Push or Request: An Investigation of HTTP/2 Server Push for Improving Mobile Performance

Sanae Rosen, Bo Han*, Shuai Hao*, Z. Morley Mao, Feng Qian†
University of Michigan, †Indiana University, *AT&T Labs – Research
{sanae,zmao}@umich.edu, fengqian@indiana.edu, {bohan,haos}@research.att.com

ABSTRACT

In HTTP/1.1, it is necessary for the client to request an object (e.g. an image in a page) in order for the server to send it, even if the server knows in advance what the client will need. Server Push is a feature introduced in HTTP/2 that promises to improve page load times (PLT) by having the server push content to the browser in advance. In this paper, we investigate the benefits and challenges of using Server Push on mobile devices. We first examine whether pushing all content or just the CSS and Javascript files performs better, and find the former leads to much better web performance. Also, we find that sites making use of domain sharding or which otherwise have content divided across many servers do not benefit much from Server Push, a major challenge for Server Push going forward. Network performance characteristics also play a major role. Server Push is especially effective at improving performance at high loss rates (16% median PLT reduction with a 2% loss rate) and high latencies (14% PLT reduction with 100 ms latency), and has little benefit for high-speed Ethernet connections. This motivates its use on mobile devices, although we also find the limited processing power of these devices limits the benefits of Server Push. Server Push also offers modest energy benefits, with energy savings of 9% on LTE for one device. Overall, Server Push is a promising approach for improving web performance in mobile networks, but there are a number of challenges in achieving the full benefits of Server Push.

1. INTRODUCTION

Recently, HTTP/2, which promises improvements over HTTP/1.1 in browsing performance due to new features such as Server Push, has been standardized. In Server Push, the server uses its knowledge of the website's content to push objects before the client requests them. In this paper, we explore whether, and to what degree, Server Push leads to performance benefits, focusing on mobile networks where network conditions are dynamic and challenging.

Recent work has shown that improving web browsing performance is a complex problem. For instance, the performance benefits of SPDY have not been as great as expected [33], and are dependent on factors such as network performance. As most sites do not use Server Push, we take snapshots of 50 popular websites and test them locally on a server that supports Server Push. With this dataset, we find that Server Push offers far higher performance improvements on WiFi and LTE networks than Ethernet networks, and in fact on a high-speed Ethernet network, Server Push is not particularly useful. This motivates focusing on mobile devices.

To understand the impact of network characteristics, we perform experiments on mobile phones, as well as controlled experiments with artificially limited network conditions on wired networks where it is easier to explore the impact of limited network performance in a systematic way. We find that Server Push performance improvements are highly dependent on network features such as the loss rate and latency, and in some cases Server Push can even be detrimental. Individual websites also vary greatly in how they are impacted by the network, due to differences in loading patterns and the impact of rendering and computation. In absolute terms, savings are typically around a few hundred milliseconds, with some pages seeing benefits of seconds. In this paper, we explore these factors in depth and provide recommendations as to when Server Push would be most useful.

We examine how other factors can impact Server Push performance as well. Pushing the entire website, rather than a few Javascript or CSS files, is necessary to see substantial performance improvements. Websites split among different domains are a challenge for Server Push. We also find that the limited processing power of mobile phones can limit the benefits of Server Push, and that more computationally powerful devices would likely benefit more. Finally, we find that Server Push reduces energy consumption, offering modest energy reductions of 9% on one LTE network on average.

Our main contribution is that this is the first study focused on understanding Server Push performance (particularly on mobile devices) and how and when Server Push should be used. More specifically, our main findings are as follows:

- Server Push shows greater relative performance improvements on high-loss or high-latency networks, such as cellular and WiFi networks, as compared to typical wired networks, motivating its use on mobile devices.
- Pushing all content on a website is on average significantly more effective than pushing a handful of Javascript or CSS files.
- Domain sharding and content otherwise split among multiple servers is a significant impediment to Server Push's effectiveness.
- Server Push shows the best relative performance improvements with high latencies and loss rates, offering a median

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052574

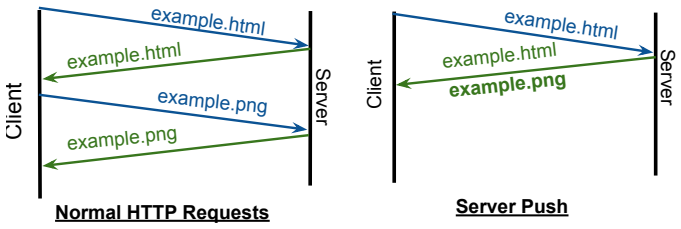

Figure 1: A simplified view of how Server Push results in performance benefits.

16% improvement in PLT with a 2% loss rate and a 14% improvement with a round trip time of 100 ms, but excessively poor network performance hurts Server Push. Some sites also see substantial benefits at low bandwidths.

- Server Push doesn't improve performance on every website, and so should be used judiciously, by testing Server Push performance before a widespread deployment.
- Server Push reduces LTE power consumption on mobile devices by about 9% and has no significant impact on WiFi power consumption.

2. BACKGROUND

HTTP/2 has been recently proposed as a replacement to HTTP/1.1, promising to offer better performance, and addressing some of the performance limitations of HTTP/1.1 [14]. One major feature of HTTP/2 is Server Push. As shown in Figure 1, in a somewhat simplified form, without Server Push it has been necessary for the browser to fetch the first HTML page and parse it before making requests for subsequent objects required by the first HTML page (and these objects may have further dependencies). Thus, two round trips are needed to load all the content for this example web page, and more round trips for more complex pages. However, the server in many cases likely already knows what content will be requested, and so the extra step of having the client request the object could be removed from the critical path.

This feature has been introduced into several major implementations of HTTP/2 [16], including nghttp2 [22], H2O [11], and, recently, Apache [2]. As it is part of the HTTP/2 standard [3], all implementations should eventually support it. Work by Varvello et al [31] showed that while HTTP/2 adoption is still rare, major players such as Google and hosting providers have adopted it and driven a slow but steady growth.

While this is the first work focusing on Server Push specifically, the performance benefits of HTTP/2 more broadly have been shown to be complex and varied [33], particularly on mobile devices [8]. Understanding how and when we can benefit from Server Push through a comprehensive measurement study would thus be valuable.

We believe this motivates examining how to use Server Push more effectively, as there is an opportunity to inform how web pages deploy Server Push in the future, and a better understanding of Server Push may lead to more widespread use.

3. DATASET AND METHODOLOGY

Since so few sites make use of Server Push, we conducted controlled experiments using mirrored sites hosted locally to test the impact of network performance, and ran other experiments to understand Server Push under a variety of circumstances.

Table 1: Summary of web page characteristics (out of a total of 50 sites).

	Min value	Median value	Max value
Num. objects	6	64.5	440
- Images	0	25.0	435
- Javascript	0	7.0	51
- CSS	1	1.0	7
Page size	46 Kb	1.86 Mb	10 Mb

We mirrored a total of 50 sites from the Alexa top 500 starting with the most popular, with essentially identical websites omitted: for instance, we included the main Google page, but not the pages of individual country versions of Google. Sites were copied using the Firefox Scrapbook extension [27], including all Javascript, and where necessary manually edited to remove popups and redirects that interfered with automated analysis, and to ensure where possible content is loaded locally. Sites that could not be hosted locally without substantial modifications were skipped. The mobile version of the page was used in our analysis. We summarize some statistics on the pages in Table 1.

These sites were hosted on a server using nghttp2 [22], which has a complete implementation of Server Push. Its library is now used by other servers, in particular Apache [2]. We collected at least 5 measurements for each experiment, randomizing the order in which the sites were visited. Except for when exploring how much content to push, all content was pushed as we found that to be the most effective approach, as we will show in the next section. Nghttp2 prioritizes the HTML page first, then the CSS files, then the Javascript files, then images and other content, according to their documentation[1]. Exploring the impact of non-standard prioritization approaches is left to future work.

On the client side, all experiments were run in an up-to-date Chrome browser. Our first set of experiments were carried out on two mobile devices: a Samsung S5 and (to compare with an older phone) a Samsung S3. Page load time measurements were collected using Chrome's debug interface accessible by plugging the phone into a computer and going to `chrome://inspect#devices`. The page load times are those listed by Chrome. All experiments were conducted with caching disabled, and the values of three measurements were averaged, rather than five, due to the manual effort involved.

Controlled experiments, where latency, loss rates and bandwidth were varied, were carried out over Ethernet on a desktop (except for the bandwidth experiments, which were performed over Ethernet with a laptop). This allowed us to be able to precisely control each of these variables. However, in most cases we are examining the sorts of network conditions that are more typical for mobile devices than for desktops or laptops on modern Ethernet connections. For these, page load time information was collected by a script which connects to Chrome's remote debug interface through a JSON API[2]. Latency and loss rates were varied using `tc`, a standard Linux utility that allows different network conditions to be emulated. Bandwidth was varied using the built-in Mac OS network emulation tools on a laptop. We also collected performance data for WiFi and tethering on the same laptop. We also collected

[1]https://nghttp2.org/blog/2014/04/27/how-dependency-based-prioritization-works/

[2]https://developer.chrome.com/devtools/docs/debugger-protocol

values on mobile devices. Unfortunately, the API was not available on mobile devices, but the page load times indicated in the GUI-based debug interface used for the mobile phone experiments is equivalent, and values were manually collected using this interface.

To calculate the energy overhead of Server Push, we made use of the power model in a recent paper [24], using the same parameters as in that paper. We collected tcpdump traces from our page loading experiments and calculated the power impact of Server Push with both the WiFi and cellular network.

4. WEB PERFORMANCE

There are a number of factors that impact Server Push performance that we examine. We first look at the impact of pushing differing amounts of content, and demonstrate that the current trend of distributing a web page's content across many domains introduces significant challenges for Server Push. We then examine how various network conditions can impact Server Push performance, demonstrating Server Push is mainly helpful for networks likely to experience poor network performance, such as mobile networks. We then examine differences between websites that do well or poorly with Server Push. We summarize the findings in this paper in Table 2.

4.1 Impact of Content Pushed

First, we examine existing sites that use Server Push and what they do, then determine a good strategy for determining what content to push.

Server Push is still rarely used in practice. We used the nghttp2 client to crawl the top 10,000 sites according to Alexa in September 2016. Our client attempted to connect using HTTP/2, and recorded whenever a PUSH_PROMISE header is seen which indicates the start of an object being pushed. We found five sites which used Server Push (plus one more that logged a Server Push request in our automated testing, but not when we examined it manually a few days later).

We looked at each of the five websites using Server Push in Google Chrome and manually examined what was pushed. They took a variety of approaches: one site pushed everything except dynamic content (`www.neobux.com`); other sites pushed just one Javascript file (`www.cloudflare.com`; `www.yoob.com`); another pushed its Javascript and CSS files (`www.kroger.com`) and one pushed a selection of Javascript and image files, but not all (`www.namepros.com`).

We next compared two strategies for pushing content: pushing only CSS and Javascript files, and pushing everything. For this experiment, we looked at our locally mirrored websites, as unfortunately we cannot set the push policy for real sites in the wild. We emulated a high-latency network (such as a cellular network) by adding 100 ms to the latency on an Ethernet connection, in addition to testing on a low latency network (1 ms ping, taking a total of about 30ms for a small object to load). The results are shown in Figure 2. We show the relative performance benefit — the ratio of the number of seconds saved due to Server Push to the original page load time. Clearly, Server Push performs a lot better when we push everything, and *we recommend pushing more content where possible*. Interestingly, pushing everything seems to matter more on higher latency networks: pushing just Javascript and CSS give similar results on the two networks, but pushing everything is much more helpful on the high latency network. Using nghttp2's hard-coded priorities, HTML was pushed first, then CSS, then Javascript, then all other content. We leave exploring other methods of prioritizing content to future work.

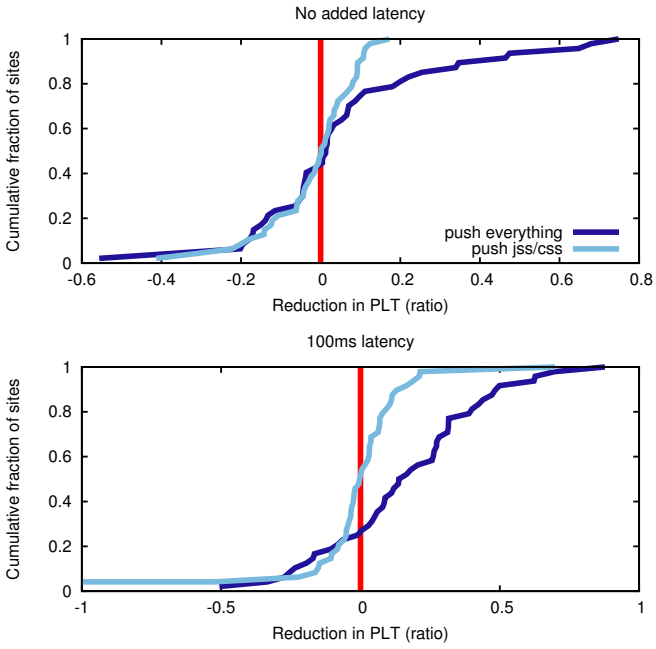

Figure 2: Pushing all content versus pushing only Javascript and CSS files. Both graphs show the ratio of the time saved due to Server Push to the original page load time.

There is one major complication in pushing everything. In practice, content is often split over several domains, whether due to third-party advertisement, domain sharding, or other reasons. To examine the impact of this problem, we went through each page and manually determined whether each object came from the same domain or a different domain. We then moved that content to another server, and didn't push that content. In the median case for the 50 sites, we moved more than 95% of the objects to another server. Most major websites host images and other content separately, although a few websites were mostly unchanged. We found almost no benefit from Server Push in this case: Only 25% showed any measurable benefit, and less than 15% showed more than a 10% performance improvement. Clearly, the way in which websites are architected today are a major problem for the deployment of Server Push, and which should be addressed in future work.

HTTP/2 promises to make domain sharding unnecessary [15], and it is generally recommended not to use domain sharding in HTTP/2 [23]. Recent work has also discussed that content on a website served from outside a CDN can cause substantial performance degradation [9], motivating further keeping content on one server wherever possible. However, in the short term it's unlikely that websites will be drastically re-architected.

4.2 Impact of the Network

To understand how network conditions impact Server Push, we vary network performance parameters in a controlled manner by adding latency, loss and limited bandwidth to an Ethernet connection.

First of all, what networks should we focus on, when deploying Server Push? We first examine several typical connections: the local LTE network, a home WiFi network, and the Ethernet connection in the lab. Typically, the LTE network experiences latencies of roughly 90 ms, the WiFi network of about 25 ms, and latencies of

Table 2: Summary of findings.

What to push	
Push as much content as possible, not just a few small files.	Fig. 2
Content divided across domains is a major problem.	§4.1
Server Push increases the loading time for the initial HTML object and so can harm performance.	Fig. 5
Network factors	
Highest improvement on WiFi and LTE for a given device; mobile devices are limited by their processing power.	Fig. 3
Higher improvement with high latencies (100ms).	Fig. 6
High loss rates harm performance improvement for some pages but others benefit between 0.5% and 2%	Fig. 7
High losses and latencies combined don't do well with Server Push.	Fig. 8
Server Push is slightly more beneficial with low bandwidth.	Fig. 9
Energy impact (§6)	
Server Push improves LTE energy consumption slightly (by about 9%).	Fig 15
Server Push has almost no impact on WiFi energy consumption.	Fig 15

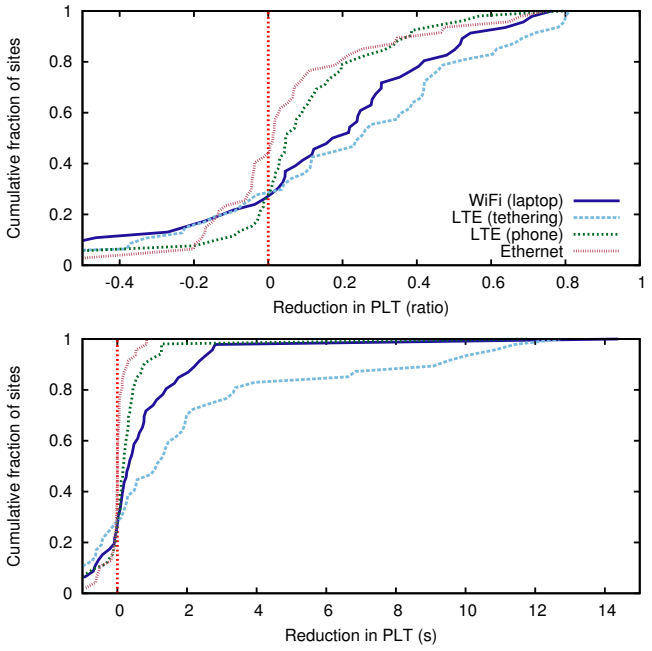

Figure 3: Server Push PLT savings for mobile websites on a variety of networks. The top graph shows the ratio of PLT savings to original PLT, and the bottom graph shows the absolute PLT improvement. Negative values cut off at -0.5 for the top graph and -1 for the bottom.

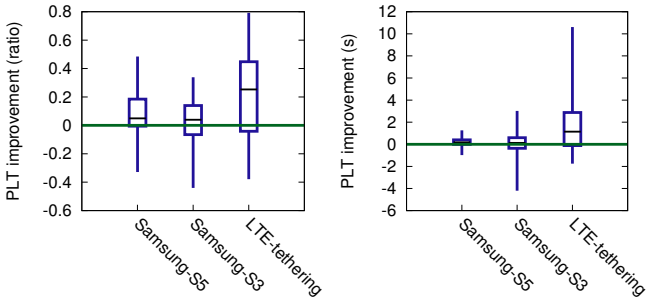

Figure 4: Impact of device processing power on Server Push. The left graph shows the ratio of the PLT savings to the original PLT, and the right graph shows the absolute PLT improvement in seconds.

about 1 ms for the Ethernet network (the server hosting the pages was on campus). The time to first byte for the first object with a 1 ms latency was around 30ms for a small object. As we show later in this section, even more typical latencies of up to 50 ms perform similarly over Ethernet.

We show the results in Fig. 3, where we show both the absolute improvement in seconds, and the ratio of the time saved due to Server Push to the initial page load time (PLT). Server Push can greatly improve performance — reducing PLT by up to 80% in the best case — but in many other cases, Server Push does not help. LTE and WiFi benefit more from Server Push than Ethernet does. For this reason, we believe it makes sense to focus on mobile devices when determining when and how to use Server Push.

However, on an actual mobile device on LTE, savings as a percent of the original PLT shrink, although they remain higher than a laptop using Ethernet. Recent work has found the lower processing power of mobile devices makes the loading time less network-dependent [20]. We still see performance improvements, though. Furthermore, despite this limitation, there are savings of hundreds of milliseconds on average, and in some cases seconds. Even saving 100 ms can have a high benefit [13].

Next, we tested a second mobile phone (a Samsung S3) on a different carrier (still LTE), to make sure our results are not specific to a particular device or network. We show the results in Figure 4. The two phones show very similar results, suggesting our results on the phone are at least somewhat representative. The older phone generally experiences larger absolute page load times. The laptop with tethering does better though; substantial improvements in processing power would be needed to see the full benefit of Server Push on mobile phones.

It is also apparent that the benefits of Server Push can be lower than zero, a phenomenon first mentioned by Wang et al [33]. We examine why Server Push does not show positive performance benefits in every case. Server Push tends to make the initial HTML file slower to load, sometimes quite substantially, as shown in Figure 5. This is the case even though nghttp2 sends HTML traffic with a

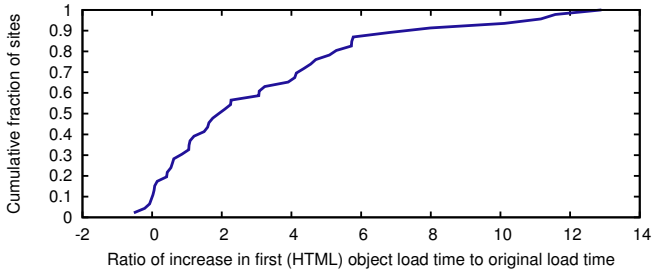

Figure 5: Relative increase in the loading time of the initial HTML object with Server Push: this increase leads to Server Push in some cases performing poorly overall.

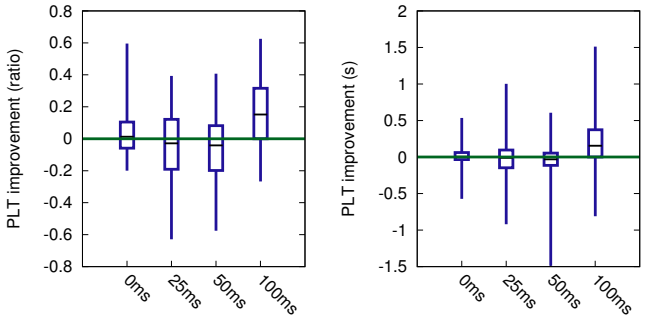

Figure 6: Impact of network latency on Server Push, with both the relative decrease in latency (left) and absolute decrease (right) shown. Latencies shown are from ping; at 0ms, a small object takes about 30ms to load including server processing etc.

higher priority — we seem to still see interference from other requests. For Server Push to be beneficial, it has to offer savings greater than this cost, which is why we only see benefits for some websites.

Next, we examine individual network performance factors, varying the bandwidth, latency and loss rate of an Ethernet connection while holding the other variables constant. We use Ethernet to measure the impact of each variable in a controlled manner, as WiFi or LTE are likely to show a wider range of latencies and losses while we run these experiments, making it harder to draw a firm conclusion.

Latency: Fig. 6 shows the results of varying the latency. There is a sudden jump in the performance improvement between 50ms and 100ms, where web pages are likely becoming more network-bound. At higher latencies, pages are more likely to become network-bound, although the effect of delaying the initial HTML object may be worsened.

Packet loss: The impact of packet loss is fairly substantial as well, as shown in Figure 7. Looking at the leftmost cluster in that figure, where there are both uplink and downlink losses, it can be seen that as loss rates increase, some websites perform substantially worse with Server Push as compared to not using Server Push. High loss rates of around 3% can also affect the initial download burst of content, exacerbating the problem of it slowing down the page load time. With slightly lower loss rates, however, for some pages Server Push is able to mask the slower resulting loading time of these pages. At about a 0.5% loss rate, websites almost consistently do better than with no loss. Also, sites that perform badly tend to perform worse at high loss rates.

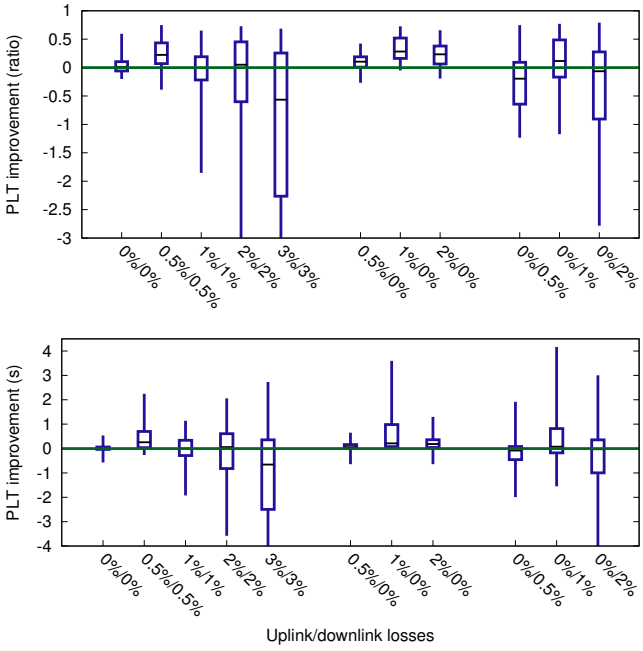

Figure 7: Impact of network packet loss on Server Push in relative (top) and absolute (bottom) terms. Labels indicate the uplink loss rate followed by the downlink loss rate. Negative values are cropped at -3 for the top graph and -4 for the bottom.

To understand why at very high loss rates performance benefits disappear, we examined some waterfall diagrams in Chrome, and found that some objects load slowly enough that loading objects somewhat earlier had a very minor effect, and also that the problem of the pushed objects slowing down the download time of the first few objects becomes a major factor. It is also known from prior work that HTTP/2 tends to perform poorly under high loss rates as there is a single TCP connection whose congestion window is affected by the losses [33].

We then look at uplink and downlink losses individually in the middle and the right hand side of the figure, and it's apparent that downlink losses hurt Server Push more than uplink losses: Server Push can mask uplink losses since fewer requests to the server are needed with Server Push, but downlink losses can affect the initial set of downloads that sometimes leads to a slower PLT with server push.

If we look at high loss rates and latencies combined, as in Figure 8, Server Push also doesn't perform well (values collected on 44 websites for these experiments only). In fact, we see decreasing performance with increasing latency when there's a substantial loss rate. Note how the distributions get much wider at high loss rates: pages that do poorly do a lot worse, but some pages that do well do a lot better.

In general, high latencies are more common on mobile networks than high loss rates.

Bandwidth: The amount of available bandwidth has an impact, as can be seen in Figure 9. For these values, a delay of 30ms was added as otherwise the impact of bandwidth is not very pronounced. As before, not all sites benefit when the bandwidth is low, and in fact the median value remains around zero, but when websites benefit from Server Push, they benefit more with low bandwidth.

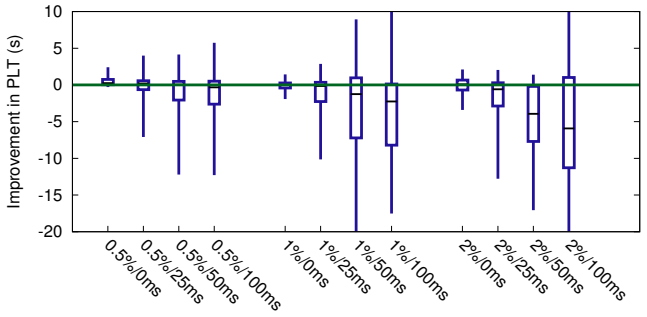

Figure 8: Impact of combined high latencies and loss rates. The loss rate is listed first (both directions), then the latency, as a percent and number of milliseconds, respectively. Extreme negative values are cropped.

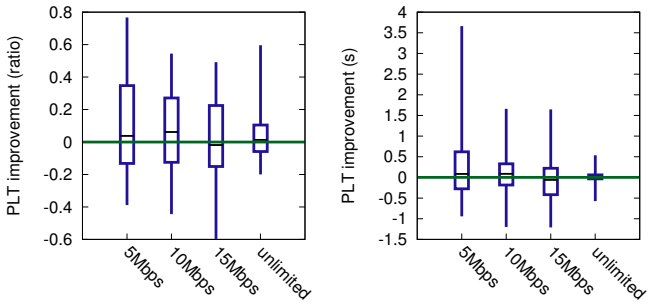

Figure 9: Impact of bandwidth at 30ms latency. The ratio of the performance savings to the loading time are shown in the left graph, and the absolute performance savings are shown in the right graph .

Overall, it appears that Server Push can mitigate network performance problems for some websites, but network performance problems also exacerbate the problem of the initial HTML object being delayed, and the latter issue can become the dominant factor when network performance is sufficiently bad.

4.3 Impact of the Web Page

We examined a number of web page metrics, and found that the one that best predicts if Server Push will show performance benefits is the time for the object after the initial HTML page to load. With a latency of 100 ms, a load time of that object of 125 ms or higher resulted in an average Server Push improvement of 27% (versus 13% overall). With loss rates of 0.5% to 2%, if the loading time is above 150 ms, there was an average performance improvement of 35%.

To understand why that is the case, we performed a controlled experiment: we created a page with a significant amount of Javascript computation, and with 5 images to load. We then varied the number of figures that loaded after the Javascript, and show the results in Figure 10. Content that loads after computation runs can be fetched early by Server Push in parallel with the computation, resulting in higher performance savings as the download time can be hidden behind the compute time. However, this network traffic has to have a substantial impact on the overall loading time to matter.

We also examined if the website size has an impact on Server Push performance. To do so, we examined loading artificial web-

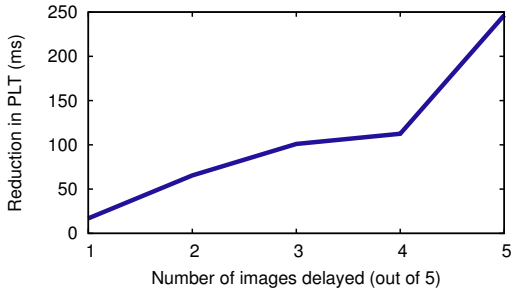

Figure 10: Relative benefits of pushing content for sites with a significant amount of computation, with 1-5 of 5 figures loading after the Javascript computation and the rest loading before.

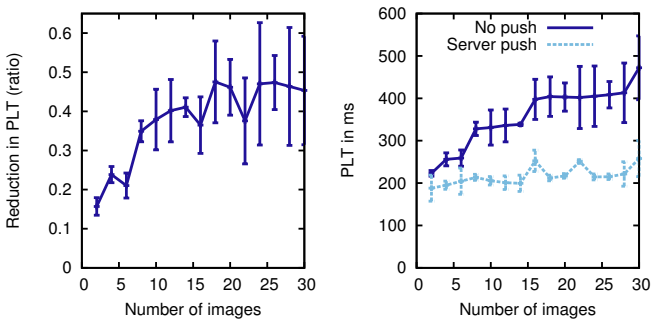

Figure 11: Impact of web page size on Server Push with a 100 ms delay and 10 Mbps bandwidth.

sites that were identical except for the number of images on the page. The results are shown in Figure 11. As we increase the number of images, the benefits of Server Push increase as it is able to reduce the resulting loading time increase. However, we did not find the size to consistently lead to better Server Push performance on real websites: other factors, such as the amount of computation and rendering, had more of an effect.

4.4 Summary

Overall, we've found that Server Push works best with wireless networks, although the performance limitations of mobile devices limit Server Push's performance. Even so, the relative performance improvements on wireless networks are higher than over Ethernet, and tends to be better when network conditions are poor. However, Server Push introduces a delay to the first HTML object, and so not all websites benefit from Server Push. Looking deeper into the network characteristics that lead to good Server Push performance, Server Push can mitigate the effects of network performance problems for many websites, but if network performance is too poor, pages become unable to benefit from Server Push.

5. CASE STUDIES

We next examine examples of real sites and how they load, in a variety of circumstances, to better understand how Server Push affects performance. These experiments were carried out on a phone over LTE, unless otherwise stated. Our plots show only the browser load times and not the push download times. The browser load time is the time for the browser to fetch and load a given object, from when the browser first makes the request to when it is fully loaded, extracted from the HTTP Archive (.har) file saved from the debug

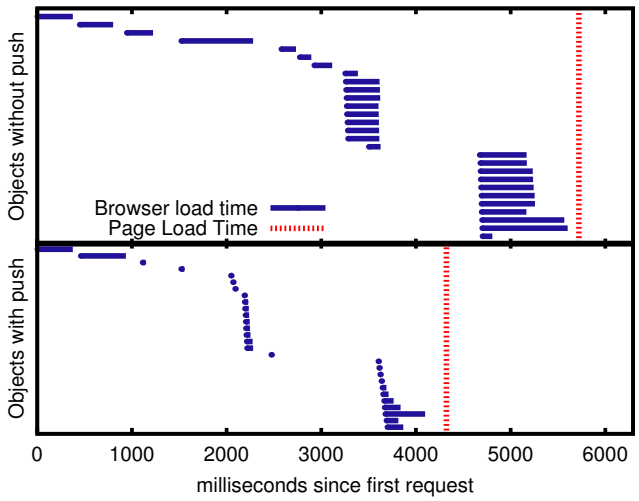

Figure 12: Waterfall diagram of loading the IKEA web site in a phone browser. The time for each successive object to load is shown as a horizontal line with a begin and end point corresponding to the first and last byte received. Objects are shown vertically in order of when they begin loading.

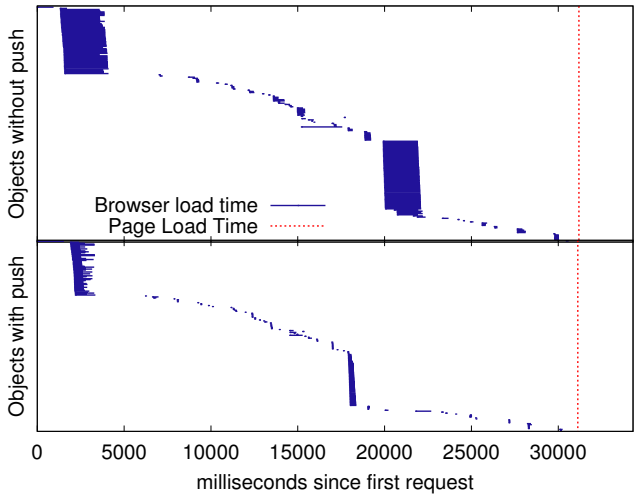

Figure 13: Waterfall diagram of loading the BBC website in a phone browser.

view in Google Chrome. When content is pushed, the browser load time for each object that is shown in the plots is much smaller, as the push time is not included in that value.

First, we show the mirrored IKEA page in Figure 12. This is a fairly straightforward case: loading happens over several stages, and especially after the first few objects, the loading time is shorter because Server Push has already delivered (most of) the content. Rendering and other computation does play a major role in the page load time, but reducing the network loading time helps substantially. Note that a significant amount of content is loaded after some computation, like in Figure 10.

The BBC website also loads in batches, but is more complex, and more time is spent in computation. We show the results in

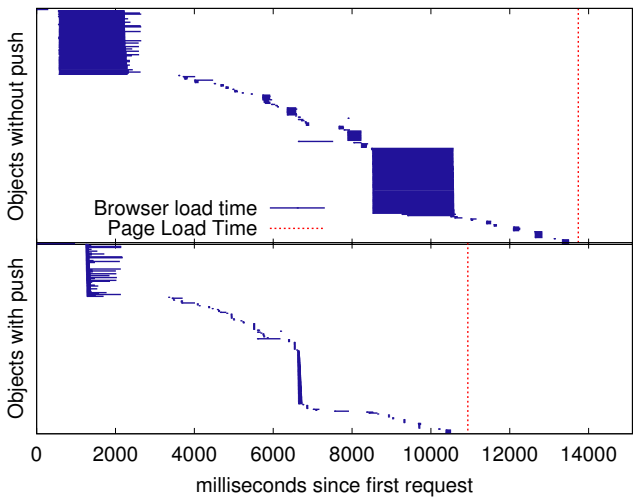

Figure 14: Waterfall diagram of loading the BBC website over WiFi on a laptop.

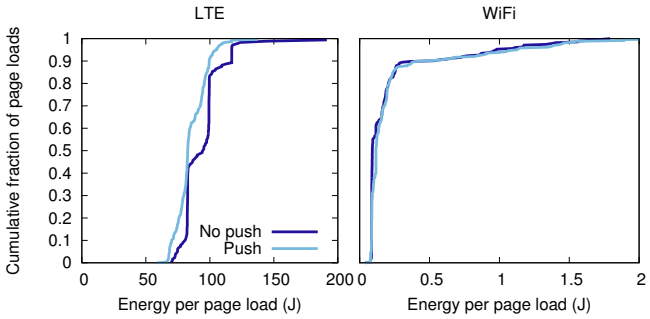

Figure 15: Radio energy trends for mobile devices. Server Push offers some savings for LTE only.

Figure 13. While it's apparent that the network loading time is reduced substantially, only the computation and rendering time is on the critical path in this case. Note that these results are on a mobile phone, where computation is more frequently on the critical path. With a laptop on WiFi, we see better performance for Server Push, as shown in Figure 14. Server Push is not able to reduce the loading time of the first burst of traffic, as there is too much in total to load. The last chunk of network traffic has been fetched by the time the browser requests it. Because of the last burst of network traffic being fetched, close to two seconds are shaved off, or a little under 20%.

6. ENERGY IMPACT OF SERVER PUSH

We next consider the impact on radio energy on mobile devices. Ideally, if Server Push reduces the time that the radio is active, by essentially "compacting" requests into one burst, then it should result in energy savings.

We collected packet traces over WiFi and LTE, and calculated the energy consumed by the radio based on the model given in a recent paper [24]. The results are shown in Figure 15. We found there to be modest but observable power savings with LTE, but not with WiFi. WiFi only keeps the radio awake briefly after data is sent, and so WiFi is less dependent on the distribution over time

Table 3: Summary of recommendations.

Test if websites actually benefit	§ 4.2	Easy
Push as much content as possible	Fig. 2	Easy
Host everything on one server	§ 4.1	Medium
Develop tools for 3rd-party content	§ 4.1	Hard
Tailor to different network conditions	§ 4.2	Hard

in which requests are made. With LTE, however, the radio stays active for some time after data is no longer sent. We see a 9% improvement on average with LTE.

7. DISCUSSION

In this paper, we found that the improvements due to Server Push are more pronounced under high loss or high latency conditions, as well as on wireless networks (WiFi and LTE). While less computationally powerful devices tend to benefit less from Server Push for a given set of network conditions, the fact that mobile devices are more likely to experience poor network conditions in the first place means these are good devices to target.

We have focused on the performance benefits of Server Push, but there are other obstacles to Server Push being effectively and commonly used. The first is the problem of client caching. Since the server doesn't know what the client has cached, there is potentially more overhead in terms of data wasted with Server Push. There have been various solutions proposed, however [12, 22]. The second challenge is dynamic content: we may not be able to determine what to push, and we leave predicting dynamic content to future work.

Also, we use the default order for pushing content, and don't explore alternate orderings based on, say, dependencies. There are a number of papers that have proposed systems to analyze web page loading dependencies [5, 21, 18]. An interesting direction for future work would be to explore and adapt these methods to further optimize loading time by avoiding pushing content unnecessarily.

Finally, we analyze mirrored websites, because there are almost no real websites using Server Push. It is possible, however, that as Server Push is deployed on production servers, other factors affecting Server Push will become apparent. We hope that our findings will motivate more sites to make use of Server Push.

We have focused on web browsing, but Server Push may be applicable to many other uses of HTTP. In particular, we have not examined mobile apps, which may be able to leverage Server Push even if they do not have the typical structure of an HTML page with images and other content embedded in it. We leave examining this to future work.

Recommendations: We summarize our recommendations in Table 3, in order of how practical they are to implement. Overall, we have found there are some fundamental challenges in making Server Push effective, and more work is likely needed to make it a solution for every website. However, there are some solutions that can be done today by website developers.

The most important takeaway from this paper is that you should *measure if Server Push helps your site* before putting time or resources into pursuing it. This could be done by setting up a test server with Server Push and simply measuring the loading time with and without Server Push with a mobile device under a range of conditions (or perhaps a range of mobile devices, if you want to be thorough). Also, websites should *push as much content as possible*. For some web pages, this would be possible. However, this may not be sufficient to make Server Push work for everyone.

In order to make pushing everything possible, the next step is to host everything on one server. This is recommended, in any case, for HTTP/2 [23]. This may not be practical for every website, however. There are likely more long-term solutions that would be harder to deploy. For instance, a mobile-specific proxy similar to Flywheel [1] which enables HTTP/2 as well could potentially be used. At least one HTTP/2 proxy exists that supports Server Push, which is intended to provide HTTP/2 for servers using other protocols [22]. These solutions would likely be more challenging to deploy in practice, though.

Future work could also explore methods of alerting third-party servers as to what to push, for instance by having small embedded objects that load early, thus informing those third-party servers that they should push content.

Finally, we found that the network characteristics of web pages make a major difference, and recommend that sites decide when to push content accordingly. This recommendation is likely the hardest to act upon. In the context of a mobile app, it might be possible to profile what network performance the average user sees, and decide whether to use Server Push accordingly. Somehow inferring network conditions to determine when to use Server Push would be an interesting direction for future research. In general, though, Server Push is more beneficial over wireless than modern, high-speed wired networks, hence our focus on mobile devices in this study.

Overall, we suggest that Server Push is ready to be used by some web pages today, but that a certain degree of effort on the part of website developers is likely required for many web pages, and there are ample future research directions to help Server Push reach its full potential.

8. RELATED WORK

Next generation web protocols: Recent work has found the performance benefits of these protocols are mixed. Varvello et al [31] found that the prevalence of HTTP/2 is small but rapidly growing, driven by a few key players, and that it offers performance benefits in the wild. "How Speedy is SPDY?" [33] observed that the performance impact of SPDY depends on many complex factors, and that Server Push gives performance improvements with high RTT. We examine the factors that impact Server Push performance in more depth. The impact of SPDY on mobile devices specifically has also been examined: work by Erman et al [8] found that SPDY does not consistently give performance benefits on cellular networks. Work by Carlucci et al [6] examines the performance of QUIC and finds mixed results. A study of SPDY performance by Elkhatib et al [7] finds that SPDY's performance as a whole is impacted by network performance and web infrastructure. Recent work by Zarifis et al [37] built and evaluated a model that can predict HTTP/2 performance from HTTP/1, and briefly examined the impact of Server Push, showing that it generally improves performance. Although some prior work has briefly touched on Server Push, we are the first to study Server Push specifically and in depth.

Understanding mobile and web performance: Understanding browsing performance more generally has been an area of recent interest. WProf [32] finds performance dependencies in browsers. Work by Netaji et al [20] finds that a major cause of slow mobile browsing is the computation overhead. Work by Qian et al [26] examines how existing websites are designed and the impact on performance and resource usage. Work by Wang et al [35] examines sources of delays in mobile web browsers. Work by Butkiewicz et al [4] examines how website complexity impacts performance.

WebProphet [18] predicts page load times from object dependencies. Work by Imh et al [17] also characterizes websites, and their finding that a substantial amount of content is cacheable reinforces the need for Server Push to account for caching. Work by Goel et al [10] examines the performance impact of using IPv6 on mobile devices and suggests some mechanisms for CDNs to ensure good network performance when transitioning to IPv6. Work by Nikravesh et al [25] examines longitudinal trends in mobile network performance generally.

Other recent papers examine network factors relevant to Server Push. Work by Sundaresan et al [30] investigates residential broadband, and determines that the round trip time is still a bottleneck. Work by Zaki et al [36] examines factors affecting network performance in a developing country, and finds that SPDY works particularly well. Work by Narayanan et al [19] observes that content is often poorly distributed among CDNs. They also observe that a significant percentage of content is served from CDNs, which may have implications for Server Push deployment by third parties.

Improving mobile and web performance: There has also been a lot of interest recently in building systems that improve browsing performance. Polaris [21] makes computation more efficient by better detecting dependencies and scheduling requests through a client-side scheduler. Parcel [29] splits browsing functionality between a proxy and mobile browser to improve performance, and Flywheel [1] is a compression proxy for mobile devices used in the wild which halves the size of mobile pages. Flexiweb [28] dynamically adapts proxy optimizations based on network conditions and website characteristics: this approach would likely be beneficial for Server Push as well. Klotski [5] examines automatically detecting dependencies and scheduling object downloads using a proxy. Shandian [34] optimizes the order and manner in which content is loaded, using a proxy.

9. CONCLUSION

Overall, we have found that Server Push can offer substantial performance benefits. Server Push shows the best relative improvements when latency or loss rates are high (bandwidth has a smaller impact) and on sites where objects are requested late in the loading process. Mobile networks are particularly suitable for Server Push, although the limited processing power of mobile devices reduces the benefits of Server Push, and Server Push is likely to become substantially more useful as mobile devices become more powerful. However, the way modern websites are constructed, with content divided across many servers, is a substantial problem. Furthermore, performance benefits vary greatly by website, and in some cases Server Push can be detrimental to performance, so it should be deployed only after verifying it will show performance benefits.

10. ACKNOWLEDGMENTS

We would like to thank our anonymous reviewers for their valuable comments. This research was supported in part by the NSF grants CNS-1566331 CNS-1059372, CNS-1345226, CNS-1629894, and CCF-1629347, as well as by an NSERC Canada PGS D scholarship.

11. REFERENCES

[1] V. Agababov, M. Buettner, V. Chudnovsky, M. Cogan, B. Greenstein, S. McDaniel, M. Piatek, C. Scott, M. Welsh, and B. Yin. Flywheel: Google's Data Compression Proxy for the Mobile Web. In *Proc. NSDI*, 2015.

[2] Apache Module mod_http2. `https://httpd.apache.org/docs/2.4/mod/mod_http2.html`.

[3] M. Belshe, R. Peon, and M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540.

[4] M. Butkiewicz, H. V. Madhyastha, and V. Sekar. Understanding Website Complexity: Measurements, Metrics, and Implications. In *Proc. ACM IMC*, 2011.

[5] M. Butkiewicz, D. Wang, Z. Wu, H. V. Madhyastha, and V. Sekar. Klotski: Reprioritizing Web Content to Improve User Experience on Mobile Devices. In *Proc. NSDI*, 2015.

[6] G. Carlucci, L. De Cicco, and S. Mascolo. HTTP over UDP: An Experimental Investigation of QUIC. Proc. ACM SAC, 2015.

[7] Y. Elkhatib, G. Tyson, and M. Welzl. Can SPDY really make the web faster? In *IFIP Networking*, 2014.

[8] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan. Towards a SPDY'Ier Mobile Web? In *CoNEXT*, 2013.

[9] U. Goel, M. Steiner, W. Na, M. P. Wittie, M. Flack, and S. Ludin. Are 3rd Parties Slowing Down the Mobile Web? In *Proc. S3 Workshop*, 2016.

[10] U. Goel, M. Steiner, M. P. Wittie, M. Flack, and S. Ludin. A Case for Faster Mobile Web in Cellular IPv6 Networks. In *Proc. ACM MobiCom*, 2016.

[11] H2O: The optimized HTTP/1.x, HTTP/2 server . `https://h2o.examp1e.net/index.html`.

[12] B. Han, S. Hao, and F. Qian. MetaPush: Cellular-Friendly Server Push For HTTP/2. In *AllThingsCellular*, 2015.

[13] Latency is Everywhere and it Costs You Sales - How to Crush it. `https://goo.gl/bRi5Xs`.

[14] HTTP/2. `https://http2.github.io/`.

[15] HTTP/2 FAQ. `https://http2.github.io/faq`.

[16] http2/http2-spec: Implementations. `https://github.com/http2/http2-spec/wiki/Implementations`.

[17] S. Ihm and V. S. Pai. Towards Understanding Modern Web Traffic. In *IMC*, 2011.

[18] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. Greenberg, and Y.-M. Wang. WebProphet: Automating Performance Prediction for Web Services. In *Proc. NSDI*, 2010.

[19] S. Narayanan, Y. Nam, A. Sivakumar, B. Chandrasekaran, B. Maggs, and S. Rao. Reducing Latency through Page-aware Management of Web Objects by Content Delivery Networks. In *Proc. ACM SIGMETRICS*, 2016.

[20] J. Nejati and A. Balasubramanian. An In-depth Study of Mobile Browser Performance. In *WWW*, 2016.

[21] R. Netravali, J. Mickens, and H. Balakrishnan. Polaris: Faster Page Loads Using Fine-grained Dependency Tracking. In *Proc. NSDI*, 2016.

[22] Nghttp2: HTTP/2 C library and tools. `https://nghttp2.org/`.

[23] 7 Tips for Faster HTTP/2 Performance. `https://www.nginx.com/blog/7-tips-for-faster-http2-performance/`.

[24] A. Nika, Y. Zhu, N. Ding, A. Jindal, Y. C. Hu, X. Zhou, B. Y. Zhao, and H. Zheng. Energy and Performance of Smartphone Radio Bundling in Outdoor Environments. In *WWW*, 2015.

[25] A. Nikravesh, D. R. Choffnes, E. Katz-Bassett, Z. M. Mao, and M. Welsh. Mobile Network Performance from User Devices: A Longitudinal, Multidimensional Analysis. In *Passive and Active Measurement Conference*, 2014.

[26] F. Qian, S. Sen, and O. Spatscheck. Characterizing Resource Usage for Mobile Web Browsing. In *Proc. ACM MobiSys*, 2014.

[27] Scrapbook: Addons for Firefox. `https://addons.mozilla.org/en-US/firefox/addon/scrapbook/`.

[28] S. Singh, H. Madhyastha, K. S.V., and R. Govindan. FlexiWeb: Network-Aware Compaction for Accelerating Mobile Web Transfers. In *Proc. ACM MobiCom*, 2015.

[29] A. Sivakumar, S. Puzhavakath Narayanan, V. Gopalakrishnan, S. Lee, S. Rao, and S. Sen. PARCEL: Proxy Assisted BRowsing in Cellular Networks for Energy and Latency Reduction. In *CoNEXT*, 2014.

[30] S. Sundaresan, N. Feamster, R. Teixeira, and N. Magharei. Measuring and Mitigating Web Performance Bottlenecks in Broadband Access Networks. In *Proc. ACM IMC*, 2013.

[31] M. Varvello, K. Schomp, D. Naylor, J. Blackburn, A. Finamore, and K. Papagiannaki. Is The Web HTTP/2 Yet? In *Passive and Active Measurement Conference*, 2016.

[32] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. Demystifying Page Load Performance with WProf. In *Proc. NSDI*, 2013.

[33] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. How Speedy is SPDY? In *Proc. NSDI*, 2014.

[34] X. S. Wang, A. Krishnamurthy, and D. Wetherall. Speeding up Web Page Loads with Shandian. In *NSDI*, 2016.

[35] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie. Why Are Web Browsers Slow on Smartphones? In *Proc. ACM HotMobile*, 2011.

[36] Y. Zaki, J. Chen, T. Pötsch, T. Ahmad, and L. Subramanian. Dissecting Web Latency in Ghana. In *Proc. ACM IMC*, 2014.

[37] K. Zarifis, M. Holland, M. Jain, E. Katz-Bassett, and R. Govindan. Modeling HTTP/2 Speed from HTTP/1 Traces. In *Passive and Active Measurement Conference*, 2016.