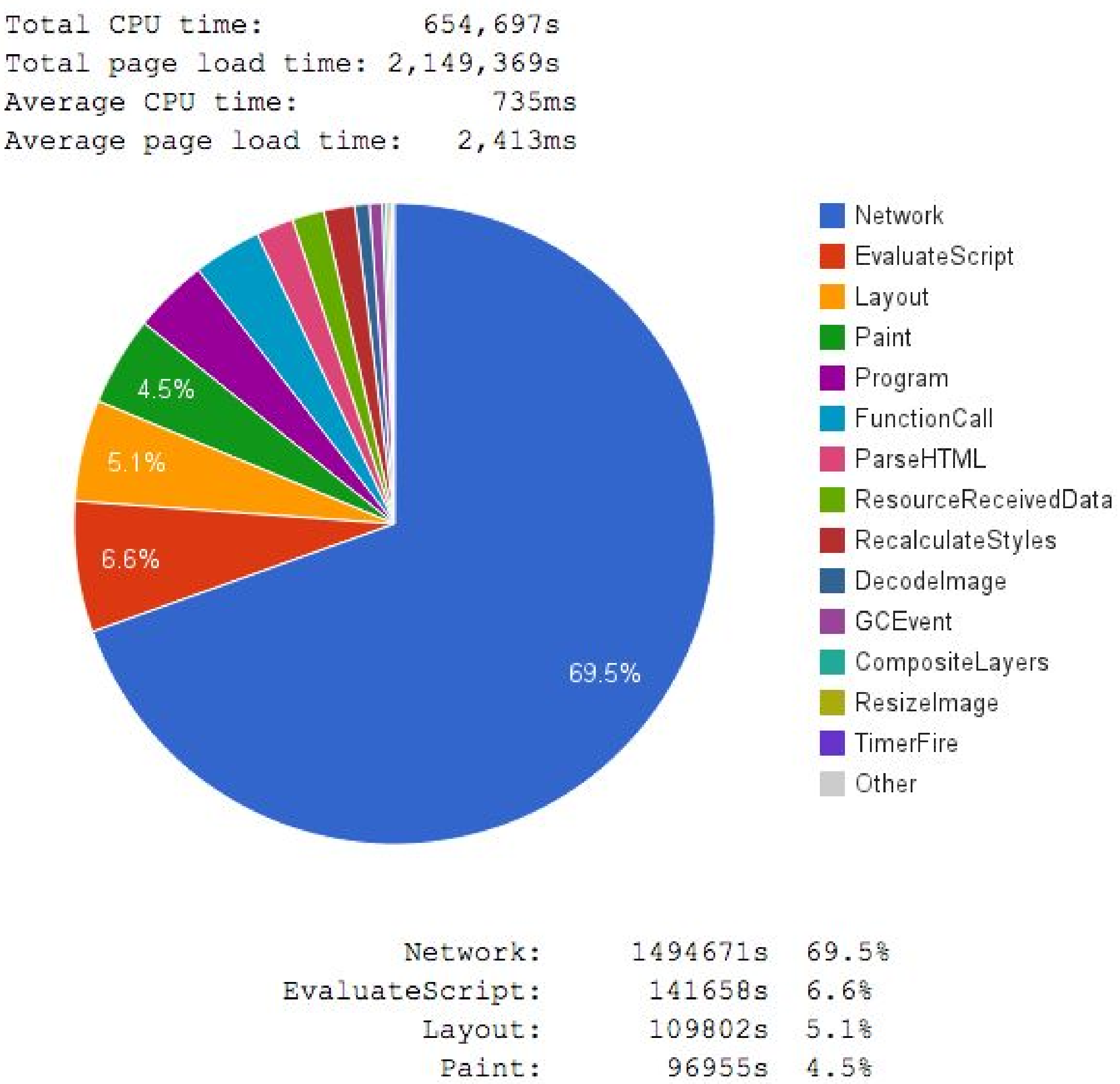
Preconnect, prefetch, prerender ...

aka, building a web performance oracle in your application!

Ilya Grigorik - @igrigorik

igrigorik@google.com



```
Total CPU time:         654,697s
Total page load time: 2,149,369s
Average CPU time:          735ms
Average page load time:  2,413ms
```

```
       Network:     1494671s   69.5%
EvaluateScript:      141658s   6.6%
        Layout:      109802s   5.1%
         Paint:       96955s   4.5%
```

Top 1M Alexa sites...

- Cable profile (5Mbps / 28 ms RTT)
- Main thread attribution in Blink
  - Measured via Telemetry

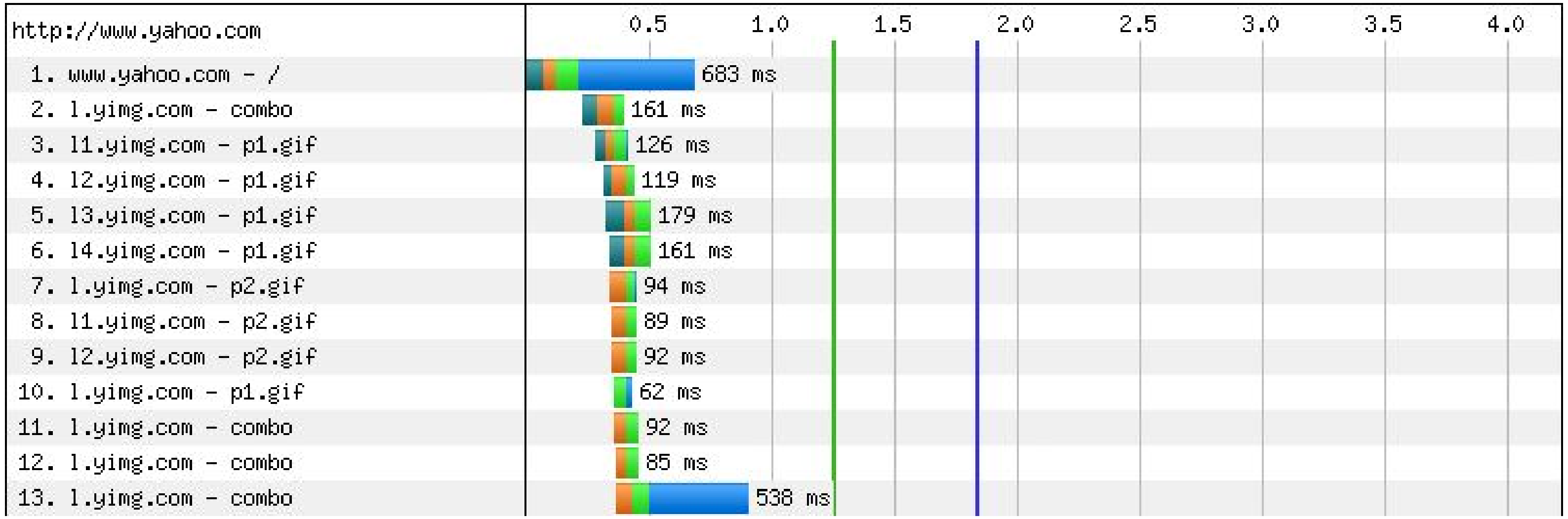
- **69.5% of time blocked on network**
- **6.6% of time blocked JavaScript**
- **5.1% blocked on Layout**
- **4.5% blocked on Paint**
- **...**

- **No surprise here (hopefully)**
  - First page load is network bound
  - First page load is latency bound

blink-dev thread

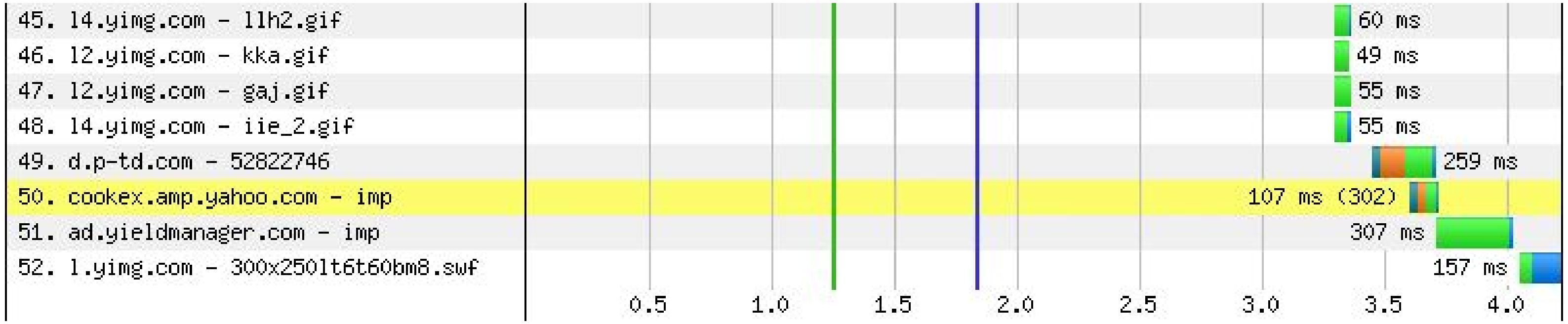
@igrigorik

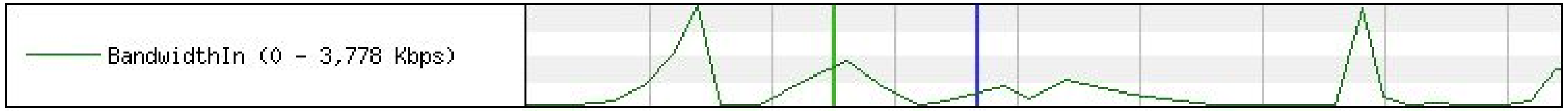
Our pages consist of dozens of assets


http://www.yahoo.com

1. www.yahoo.com - / 683 ms
2. l.yimg.com - combo 161 ms
3. l1.yimg.com - p1.gif 126 ms
4. l2.yimg.com - p1.gif 119 ms
5. l3.yimg.com - p1.gif 179 ms
6. l4.yimg.com - p1.gif 161 ms
7. l.yimg.com - p2.gif 94 ms
8. l1.yimg.com - p2.gif 89 ms
9. l2.yimg.com - p2.gif 92 ms
10. l.yimg.com - p1.gif 62 ms
11. l.yimg.com - combo 92 ms
12. l.yimg.com - combo 85 ms
13. l.yimg.com - combo 538 ms

... (snip 30 requests) ...


45. l4.yimg.com - llh2.gif 60 ms
46. l2.yimg.com - kka.gif 49 ms
47. l2.yimg.com - gaj.gif 55 ms
48. l4.yimg.com - iie_2.gif 55 ms
49. d.p-td.com - 52822746 259 ms
50. cookex.amp.yahoo.com - imp 107 ms (302)
51. ad.yieldmanager.com - imp 307 ms
52. l.yimg.com - 300x250lt6t60bm8.swf 157 ms

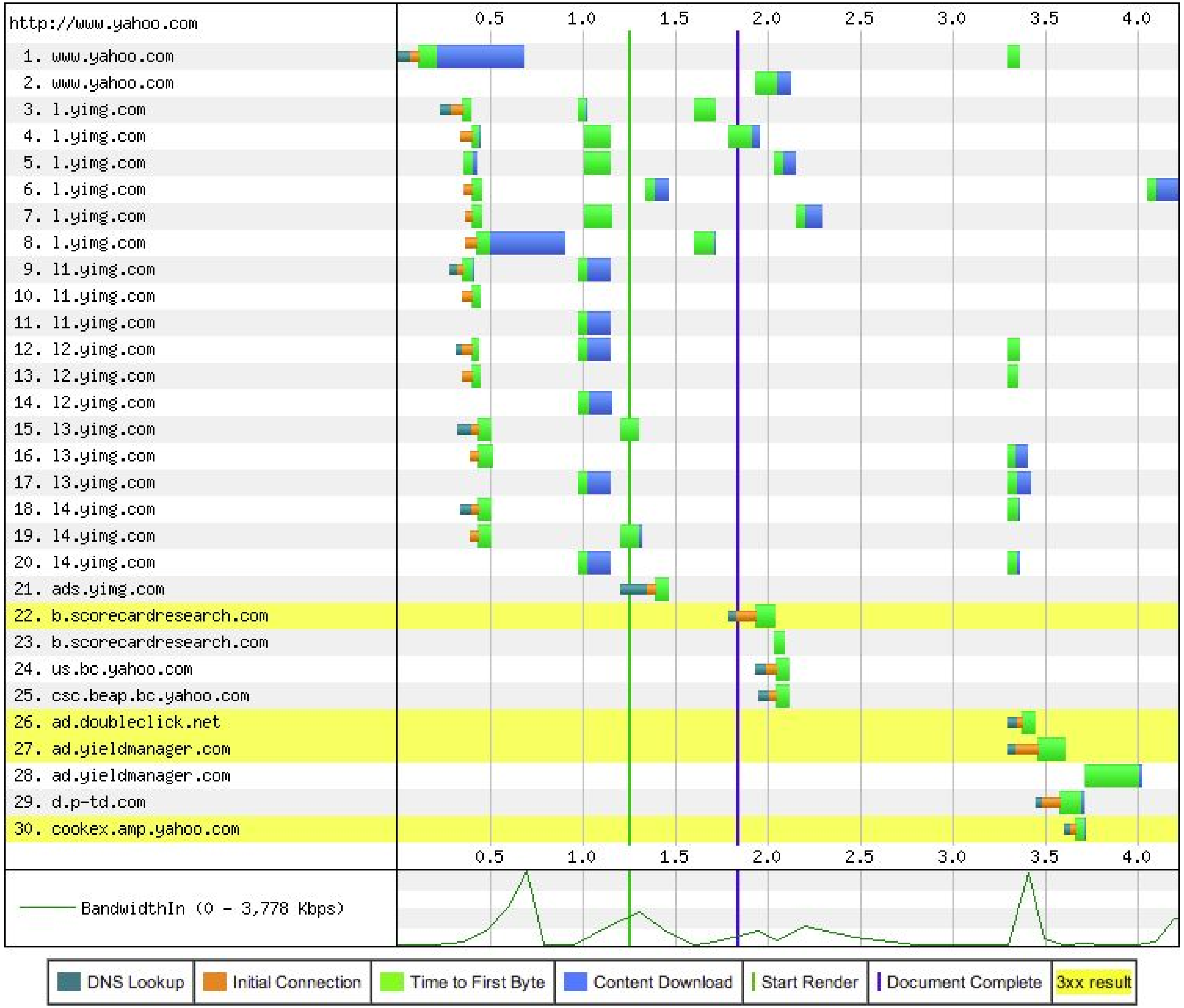


- 52 requests
- 4+ seconds

Huh?

Primer on Web Performance (Chapter 10)

“Connection view” tells the story...



- 30 connections
  - DNS lookups
  - TCP handshakes
  - ...
- **Blue:** download time

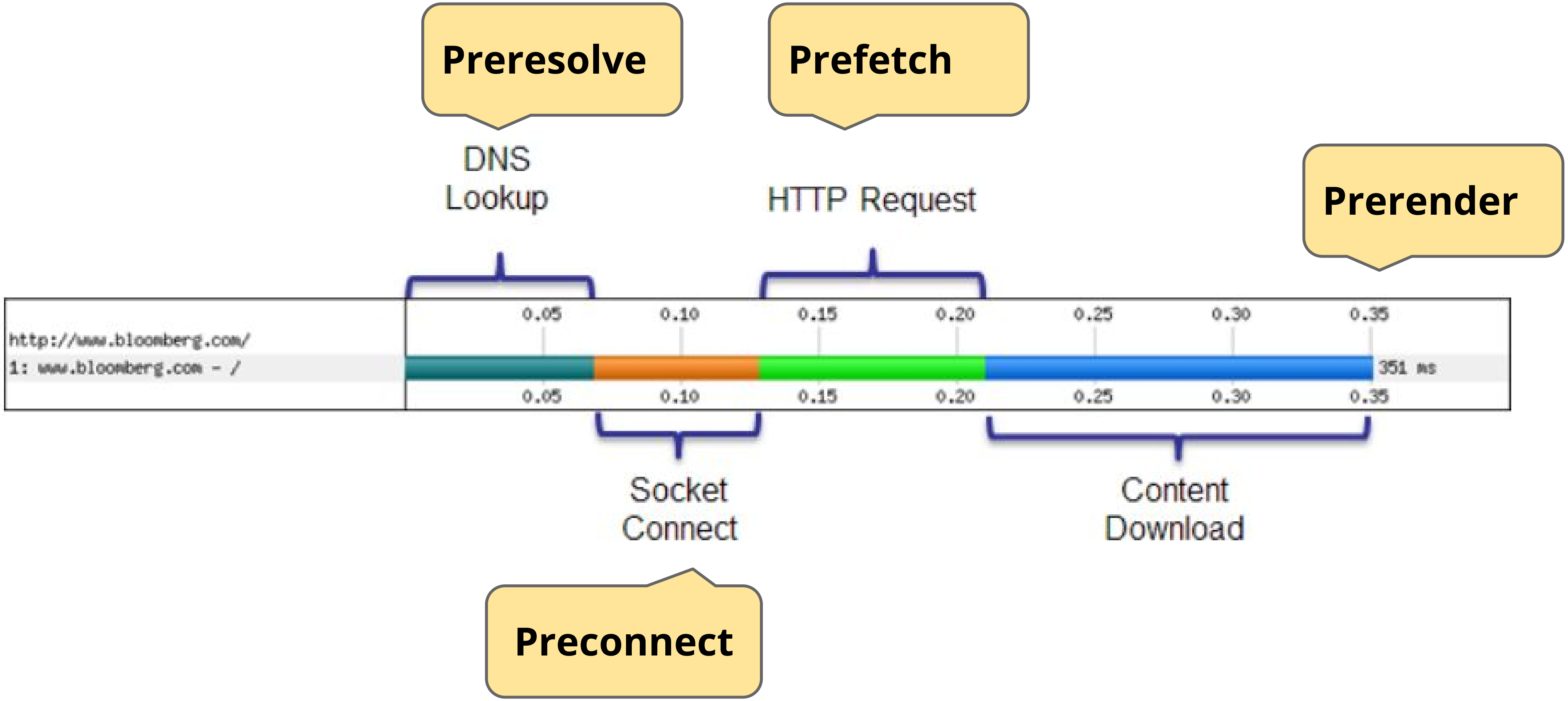
We’re not BW limited, we’re literally idling, waiting on the network to deliver resources.

@igrigorik

Let’s build a smarter browser!

We can hide some of the network latency through clever tricks.

The pre- party...*



Preresolve

DNS Lookup

Prefetch

HTTP Request

Prerender

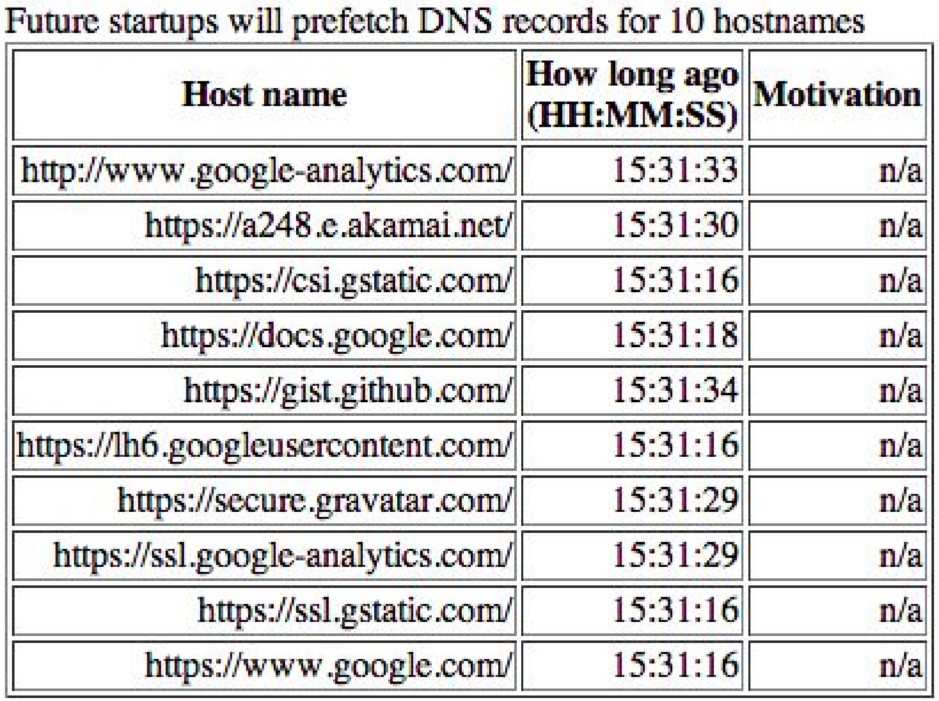
Socket Connect

Content Download

Preconnect

Primer on Web Performance (Chapter 10)

Pre-resolve DNS names on browser startup...



Future startups will prefetch DNS records for 10 hostnames

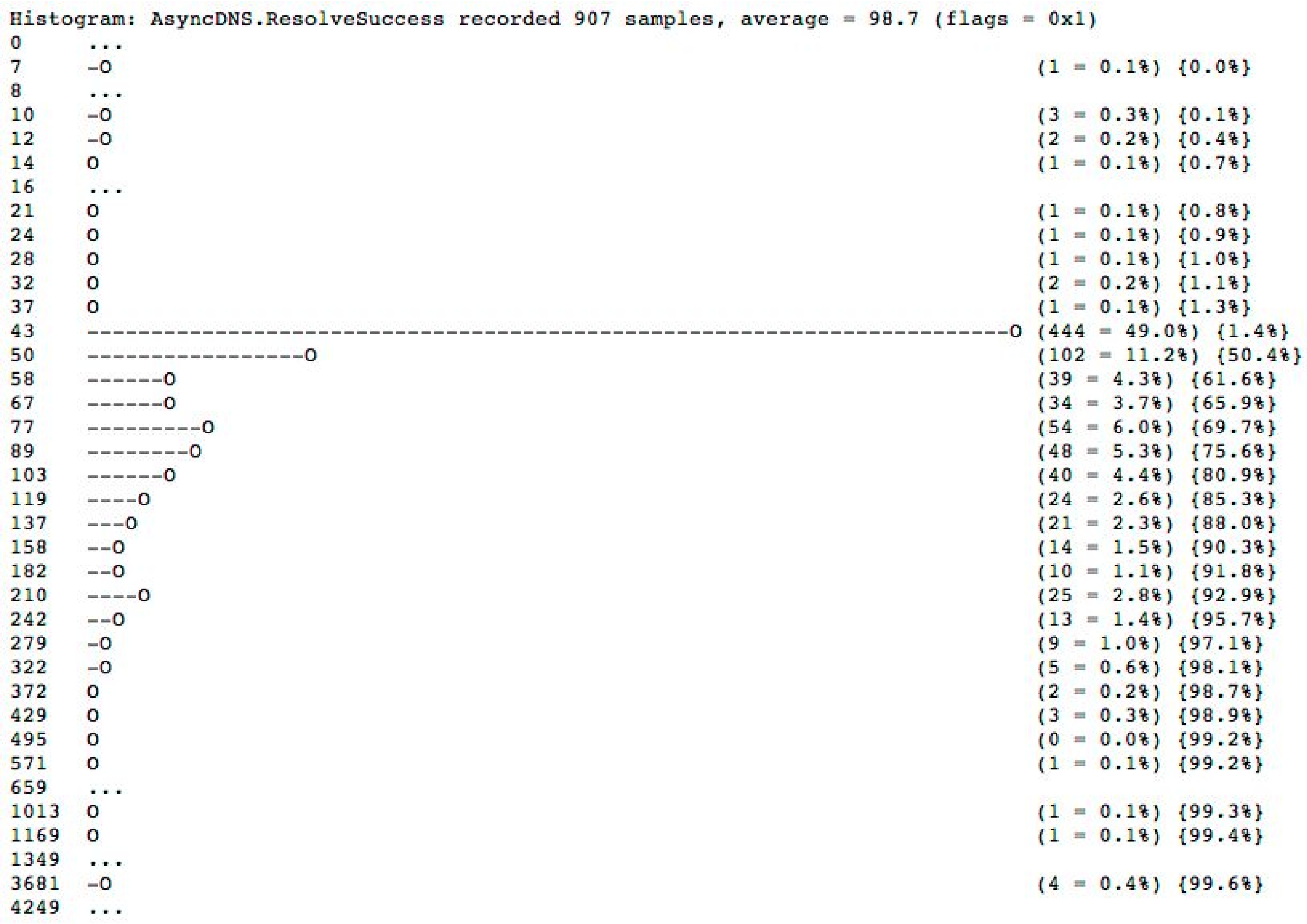
Host name	How long ago (HH:MM:SS)	Motivation
http://www.google-analytics.com/	15:31:33	n/a
https://a248.e.akamai.net/	15:31:30	n/a
https://csi.gstatic.com/	15:31:16	n/a
https://docs.google.com/	15:31:18	n/a
https://gist.github.com/	15:31:34	n/a
https://lh6.googleusercontent.com/	15:31:16	n/a
https://secure.gravatar.com/	15:31:29	n/a
https://ssl.google-analytics.com/	15:31:29	n/a
https://ssl.gstatic.com/	15:31:16	n/a
https://www.google.com/	15:31:16	n/a

- **Scenario:** when you load the browser first thing in the morning (or after restart), where do you usually head?

- **Let's pre-resolve all of the popular names!**
  - Chrome resolves top 10 destinations.

Head to **chrome://dns/** to see your list.

Preresolve is cute, but does it help much?



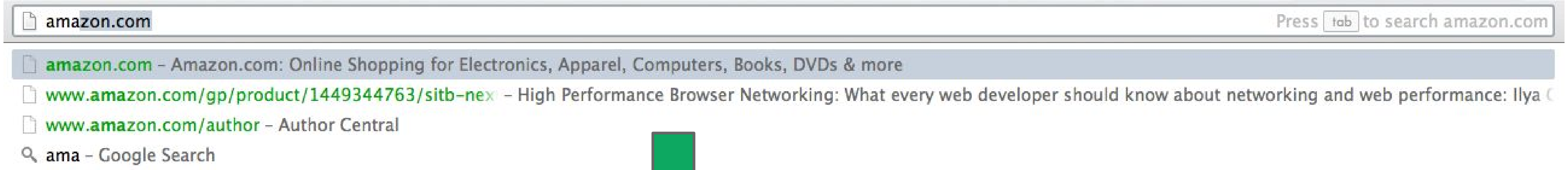
```
Histogram: AsyncDNS.ResolveSuccess recorded 907 samples, average = 98.7 (flags = 0x1)
0     ...
7     -O                                                                        (1 = 0.1%) {0.0%}
8     ...
10    -O                                                                        (3 = 0.3%) {0.1%}
12    -O                                                                        (2 = 0.2%) {0.4%}
14    O                                                                         (1 = 0.1%) {0.7%}
16    ...
21    O                                                                         (1 = 0.1%) {0.8%}
24    O                                                                         (1 = 0.1%) {0.9%}
28    O                                                                         (1 = 0.1%) {1.0%}
32    O                                                                         (2 = 0.2%) {1.1%}
37    O                                                                         (1 = 0.1%) {1.3%}
43    ------------------------------------------------------------------------O (444 = 49.0%) {1.4%}
50    -----------------O                                                        (102 = 11.2%) {50.4%}
58    ------O                                                                   (39 = 4.3%) {61.6%}
67    ------O                                                                   (34 = 3.7%) {65.9%}
77    ---------O                                                                (54 = 6.0%) {69.7%}
89    --------O                                                                 (48 = 5.3%) {75.6%}
103   ------O                                                                   (40 = 4.4%) {80.9%}
119   ----O                                                                     (24 = 2.6%) {85.3%}
137   ---O                                                                      (21 = 2.3%) {88.0%}
158   --O                                                                       (14 = 1.5%) {90.3%}
182   --O                                                                       (10 = 1.1%) {91.8%}
210   ----O                                                                     (25 = 2.8%) {92.9%}
242   --O                                                                       (13 = 1.4%) {95.7%}
279   -O                                                                        (9 = 1.0%) {97.1%}
322   -O                                                                        (5 = 0.6%) {98.1%}
372   O                                                                         (2 = 0.2%) {98.7%}
429   O                                                                         (3 = 0.3%) {98.9%}
495   O                                                                         (0 = 0.0%) {99.2%}
571   O                                                                         (1 = 0.1%) {99.2%}
659   ...
1013  O                                                                         (1 = 0.1%) {99.3%}
1169  O                                                                         (1 = 0.1%) {99.4%}
1349  ...
3681  -O                                                                        (4 = 0.4%) {99.6%}
4249  ...
```

- **How fast is your local DNS?**
  Chrome knows the answer…

  chrome://histograms/DNS

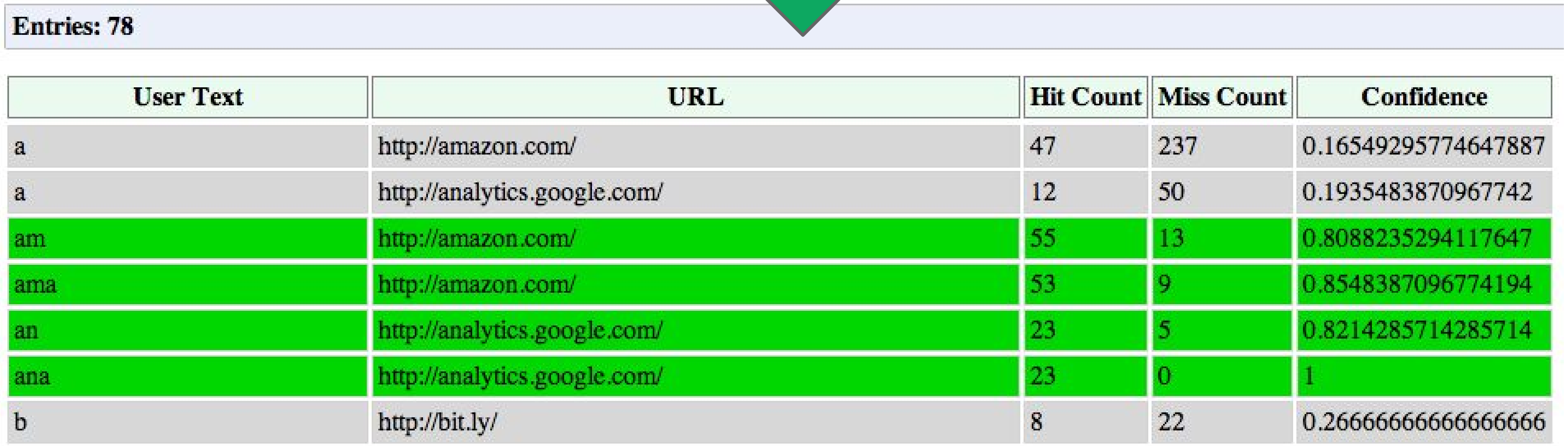
- **Good case: < 30 ms**
- **Average: 30-100 ms**
- **Ouch: 100 ms+**

Let's predict where you're heading next...



amazon.com — Press tab to search amazon.com

- amazon.com – Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more
- www.amazon.com/gp/product/1449344763/sitb-nex – High Performance Browser Networking: What every web developer should know about networking and web performance: Ilya
- www.amazon.com/author – Author Central
- ama – Google Search



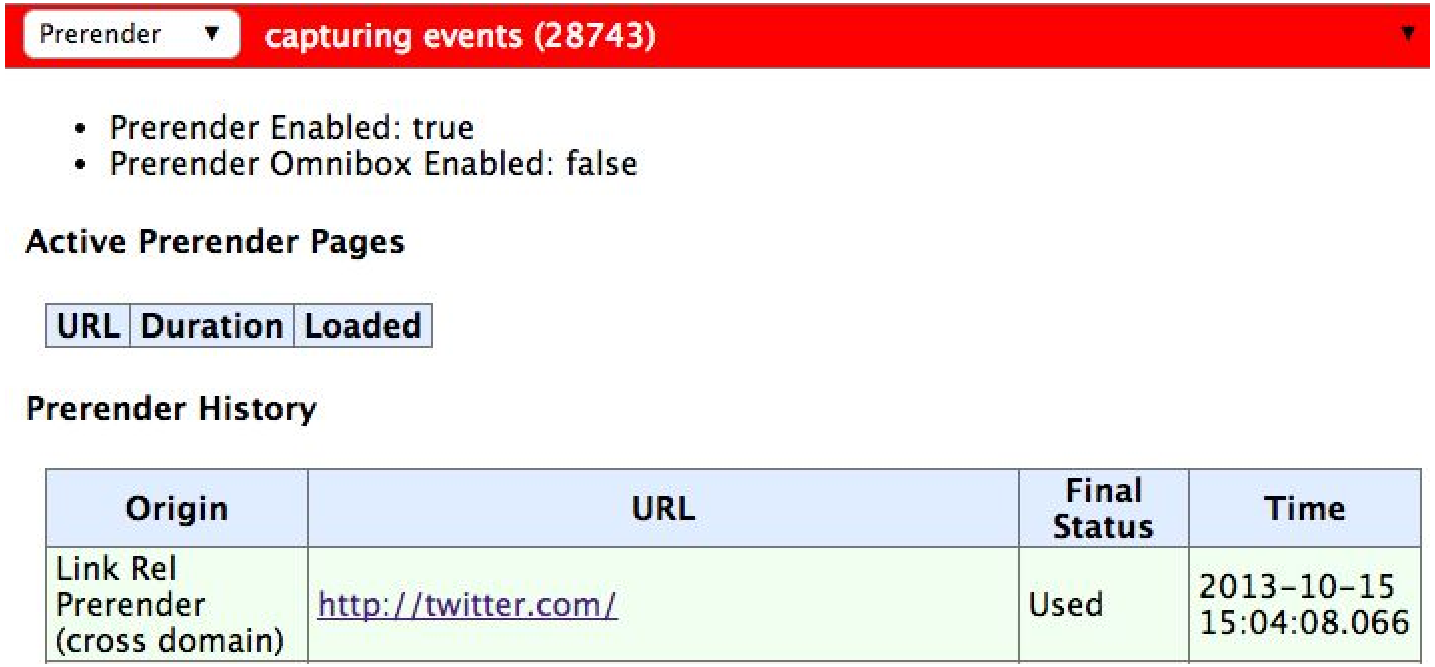
Entries: 78

User Text	URL	Hit Count	Miss Count	Confidence
a	http://amazon.com/	47	237	0.16549295774647887
a	http://analytics.google.com/	12	50	0.1935483870967742
am	http://amazon.com/	55	13	0.8088235294117647
ama	http://amazon.com/	53	9	0.8548387096774194
an	http://analytics.google.com/	23	5	0.8214285714285714
ana	http://analytics.google.com/	23	0	1
b	http://bit.ly/	8	22	0.26666666666666666

If you type in "ama" what's the likelihood you're heading to Amazon?

- Chrome tracks the hit / miss count, and uses it to initiate **DNS preresolve** and **TCP preconnect**!
- High confidence hits may trigger a **full prerender** in a background tab.
- Head to **chrome://predictors/** to see your list.

Hmm, pre-rendering you say? Tell me more...



- Prerender Enabled: true
- Prerender Omnibox Enabled: false

Active Prerender Pages

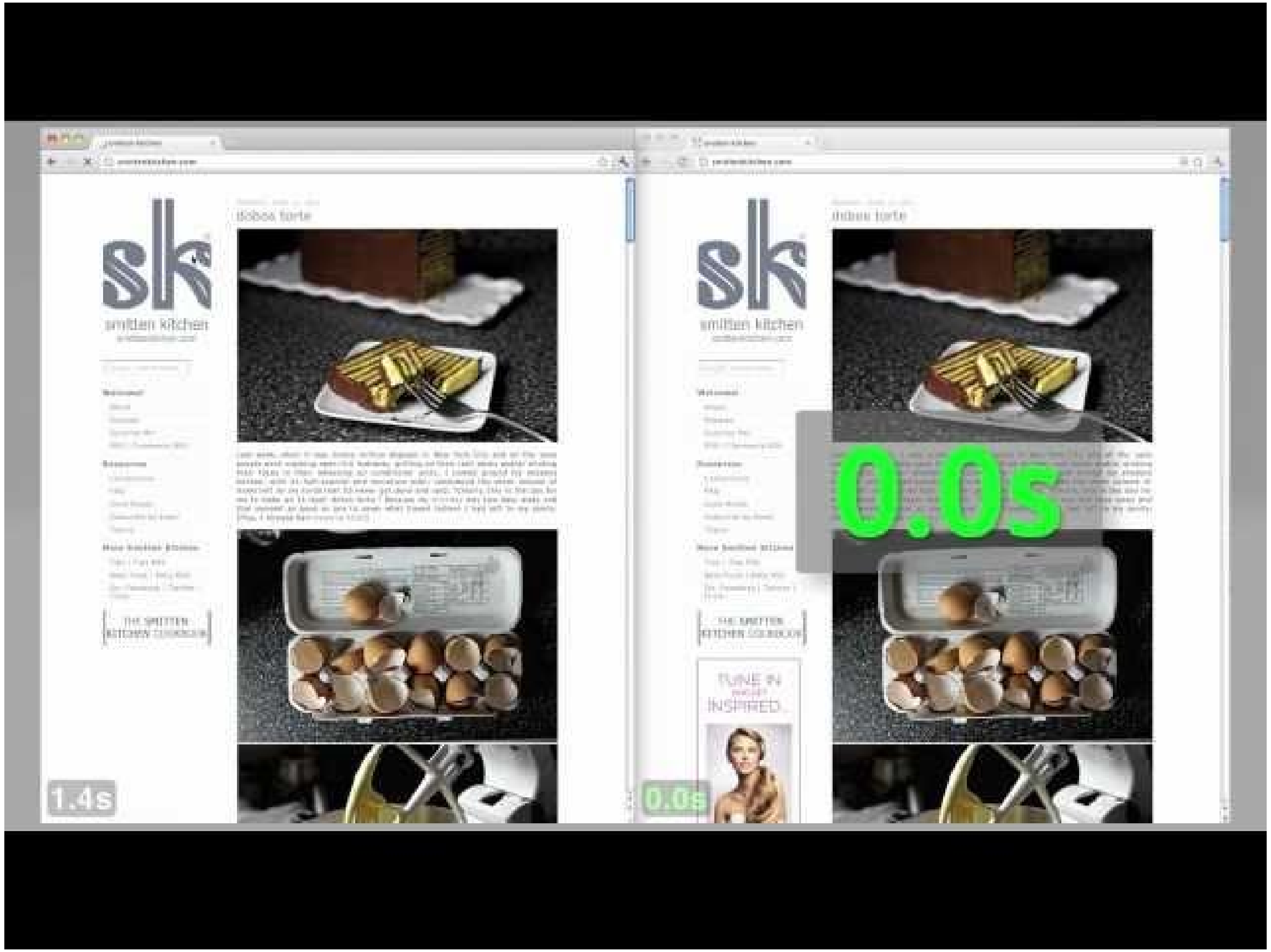
URL	Duration	Loaded

Prerender History

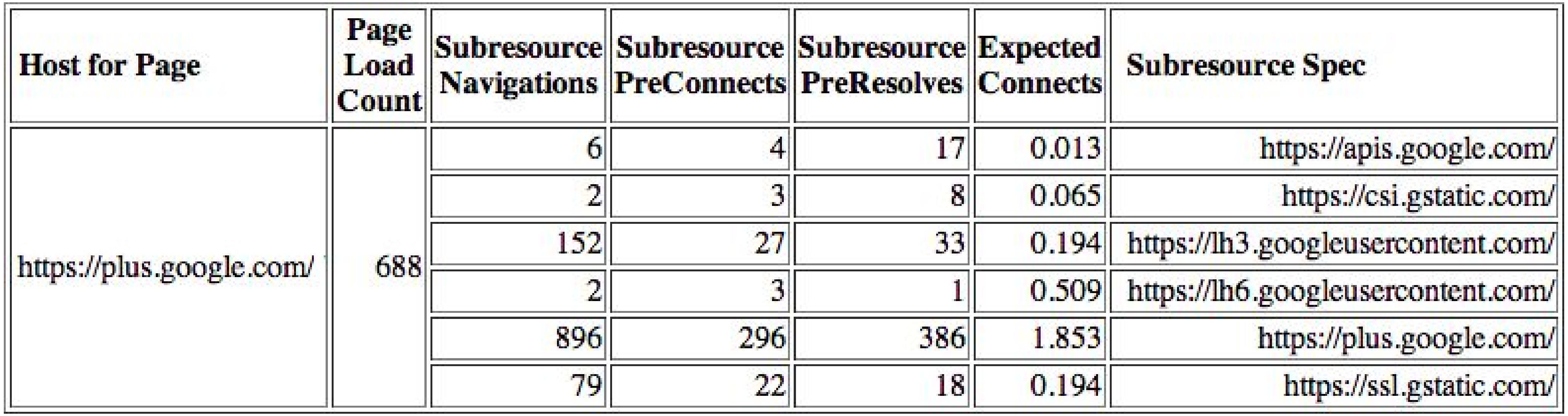
Origin	URL	Final Status	Time
Link Rel Prerender (cross domain)	http://twitter.com/	Used	2013-10-15 15:04:08.066

- Head to **chrome://net-internals/#prerender**
- Try it yourself via prerender-test.appspot.com.

*Instant Pages *is* Chrome Prerendering!*



Could we optimize repeat visits further? Why, yes!

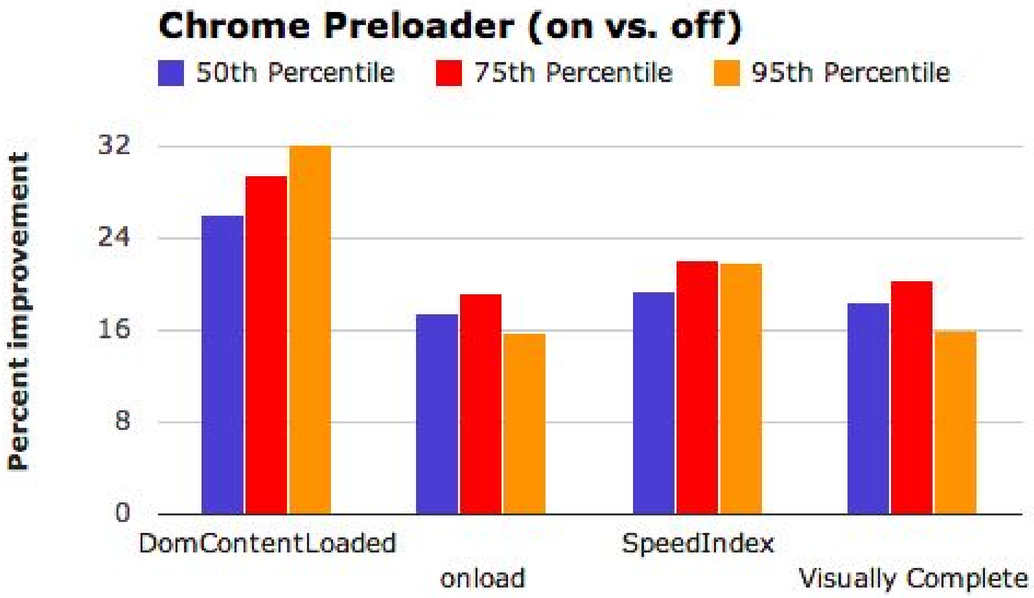


Host for Page	Page Load Count	Subresource Navigations	Subresource PreConnects	Subresource PreResolves	Expected Connects	Subresource Spec
https://plus.google.com/	688	6	4	17	0.013	https://apis.google.com/
		2	3	8	0.065	https://csi.gstatic.com/
		152	27	33	0.194	https://lh3.googleusercontent.com/
		2	3	1	0.509	https://lh6.googleusercontent.com/
		896	296	386	1.853	https://plus.google.com/
		79	22	18	0.194	https://ssl.gstatic.com/

- Remember subresource hostnames + track stats on pre{connect, resolve} rates
- Use above information on future navigations to initiate appropriate actions...

Check your own site: **chrome://dns**

Can we *discover critical resources quicker*? Yep...



Chrome's preloader delivers a **~20% speed improvement!**

- Blocking resources block DOM construction...
- Preload scanner "looks ahead" in the HTML document to identify critical resources
  - JavaScript, CSS, etc.

Don't hide your resources from the preload scanner! E.g. JS loaders, polyfills, etc.

Chrome preloader

*“**We were using XHR to download CSS**... When it came it our attention that XHRs are requested at low priority we decided to run an experiment to see its impact on G+ latency (vs using declarative markup like <link>).*

*In SPDY capable browsers it (using <link>) resulted in a big latency improvement. **In Chrome 27 we saw a 4x speedup at the median**, and 5x at 25th percentile. **In Firefox 21 we saw a 5x speedup at median**, and 8x at 25th percentile.”*

Shubhie Panicker - G+ Front-end Team

In short, Chrome does a lot!

but you can help it...

The browser is trying to predict and anticipate user activity, but ***you have the app-specific insights - leverage them!***

1. Pre-resolve DNS hostnames
2. Mark critical subresources (don't hide them!)
3. Prefetch critical resources
4. Prerender where applicable

Embed “dns-prefetch” hints...

```
<link rel="dns-prefetch" href="hostname_to_resolve.com">
<link rel="dns-prefetch" href="host2.com">
```

Embed prefetch hints in <head> to hint the browser to pre-resolve these names.

- Useful for critical resources later in the page
- Useful for resources behind a redirect
  - *host1.com/resource > 301 > host2.com/resouce*
    - *dns-prefetch: host2.com*
  - *(or even better, eliminate the redirect :))*

Embed “subresource” hints...

```
<link rel="subresource" href="critical/app.js">
<link rel="subresource" href="critical/style.css">
```

Embed subresource hints in <head> to initiate immediate fetch for **current** page!

- Subresource hint identifies critical resources required for current page load.
- Place subresource hints as early as possible.
  - *In a way, this is a “manual preload scanner” strategy ...*

Embed “prefetch” hints...

```
<link rel="prefetch" href="checkout.html">
<link rel="prefetch" href="other-styles.css">
```

Embed prefetch hints in <head> to initiate deferred fetch for **later** navigation.

- Prefetch hint identifies resources that may be needed in **future navigation.**
- Prefetch hints have lowest possible priority.
- Prefetch hints are content agnostic: fetch asset, place in cache.
  - *You do have cache headers enabled, right? Right?*

Embed “prerender” hints...

```
<link rel="prerender" href="checkout.html">
```

Embed prerender hints in <head> to initiate background prerender of entire page!

- The page is fetch, and all of its assets!
- Use **Page Visibility API** to defer JS actions until page is visible.
  - Analytics beacons (GA does this already), custom code, etc.

- Only “safe” pages can be prerendered (aka, GET).
- Prerendering is resource heavy - use with caution.

Predict, measure, optimize... iterate.

You can inject each of the hints when the page is generated

- *You know the structure of the page / application, use it...*
- *Run offline log analysis (e.g. step_a.html > step_b.html)*

You can inject hints “at runtime” based on user interactions!

- *Via the magic of JavaScript, simply add the appropriate link tag:*

```
var hint = document.createElement("link")
hint.setAttribute("rel", "prerender")
hint.setAttribute("href", "next-page.html")

document.getElementsByTagName("head")[0].appendChild(hint)
```

P.S. If the hint is no longer relevant, reverse works also.. nuke it from the DOM!

TL;DR

1. **<link rel="dns-prefetch" href="hostname_to_resolve.com">**
   a. *Pre-resolve DNS hostnames for assets later in the page! (Most browsers)*
2. **<link rel="subresource" href="/javascript/myapp.js">**
   a. *Initiate early resource fetch for current navigation (Chrome only)*
3. **<link rel="prefetch" href="/images/big.jpeg">**
   a. *Prefetch asset for a future navigation, place in cache… (Most browsers)*
4. **<link rel="prerender" href="//example.org/next_page.html">**
   a. *Prerender page in background tab for future navigation*

Twitter	igrigorik
Email	igrigorik@google.com
Web	igvita.com

- Slides @ **`bit.ly/1bUFCsI`**
- Checkout Steve's **prebrowsing slides**!