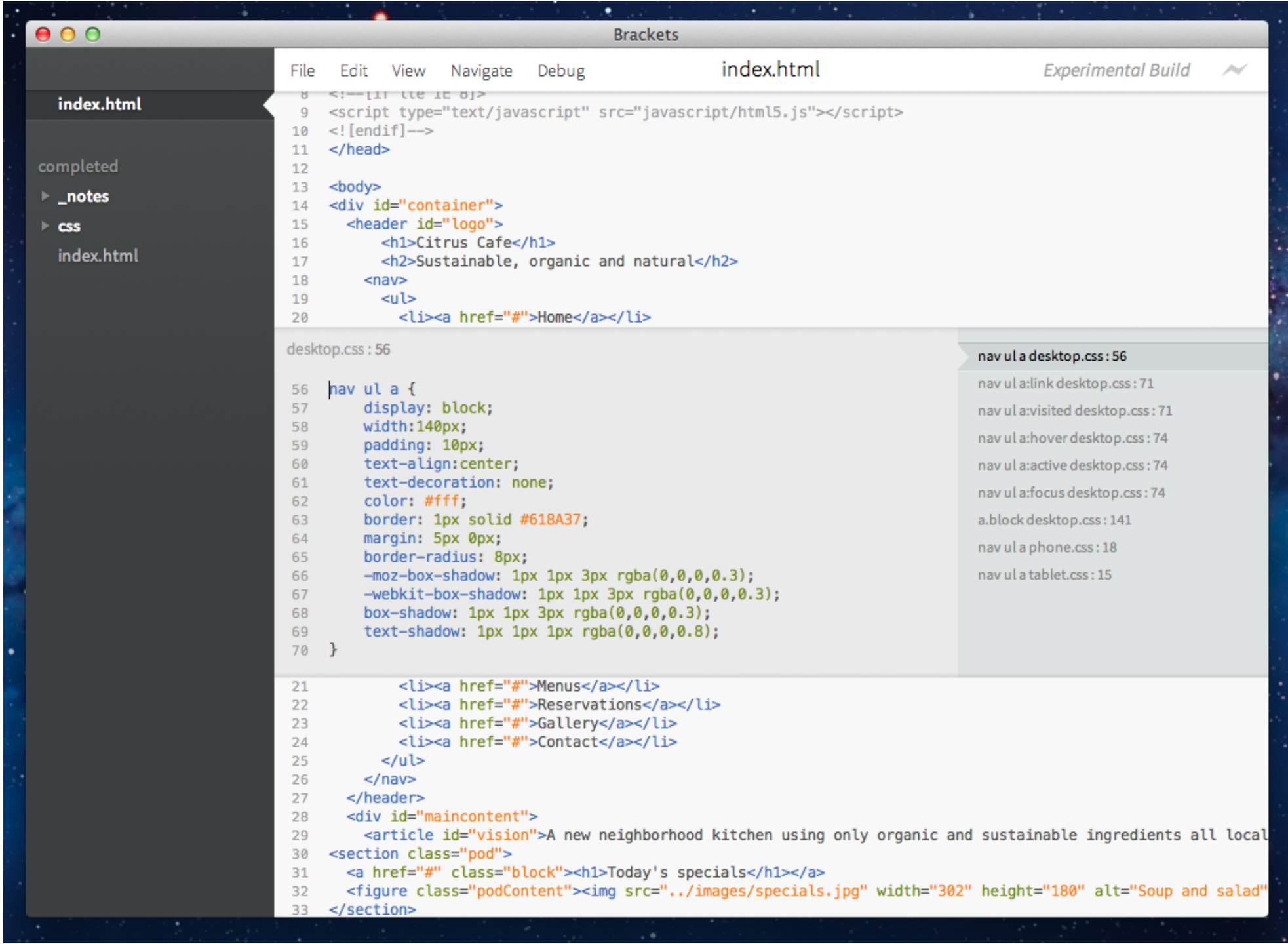
Binary distributions, which include all files necessary to build a CEF-based application, are available in the Downloads section. Binary distributions are stand-alone and do not require the download of CEF or Chromium source code. Symbol files for debugging binary distributions of libcef can be downloaded from here.

Source Distributions

The CEF project is an extension of the Chromium project hosted at http://www.chromium.org. CEF maintains development and release branches that track Chromium branches. CEF source code can be downloaded, built and packaged manually or with automated tools. Visit the BranchesAndBuilding Wiki page for more information.

Brackets

open-source code editor
built *with* the web *for* the web



brackets.io

Build Desktop Applications

for Linux, Windows and Mac using HTML, CSS and Javascript

Download AppJS (v0.0.19) Read Documentation

Why AppJS?

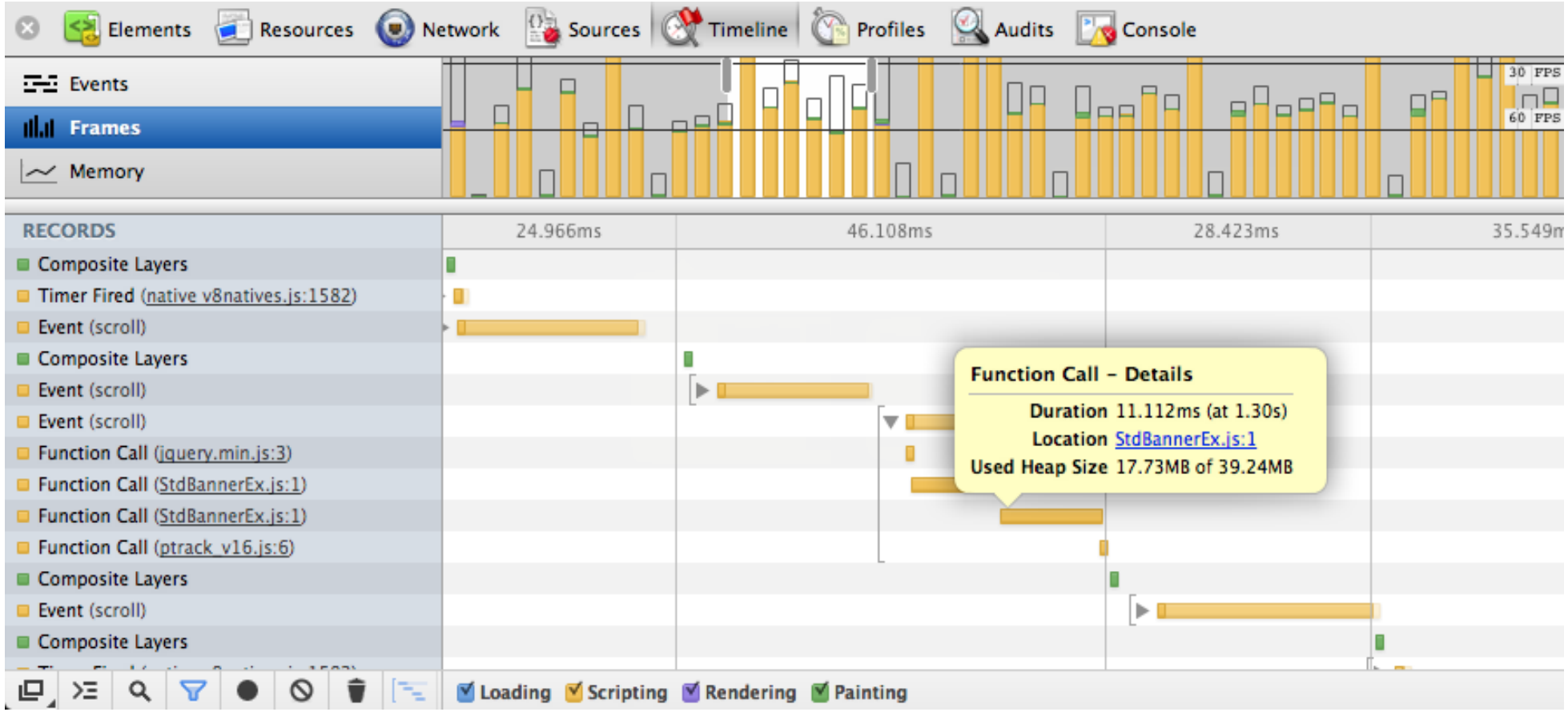
Because it is simple and yet powerful. Using AppJS you don't need to be worry about coding cross-platform or learning new languages and tools. You are already familiar with HTML, CSS and Javascript. What is better than this stack for application development? Beside, AppJS uses Chromium at the core so you get latest HTML 5 APIs working. So relax and focus on the task your application should do.

appjs

Tools, tools, more tools!

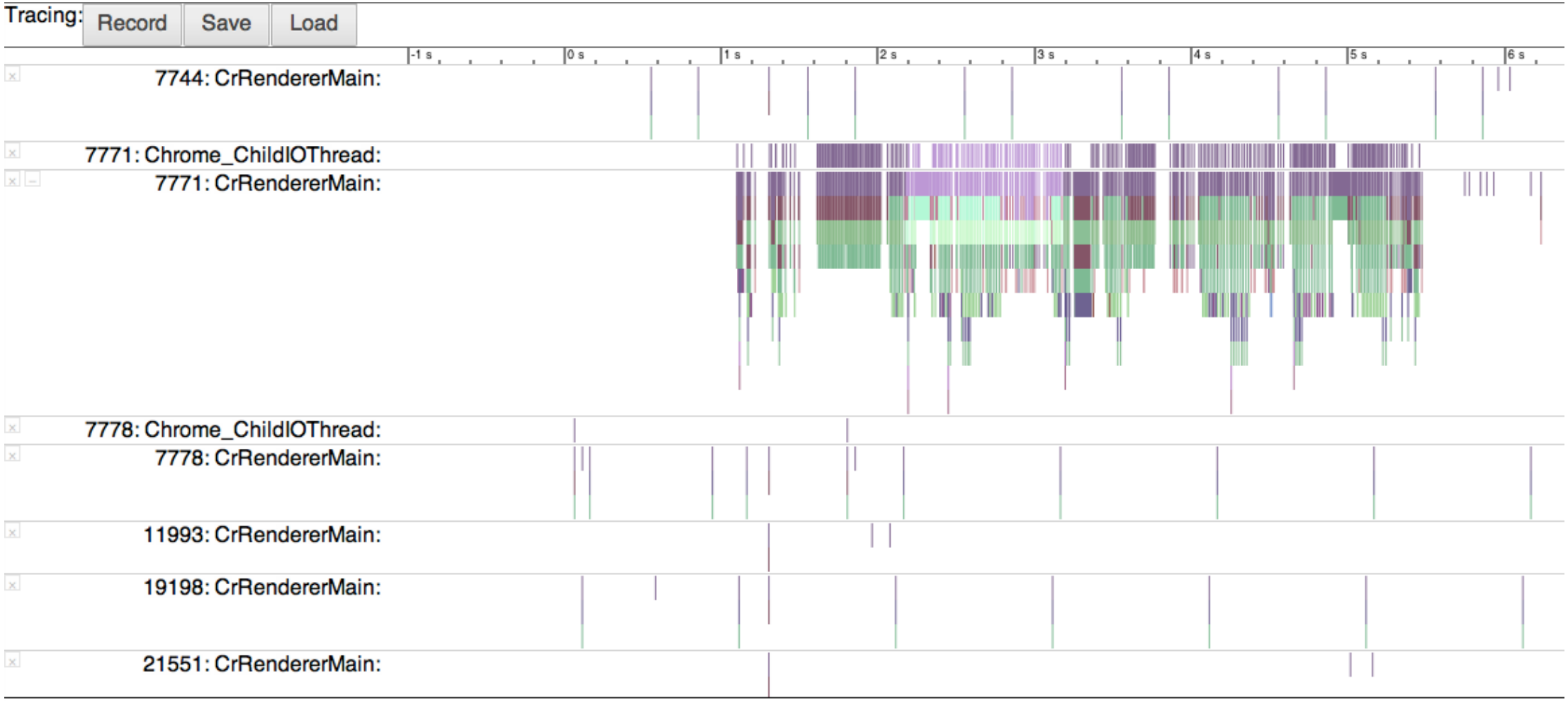
one of the best investments you can make...

60 FPS in the browser? Yes.



- Chrome 21+ > Timeline > **Frames**
- 60 FPS target affords you a **16.6 ms** budget per frame

chrome://tracing



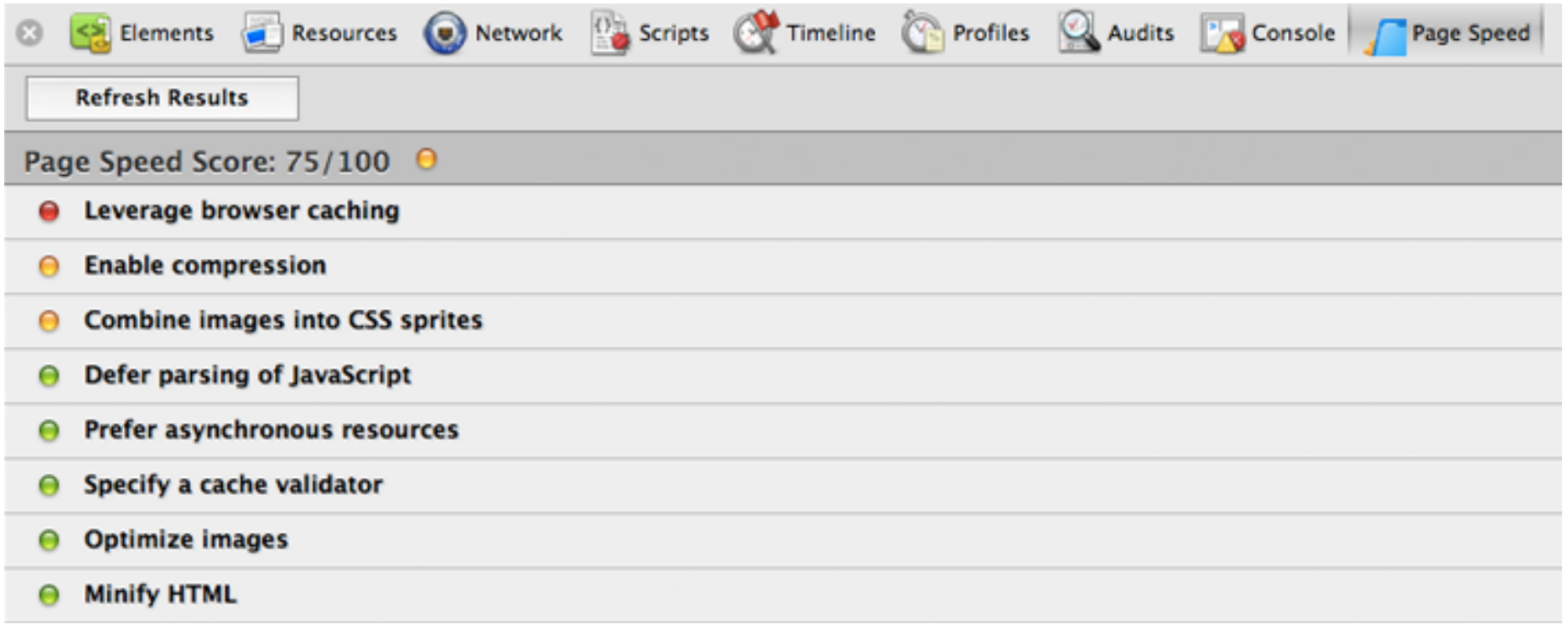
Slices:

MessageLoop::RunTask	589.791 ms	492 occurences
ChannelProxy::Context::OnDispatchMessage	400.154 ms	196 occurences

PageSpeed

diagnostics and optimization...

PageSpeed Insights + Browser Plugins



- Free browser plugins for Chrome & Firefox
- **Open-source SDK** for embedding (same rules, same code)
- Active pagespeed community

PageSpeed + Apache: mod_pagespeed

```
<VirtualHost *:80>
    ServerName www.awesome-app.com
    DocumentRoot /apps/foo/public

    ModPagespeed on
    /* ... */
</VirtualHost>
```

Open-source Apache module that automatically optimizes web pages and associated resources.

combine_css	combine_javascript	extend_cache
inline_css	inline_import_to_link	inline_javascript
inline_images	resize_mobile_images	insert_image_dimensions
convert_png_to_jpeg	convert_jpeg_to_webp	convert_jpeg_to_progressive
defer_javascript	lazyload_images	inline_preview_images ...

mod_pagespeed filters

Thanks! Questions?

Slides @ **bit.ly/perfloop**

@igrigorik

igvita.com