Is The Web HTTP/2 Yet?

http://isthewebhttp2yet.com/

Matteo Varvello[1], Kyle Schomp[2], David Naylor[3], Jeremy Blackburn[1], Alessandro Finamore[1], and Konstantina Papagiannaki[1]

Telefónica Research[1], Case Western Reserve University[2], Carnegie Mellon University[3]

Abstract. Version 2 of the Hypertext Transfer Protocol (HTTP/2) was finalized in May 2015 as RFC 7540. It addresses well-known problems with HTTP/1.1 (e.g., head of line blocking and redundant headers) and introduces new features (e.g., server push and content priority). Though HTTP/2 is designed to be the future of the web, it remains unclear whether the web will—or should—hop on board. To shed light on this question, we built a measurement platform that monitors HTTP/2 adoption and performance across the Alexa top 1 million websites on a daily basis. Our system is live and up-to-date results can be viewed at [1]. In this paper, we report findings from an 11 month measurement campaign (November 2014 – October 2015). As of October 2015, we find 68,000 websites *reporting* HTTP/2 support, of which about 10,000 *actually* serve content with it. Unsurprisingly, popular sites are quicker to adopt HTTP/2 and 31% of the Alexa top 100 already support it. For the most part, websites do not change as they move from HTTP/1.1 to HTTP/2; current web development practices like inlining and domain sharding are still present. Contrary to previous results, we find that these practices make HTTP/2 more resilient to losses and jitter. In all, we find that 80% of websites supporting HTTP/2 experience a decrease in page load time compared with HTTP/1.1 and the decrease grows in mobile networks.

1 Introduction

HTTP/2 (H2 for short) is the new version of HTTP, expected to replace version 1.1 (H1), which was standardized in 1999. H2 promises to make the web faster and more efficient by compressing headers, introducing server push, fixing the head of line blocking issue, and loading page elements in parallel over a single TCP connection (cf. Section 2). Although the standard does not require encrypting H2 connections with Transport Layer Security (TLS), the major browser vendors currently only support encrypted H2 [19].

While on paper H2 represents the future of the web, it is unclear whether its adoption will face a struggle similar to IPv6. As discussed in [5], the adoption of a new protocol largely depends on the ratio between its benefits and its costs. Modern websites are already designed to deal with H1's inefficiencies, employing hacks like spriting, inlining, and domain sharding [18]. While H2 would remove the need for such hacks, in theory simplifying web development, given their widespread use it is unclear how much H2 can improve performance over H1. Furthermore, it is unclear how these practices will affect H2 performance (which is crucial, since web developers cannot rebuild their sites overnight nor are they likely to maintain two versions until H1 disappears).

Motivated by these uncertainties, in this work we build a measurement platform that monitors the adoption and performance of H2. Using machines on PlanetLab [3] and in our labs in Spain and the U.S., we probe the top 1 million Alexa websites each day to

see which support H2. For those that do, we note which features they use and measure performance with H1 and H2. Results are published daily at [1].

This paper reports findings from an 11-month measurement campaign, from November 2014 until October 2015 (cf. Section 4). As of October 2015, we find 68,000 websites *reporting* H2 support, of which only 10,000 *actually* serve website content over H2. NGINX, a popular web server implementation, currently powers 71.7% of the working H2 websites, with LiteSpeed following at 13.7% (in contrast to the 98% they claim[1]). Our results also show that sites that have deployed H2 have not significantly altered their content; classic H1 hacks are still used in the H2 version. For example, inlining (putting CSS styles and JavaScript code directly in HTML) is still widely used, reducing caching benefits. The same is true of domain sharding (spreading web objects across multiple domains), causing H2 to use more TCP connections than necessary. In terms of page load time, for 80% of the websites we measured an average reduction in page load time of 300 and 560 ms when accessed from a wired connection, respectively from Europe and the USA, 800 ms from a European 4G connection, and 1.6 seconds from a European 3G connection. The observed H2 benefits for mobile contradict previous studies; our analysis suggests that domain sharding, whether intentional or not, triggers the usage of several TCP connections, making H2 more resilient to losses and jitter typical of mobile networks.

2 Background and Related Work

H1 is an *ASCII* protocol that allows a client to request/submit content from/to a server. H1 is mostly used to fetch web pages, where clients request *objects* from a server and the resulting response is serialized over a persistent TCP connection. H1 provides pipelining to request multiple objects over the same TCP connection, but the benefits are limited since servers must respond to requests in order. Thus, an early request for a large object can delay all subsequent pipelined requests (*head of line blocking*). Clients mitigate this by opening several concurrent TCP connections to the server, which incurs additional overhead (TCP state on the server, TCP handshake latency, and TLS session setup in the case of HTTPS [13]). Accordingly, browsers limit the number of simultaneous connections to each domain (e.g., 6 in Chrome and 15 in Firefox [22]). Web developers have responded to this limitation with *domain sharding*, where content is distributed across multiple domains, circumventing the per-domain connection limit. Finally, H1 requires the explicit transmission of headers on a per request/response basis. Therefore, common headers (e.g., server version) are retransmitted with each object—particularly wasteful for pages with many small objects.

SPDY and H2 SPDY is Google's update to H1. It is binary rather than ASCII, enabling efficient parsing, lighter network footprint, and reducing susceptibility to security issues caused by unsanitized input strings. SPDY opens a single TCP connection to a domain and *multiplexes* requests and responses, called *streams*, over that connection, which reduces the number of TCP/TLS handshakes and the CPU load at the server. SPDY also introduces content *priority* (clients can load important objects like CSS and JavaScript earlier), *server push* (the server can push objects before the client requests them), and *header compression* (reduces redundant header transmission). H2 builds on

[1] https://www.litespeedtech.com/http2-ready—To their credit, another 27,000 websites powered by LiteSpeed redirect to an error page that loads over H2.

SPDY, making only relatively small changes. For example, H2 uses HPACK [16] for header compression, eliminating SPDY vulnerability to the "crime" attack [12].

NPN and ALPN Since SPDY, H1, and H2 all use TLS over port 443, port number is no longer sufficient to indicate to web servers which application protocol the client wants to use. The Next Protocol Negotiation (NPN) [4] is a TLS extension developed by Google as part of its SPDY effort. During the TLS handshake, the server provides a list of supported application protocols; the client then chooses the protocol to use and communicates it to the server via an encrypted message. Application Layer Protocol Negotiation (ALPN) [20] is a revised version of NPN standardized by the IETF. In ALPN, the client sends which application protocols it supports to the server, ordered by priority. The server selects the protocol to use based on the protocols it supports and the client priority; next, it returns the selected protocol to the client via a plain text message.

Related Work Previous work mostly investigate SPDY performance [10][8][11][15]; to the best of our knowledge, [17] is the only work previous to ours focusing on H2. Although the results of these studies are mostly contradictory, they converge on reporting poor SPDY (and H2) performance on mobile networks.

Erman et al. [7] measure page load time for the top 20 Alexa websites via SPDY and H1 proxies in 3G. They find that SPDY performs poorly in mobile networks since TCP interprets cellular losses and jitter as congestion, causing unnecessary backoffs. Since SPDY uses fewer TCP connections than H1, its performance suffers more.

Xiao et al. [23] introduce new measurement techniques to provide a more robust characterization of SPDY. They show that, in absence of browser dependencies and computation, SPDY tends to outperform H1; however, the gains are reduced when dependencies and computation are factored back in (with the caveat that server push can squeeze additional performance gains from SPDY).

De Saxcè et al. extend this analysis to H2 [17]. Using the top 20 Alexa websites, they investigate H2 performance under various network delay, bandwidth, and loss using an open-source client and server. Their results confirm those for SPDY in [23]. Unfortunately, by serving clones of the websites from their own test server, they ignore the impact of important real-world website properties like domain sharding.

Our aim is to take the next step in characterizing H2 performance. Our measurements improve prior art in five ways: (1) we target more websites (1000s as opposed to 10s or 100s); (2) we measure real servers from real networks (wired, 3G, and 4G); (3) we test real websites, not clones or synthetic traces; (4) we build on Chrome reliability to develop an accurate performance estimation tool; (5) we also study adoption and website structure trends.

3 Measurement Platform

This section describes our measurement platform. We start by summarizing a set of tools we have deployed, and then explain how we use them together to monitor H2 deployment and performance.

prober is a lightweight bash script that identifies which application protocols a website announces. `Prober` uses OpenSSL [14] to attempt ALPN and NPN negotiations and returns either the list of protocols announced by the server or failure. Next, `prober` checks for H2 cleartext (H2C) support—that is, H2 without TLS—by including an UPGRADE header in an H1 request.

H2-lite is a lightweight client that attempts to download only the root object of a website using H2. `H2-lite` uses the Node.js [2] H2 library [9]. `H2-lite` follows HTTP redirects to obtain the root object and reports any protocol errors encountered along the way. `H2-lite` also identifies sites with certificate problems, e.g., self- signed certificates, mismatches between hostname and common name, or expired/revoked certificates.
chrome-loader is a Python tool that loads pages using Chrome. It extracts object sizes and timing information using chrome-har-capturer [6]. `Chrome-loader` can instruct Chrome to use either H1 or SPDY/H2 (Chrome does not allow separate control over SPDY and H2). However, using Chrome's remote debugging protocol, `chrome-loader` reports which protocol was used to retrieve each individual object in a page.

We now describe our measurement platform in detail. It consists of a single *master* and many *workers*; the master issues crawl requests to the workers, which are deployed on both PlanetLab [3] and machines in our labs (U.S. and Spain). We use PlanetLab for simple measurements at a large scale and our lab machines for more complex measurements at a smaller scale and where machine reliability is important. The master constantly monitors PlanetLab to identify a pool of candidate machines (at least 500 MB of free memory, CPU load under 30%, and no network connectivity issues). We collect measurements in three phases:
Phase I: It discovers, daily, which protocols are supported by the top 1 million Alexa websites. First, the master launches an instance of the `prober` on each PlanetLab worker. The worker is then assigned a unique 100-website list to probe. When it finishes, it reports results to the master and obtains a new list if one is available. This approach ensures load balancing among heterogeneous workers allowing faster workers to complete more tasks. To deal with slow workers, the master re-assigns uncompleted tasks to new workers after a timeout T (set to the average task completion time across workers). Phase I terminates when the tracker has a complete set of results.
Phase II: It verifies, daily, whether the sites that reported H2 support in Phase I *actually* serve content over H2. After Phase I, the master launches several instances of `h2-lite` and, as above, it dynamically assigns each 100 sites that reported H2 support in Phase I. Because the H2 library requires more up-to-date software than is available on PlanetLab, we run `h2-lite` on 4 machines under our control, 2 in Barcelona (Spain) and 2 in Cleveland (U.S.). When Phase II terminates, the master has a list of sites that *actually serve* content using H2.
Phase III: It fetches both the H1 and H2 version of websites that serve content via H2 using multiple network locations and access network types (e.g., fiber and 4G). The master is responsible for selecting the machines to be used and instructing them which websites to test. The master uses one of three strategies: (1) *regular*, where each network location with fiber access is weekly instructed to test all H2 websites identified by Phase II; (2) *lazy*, the same as the regular strategy but a website is tested only if one of these conditions is met: (a) it is a new website that recently adopted H2, (b) its content significantly changed from the last time it was tested, or (c) a timeout elapsed since the last test; (3) *mobile*, where only mobile-enabled locations are selected, and a subset of websites are tested based on their Alexa popularity.

To test a website, we fetch it 5 times with H1 and 5 times with either SPDY or H2 (as discussed above, Chrome does not provide fine-grained control between SPDY and H2). Fetches are run sequentially to limit the impact of network load. While testing a website, we run a background ping process to collect statistics about network latency and packet loss. For the mobile strategy, we force Chrome to report a mobile user-agent to the server, which may respond with a mobile version of the website. Before the five

trials, an initial "primer" load is performed. This primer has a double purpose: (1) test whether a website significantly changed its content since the last time it was tested (used by the lazy strategy), and (2) ensure that DNS entries are cached at the ISP's DNS resolver before the first real trial (to prevent a cache miss on the first trial from skewing the load time results). Local content and DNS caches are disabled and each request carries a `cache-control` header instructing network caches not to respond.

We currently have Phase III workers in three different locations: Barcelona (Spain), Cleveland (USA), and Pittsburgh (USA). Each location consists of three machines with fiber connectivity. In addition, machines in Spain are connected to Android phones via USB tethering; each phone is configured to use either 3G or 4G. Because each SIM card has a 4 GB monthly limit, a maximum of about 200 websites—5 trials per protocol plus the primer—can be tested before exhausting the available plan. We ran Phase III with the regular strategy from November 2014 to September 2015, when the widespread deployment of H2 made it impossible to run Phase III weekly without additional machines. Thus, we switched to the lazy strategy. Finally, we ran Phase III with the mobile strategy only once in October 2015. We plan to run the mobile strategy once per month, as constrained by the mobile data plan.

4 Results

This section presents and analyzes the data collected using our measurement platform between November 10th, 2014 and October 16th, 2015. We invite the reader to access fresh data and analysis at [1].

4.1 Adoption

We recorded protocol support announced via ALPN/NPN for 11 months, during which time we saw support for 44 protocols (34 of which were versions of SPDY). Table 1 summarizes the evolution over time of the three most popular protocols, namely H1, SPDY 3.1, and H2; in addition, we also report NPN, ALPN, and H2C support when available. Each percentage is the maximum observed during the month; note that announce rates for H1 are underestimated since H1 sites are not required to use NPN/ALPN. The table also reports the total number of probed sites. From November 2014 to August 2015, we use the Alexa Top 1 Million list fetched on November 10th, 2014. Beginning in September 2015, we merge the current Alexa list with our own each day, constantly expanding the pool of sites we query (1.8 M as of October 16th 2015).

Overview From November 2014 through August 2015, Table 1 shows mostly constant H1 and SPDY announce rates, with the exception of a large drop in May 2015 (2.2 and 2.5 percentage points respectively.[2]) By contrast, the H2 announce rate grows by 50%—from 1.4% to 2.2%—with most of the growth occurring after May 2015, when H2 was standardized. As we start expanding the pool of sites that we probe (September 2015), H1 and SPDY announce rates grow slowly (SPDY grows from 5.1% in July to 5.7% in October), while H2 announce rates grow more quickly (2.2% in July to 3.7% in October). Interestingly, we measured a 0.1% drop in SPDY in October (about 2,000 sites) and a 0.6% increase in H2 (about 10,000 sites). If the trend continues, H2 support will overtake SPDY in the near future.

[2] We did not find a public explanation of this drop, but we verified it was not measurement error.

	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct
HTTP/1.1	9.8%	10.2%	10.5%	10.8%	11.2%	11.3%	9.1%	9.7%	10.8%	11.4%	12.1%	12.1%
SPDY/3.1	5.8%	5.9%	6.0%	6.2%	6.3%	6.3%	3.7%	4.1%	5.1%	5.6%	5.8%	5.7%
HTTP/2	1.4%	1.4%	1.3%	1.3%	1.3%	1.3%	1.2%	1.4%	2.2%	2.7%	3.0%	3.6%
NPN	9.8%	10.2%	10.5%	10.8%	11.2%	11.3%	9.1%	9.7%	10.8%	11.4%	12.1%	12.1%
ALPN	–	–	–	–	–	–	–	–	–	–	5.1%	5.1%
H2C	0%	0%	0%	0%	0%	0.006%	0.06%	0.2%	1.04%	1.20%	1.38%	1.37%
Tot. sites	1M	1M	1M	1M	1M	1M	1M	1M	1M	1.4M	1.7M	1.8M

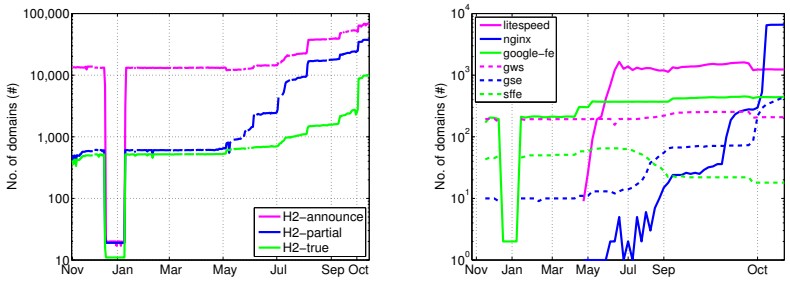
Table 1. Protocol support from Nov. 2014–Oct. 2015 as percentage of top 1 million Alexa sites.

NPN, ALPN, and H2C As of October 2015, 50% of the sites supporting NPN also support ALPN. We added ALPN support to our measurement platform in September 2015; previous manual ALPN tests showed less than 0.5% adoption. Although not shown in the table, as of October 2015 there are already 1,200 websites supporting ALPN but not NPN, 20% more than in September. H2C announcement also grows quickly, reaching 1.37% in October 2015 (about 22,000 sites). However, our UPGRADE header tests found only 20–30 sites that actually support H2C.

H2 Adoption We now examine H2 adoption in detail. Figure 1(a) shows the evolution of the number of websites that (1) announce H2 support (H2-announce), (2) respond using H2, even just to return an error page or redirect (H2-partial), and (3) actually serve page content with H2 (H2-true, cf. Section 3).

Figure 1(a) shows the substantial growth of H2 adoption after its standardization (May 2015). The Figure also shows a large difference between "announced" (H2-announce) and "actual" H2 support (H2-partial and H2-true). However, while between November 2014 and May 2015 only 5% of the websites announcing H2 support actually responded using H2 (H2-partial), this increases to 55% as of October 2015. We also see that before May 2015, most websites serving content with H2 are well configured and properly working (i.e., the gap between H2-partial and H2-true is small). After May, the gap widens considerably: most new websites supporting H2 show some misconfiguration, like redirecting to an error page. This trend continues until October 2015 when the gap between H2-partial and H2-true is suddenly reduced.

Web Server Implementations In order to further understand the different levels of H2 support noted above, Figure 1(b) plots the popularity of the web server implementations powering the H2-true websites. Note that the resolution of this data is weekly until September 2015 and daily thereafter. For visibility, Figure 1(b) shows only the 6 most popular web-server implementations (out of the 53 encountered). Figure 1(b) shows that, between October 2014 and May 2015, most H2-true websites are powered by Google web-servers (google_frontend, gws, gse and sffe). Then, in May 2015, two new H2 implementations debut: LiteSpeed and NGINX. Their adoption rates are very different; while LiteSpeed websites are quickly updated—LiteSpeed usage grows from 10 websites to 2,000 within two months—NGINX websites upgrade more slowly. The combination of fast adoption and the fact that most LiteSpeed websites are H2-partial and not H2-true suggests that LiteSpeed is used for virtual hosting more than NGINX. Despite lower overall numbers, NGINX currently powers more than 7,000 H2-true websites (71.7% of all H2-true websites) compared to LiteSpeed's 1,300.



(a) H2-announce, H2-partial and H2-true.

(b) Web server implementations.

Fig. 1. H2 adoption over time. November 2014 – October 2015.

Notable Events Figure 1(a) shows a few events that are worth noting. First, the number of servers supporting H2 drastically drops over a period of four days (December 14th-18th). This is because Google, the primary adopter of H2 at that time, disabled H2 support on their servers (cf. Figure 1(b)) due to an issue with SDCH compressed content in Chromium.[3] It takes a month for the problem to be fixed, and H2 support is re-enabled again over four days (January 11th-15th), likely reflecting a specific roll-out policy for server configuration changes. The next interesting event happens between May and June 2015, when the number of H2-partial websites—though they are mostly error pages—doubles from 600 to 1,200. This spike is due to Hawk Host, a virtual web hosting provider, updating its servers with the latest version of LiteSpeed, which supports H2 (cf. Figure 1(b)).[4] Figure 1(a) also shows a sudden increase in H2 support on October 3rd, caused by NGINX's footprint growing 40% (from 25,000 to 35,000 websites) thanks to WordPress updating to the latest NGINX release.[5] Finally, note that the spikes in August and September are due to the extension of the pool of websites we query (cf. Table 1).

Takeaway: *The H2 adoption trend reflects the broader Internet ecosystem, where most websites do not control their own technology stacks. There are a few big players responsible for the spread of new technologies; the average website owners might not even know they are serving content over H2.*

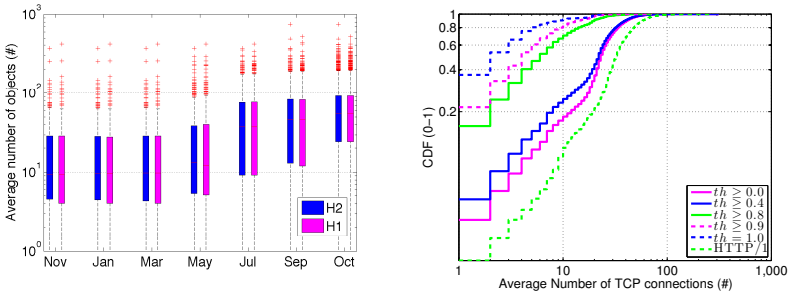
4.2 Website Structure

Number of objects Modern web pages are usually composed of several embedded objects, including images, JavaScripts, and style sheets. During our 11-month measurement campaign, we find no indication of a difference in the number of objects between pages served with H1 and H2 (cf. Figure 2(a)). To verify this quantitatively, we compute the cosine similarity of the object counts for H1 and H2 every week and find values ranging from 0.96 to 0.98. Thus, at a macroscopic level, there is no difference in the content composition between the two protocols. Fig. 2(a) also shows that from November 2014 to May 2015, pages served with H2 are mostly small pages, i.e., 50% have

[3] https://lists.w3.org/Archives/Public/ietf- http-wg/2014OctDec/0960.html

[4] http://blog.hawkhost.com/2015/07/13/http2-more-now- available-at-hawk-host-via- litespeed-5-0/

[5] NGINX 1.9.5 (September 22nd) was sponsored by Automattic, the creators of WordPress. http://nginx.org/en/CHANGES



(a) Distribution of the number of objects per website. Nov. 2014-Oct. 2015

(b) CDF of number of connections per website as a function of th. Oct. 6th-13th

Fig. 2. Content analysis and delivery (Cleveland).

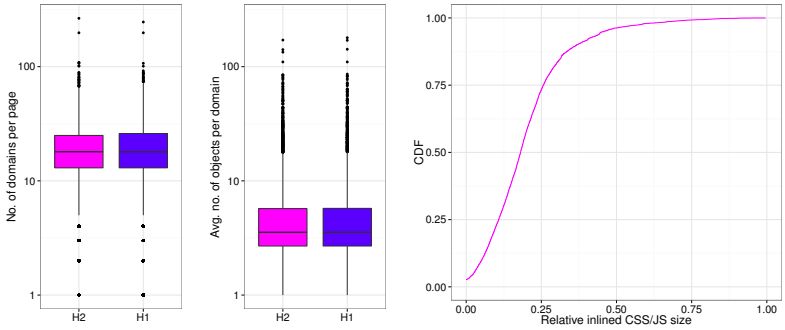
10 objects or less. After May 2015, more complex websites started to support H2; for example, in October 2015, 50% of websites have about 60 objects, which is in line with the average number of objects per webpage reported in [21].

The objects embedded within a web page often reside on another webserver and the browser must open new TCP connections independent of the application protocol in use. In addition, the browser must renegotiate which application protocol to use for each new TCP connection. It follows that even if a website supports H2, it is possible that only a fraction of its objects are delivered using H2. Although not shown due to space limitations, we find that half of the websites actually serve about 50% of their content using H2. Only 10% of the websites serve 100% of their objects using H2; these websites contain fewer objects and have an overall smaller size—400 KB on average compared to 1.5MB for all websites.

Takeaway: *There is no significant difference in object composition between the H1 and H2 versions of a website. Also, 90% of the websites still leverage H1 to serve a portion of their content.*

Number of connections We now investigate the number of TCP connections used to load a website using H1 versus H2. Motivated by the previous result, we differentiate websites based on how many of their objects are served with H2. Figure 2(b) shows the Cumulative Distribution Function (CDF) of the number of TCP connections per website as a function of *th*, or the fraction of the webpage's objects served with H2. In case of H1 we show a single curve as objects are always served with H1. The data for this plot was collected from our machines in Cleveland between October 6th-13th (8,492 distinct websites).

We first focus on the H1 curve and the H2 curve obtained with $th = 0$, where no websites are filtered. On average, H2 requires half as many TCP connections as H1; a few websites can even be served via a single TCP connection, but there are extreme cases where up to 100 connections might be required. As th increases, the required number of TCP connections decreases; this is intuitive since H1 and H2 cannot share the same TCP connection. For $th = 1$, all objects are served with H2, up to 40% of the websites only need a single TCP connection indicating that the websites content is hosted entirely on a single webserver; still, the remaining 60% need up to 20 TCP connections. This is a consequence of the way web content is organized today, where objects might reside on 3rd party domains. We further analyze this phenomenon next.



(a) Domains per page. (b) Objects per domain. (c) Relative Inline CSS/JSS size.

Fig. 3. Content analysis and delivery.

Takeaway: *H2 does not currently succeed in serving a page using a single TCP connection: 50% (4,300) of the websites using H2 today use at least 20 TCP connections.*

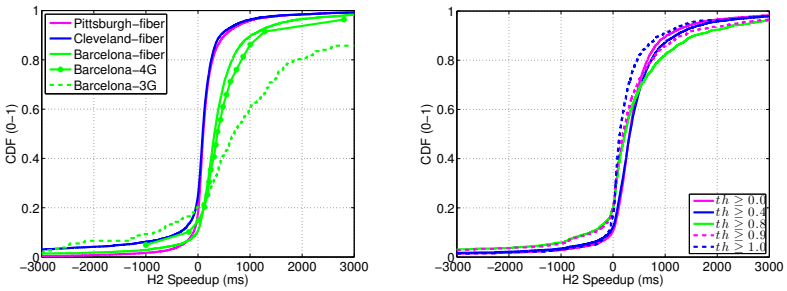
Page composition Websites today try to optimize delivery using several techniques like *sharding*, *inlining*, *spriting* and *concatenation* (cf. Section 1 and [18]). We see no straightforward way to measure spriting and concatenation, so in this work we only focus on sharding and inlining. Figure 3 summarizes the results of this analysis.

Focusing first on domain sharding, Figure 3(a) reports the distribution of the number of unique domains per website. The distributions for H1 and H2 are essentially the same with a median of 18 domains per page each, and outliers of $\sim$250 unique domains on a single page. This finding is related to the number of TCP connections opened per page (cf. Figure 2(b)). While H2 allows multiplexing object requests over a single TCP connection, most web pages embed objects from many different domains, thus forcing the use of a larger number of TCP connections. There are two plausible explanations for this: (1) web masters are deliberately sharding, and (2) it is just a natural consequence of deep linking (as opposed to re- hosting) practices that are common across the web.

Next, Figure 3(b) plots the distribution of the average number of objects per domain per website. We again find that there is no meaningful difference between H1 and H2: most domains are used for relatively few objects (median = $\{3.5, 3.5\}$, mean = $\{5.5, 5.4\}$, top 10th-percentile of $\{9.7, 9.6\}$, and a max of $\{179, 171\}$ for H1 and H2 respectively). This *is* further evidence of sharding-esque behavior since web page objects are dispersed over a large number of domains.

Finally, Figure 3(c) plots the size of inlined CSS/JS relative to the total size of the main HTML for each website; *i.e.,* what fraction of the total bytes making up an HTML document are from inlined CSS/JS. We find that there is a long tail where inlined CSS/JS makes up a considerable portion of the page: about 25% of websites' page contents is more than 25% inlined CSS/JS. With H1, inlining can help ensure the page loads faster by not requiring an extra TCP connection; however this is no longer an issue in H2, so these websites are potentially suffering a performance hit since inlined content cannot be cached separately by the client.

Takeaway: *Most websites exhibit H1 practices like domain sharding and inlining in H2. Sharding causes H2 websites to use more TCP connections than necessary and inlining may reduce the utility of caching.*



(a) Function of location and access. (b) Function of *th*.

Fig. 4. CDFs of H2 speedup.

4.3 Performance

Comparison to H1 First, we investigate H2 performance by comparing a website's *page load time (PLT)* when using H1 and H2. There is no standardized definition of PLT; from a user perspective, it is the time from when the user enters a URL in the browser to when the page is displayed. Similar to other studies [7], we approximate PLT as the time from when Chrome starts fetching the page to the firing of the JavaScript "onLoad" event; this event occurs once the page's embedded resources have been downloaded, but possibly before all objects loaded via scripts are downloaded. We define "H2 speedup" as the difference in PLT between the H1 and H2 versions of a website; speedup values > 0 indicate websites that are delivered faster with H2 than H1.

For this analysis, we leverage the data collected from multiple locations and access networks in the second week of October. From fiber connections, we target 8,492 websites with verified H2 support (cf. Fig. 1(a)) using three network locations: Barcelona (Spain), Pittsburgh (U.S.), and Cleveland (U.S.). Experiments using 3G/4G connections are performed at a single location (Barcelona) and we restrict the list to the 200 most popular websites, as ranked by Alexa, that also support H2.

Figure 4(a) shows the CDF of H2 speedup at each location, and access network. We start by focusing on fiber access; the figure shows positive speedup values for 75-85% of the websites depending upon network location. While Pittsburgh and Cleveland show very similar H2 speedup values (decreasing the likelihood of this being an anomalous observation), speedup values measured from Barcelona are 10-15% higher. Our latency estimations coupled with the measurements (cf. Section 3) show that, on average, the Barcelona experiments suffer an additional 14 ms in RTT since most target websites are located in the U.S. Likely, the longer RTT negatively impacts H1 more than H2 due to the handshakes required by the additional TCP connections in H1 (cf. Section 4.2).

Next, we focus on a single location (Barcelona) and different network access, namely fiber, 4G, and 3G ordered by decreasing "speed." Figure 4(a) shows that as the access speed decreases, the benefit of adopting H2 also increases: the average speedup grows from 560 ms on fiber, up to respectively 800 ms on 4G and 1.6 seconds on 3G. Latency measurements show that the median RTT more than doubles from fiber to 4G (46 up to 105 ms), and it grows by an additional 10% on 3G (117 ms). This additional latency again negatively impacts H1 more than H2. Figure 4(a) also shows that an additional 10% of websites see a performance degradation, negative speedup, over 3G compared with 4G and fiber. This results is due to the fact that for 3G and 4G we target Alexa's

200 most popular websites that also support H2, which tend to be landing pages and are simpler than the average website. In this case, the application protocol used has little impact as PLT is dominated by RTT.

Takeaway: *80% of the websites adopting H2 see an average page load time reduction of 500 ms (fiber) and 1.6 seconds (3G). The remaining 20% see an average page load time increase of 1 second (fiber) and 1.4 seconds (3G).*

Partial adoption We now analyze the impact of the fraction of the webpage served with H2 (*th*) on H2 speedup (cf. Fig. 2(a)). For this analysis, we focus on a single location and access network (Cleveland, fiber) since no significant differences arise across locations and access types. We find that for $th < 1$ there is no statistically significant difference between the curves[6] and most websites benefit from even partial H2 adoption. On average, the speedup reduces by 10% when we consider $th = 1$ which seems counter-intuitive. This again related to a simpler object composition of these websites for which H2 benefits result marginal.

Takeaway: *Even partial H2 adoption improves PLT. In fact, the decrease in PLT for websites using both H1 and H2 is often greater than that for pure H2 websites (though this is likely an artifact of the small subset of websites that are fully H2, which tend to be very simple).*

4.4 Discussion

In December 2015, a notable event shook our measurement infrastructure [1]. CloudFlare, a global CDN and DNS provider, enabled H2 for all its free customers.[7] This resulted in an additional 80,000 websites announcing H2 support (H2-announce) of which about 60,000 exhibit true support (H2-true). Note that CloudFlare uses an in-house version of NGINX, reported as cloudflare-nginx, which rapidly became the most popular web server implementation supporting H2.

Such rapid growth in H2-true websites affected Phase III's feasibility: at such scale, even the lazy strategy would require over a month to complete. Considering even further growth moving forward, we can either divide Phase III among our vantage points or sample the websites to test. We have temporarily suspended Phase III and are rethinking this component of our infrastructure. In addition, we are currently collaborating directly with CloudFlare to collect both server and client side measurements. We will soon report the outcome of an improved Phase III on [1] as well as in an extension of this work.

5 Conclusion

This work presents a measurement platform to monitor both adoption and performance of H2, the recently standardized update of H1. On a daily basis, our platform checks the top 1 million Alexa websites for which protocols they announce support. Next, it checks which websites *actually* support H2, and, once a week, tests their content structure and performance from multiple network locations. Once a month, a popular subset of these websites is also tested from 3G and 4G networks. Results are updated daily at [1]. In

[6] Two sample Kolmogorov-Smirnov tests provided no support to reject the null hypothesis.

[7] https://www.cloudflare.com/http2/

this paper, we report our initial findings from an 11 month measurement campaign, from November 2014 until October 2015. We find 68,000 websites already announcing H2 support, out of which 10,000 serve actual content, i.e., not an error page or redirect. An in-depth analysis of the content being served reveals that classic H1 hacks are still present with H2. In performance, we find that 80% of the websites load faster with H2 than H1. The average decrease in page load time is 300-560 ms from multiple locations with a fiber access, and up to 1.6 seconds from a European 3G connection.

References

1. Is the Web HTTP/2 Yet?, http://isthewebhttp2yet.com
2. Node.js, https://nodejs.org/
3. Planetlab, http://planet-lab.org
4. Adam Langley: TLS Next Protocol Negotiation, https://technotes.googlecode.com/git/nextprotoneg.html
5. Akhshabi, S., Dovrolis, C.: The Evolution of Layered Protocol Stacks Leads to an Hourglass-shaped Architecture. In: Proc. ACM SIGCOMM. Toronto, Canada (Aug 2011)
6. Andrea Cardaci: Chrome har capturer, https://github.com/cyrus-and/chrome-har-capturer
7. Erman, J., Gopalakrishnan, V., Jana, R., Ramakrishnan, K.: Towards a SPDYier Mobile Web? In: Proc. ACM CoNEXT. Santa Barbara, CA (Dec 2013)
8. G. White and J-F. Mule and D. Rice: Analysis of spdy and tcp initcwnd.., https://tools.ietf.org/html/draft-white-httpbis-spdy-analysis-00.
9. Gábor Molnár: node-http2, https://github.com/molnarg/node-http2
10. Google: Spdy whitepaper., http://www.chromium.org/spdy/spdy-whitepaper
11. Guy Podjarny: Not as spdy as you thought., http://www.guypo.com/not-as-spdy-as-you-thought/
12. J. Rizzo and T. Duong: The Crime Attack, in Ekoparty, 2012
13. Naylor, D., Finamore, A., Leontiadis, I., Grunenberger, Y., Mellia, M., Munafò, M., Papagiannaki, K., Steenkiste, P.: The Cost of the "S" in HTTPS. In: Proc. ACM CoNEXT. Sydney, Australia (Dec 2014)
14. OpenSSL: OpenSSL: The Open Source Toolkit for SSL/TLS, https://www.openssl.org/
15. Padhye, J., Nielsen, H.F.: A comparison of spdy and http performance. Tech. rep. (2012)
16. Roberto Peon and Herve Ruellan: Hpack - header compression for http/2, https://tools.ietf.org/html/draft-ietf-httpbis-header-compression-12
17. Saxcè, H.D., Oprescu, I., ChenSaamer, Y.: Is HTTP/2 Really Faster Than HTTP/1.1? In: Proc. IEEE Global Internet Symposium (GI). Hong Kong, CH (Apr 2014)
18. Stenberg, D.: HTTP2, background, the protocol, the implementations and the future, http://daniel.haxx.se/http2/http2-v1.9.pdf
19. Stenberg, D.: HTTP2 Explained, http://http2-explained.haxx.se/content/en/part5.html
20. Stephan Friedl, Andrei Popov, Adam Langley, Emile Stephan: Transport layer security (tls) application-layer protocol negotiation extension, https://tools.ietf.org/html/rfc7301
21. The http archive: http://httparchive.org
22. Tuan, N.A.: Maximum concurrent connections to the same domain for browsers, http://sgdev-blog.blogspot.com.es/2014/01/maximum-concurrent-connection-to-same.html
23. Wang, X.S., Balasubramanian, A., Krishnamurthy, A., Wetherall, D.: How speedy is spdy. In: Proc. NSDI. Seattle, WA (Apr 2014)