Need for Mobile Speed: A Historical Analysis of Mobile Web Performance

Javad Nejati
Stony Brook University
jnejati@cs.stonybrook.edu

Meng Luo
Stony Brook University
meluo@cs.stonybrook.edu

Nick Nikiforakis
Stony Brook University
nick@cs.stonybrook.edu

Aruna Balasubramanian
Stony Brook University
arunab@cs.stonybrook.edu

***Abstract*—As more and more users rely on their mobile devices for the majority of their computing needs, browser vendors are competing for the user's attention on the mobile platform. As a result, mobile Web page performance has improved over the years. However, it is not clear if these improvements are a result of better browsers, optimized Web pages, new platforms, or improved network conditions. In this work, we conduct a historical study over 4 years with 8 popular mobile browsers, loading over 250K Web pages, to isolate the effect of different factors on page load improvements. Using a combination of record-and-replay proxies, historical data from the Internet archive, and specifications about current and past network speeds, we quantify the performance of mobile browsers over the years (i.e. measuring how a user with a 2015 smartphone and a 2015 mobile browser experienced the 2015 web, compared to today). Our key findings are that: (a) browsers have become more bloated, and older browsers perform better on newer devices, compared to newer browsers on the same new devices, (b) Web pages have become more complex, and (c) improvement in network speeds have a diminishing returns in terms of page performance. Taken together, the improvements we see in mobile page performance over the 4 years is largely due to improved platform in new mobile devices, and not a result of browser, network, or Web page improvements.**

I. Introduction

Mobile browsers are an important part of the Web ecosystem, with over 50% of the Web traffic being transferred to and from mobile devices [11]. Unsurprisingly, users want faster Web access from their mobile devices and there has been considerable work in improving the performance of mobile browsers. There are over 120 mobile browsers in the Android Marketplace alone [24] and each browser has multiple new versions released on a yearly basis, each touting better performance than its previous versions and the competition. For example, Chrome alone has had over 50 [3] versions released in the last 5 years.

The results of these advances are generally promising and anecdotally, page load performance has improved over the years. Our experiments with eight of the most popular mobile browsers on the Android Marketplace show a promising performance trend. Figure 2 shows that if a user loads a page on phone released in 2018 versus an older phone, they will see an improvement across different page-performance metrics including Page Load Time (PLT) [9], SpeedIndex [25], and Time-To-Interactive (TTI) [15].

978-3-903176-27-0©2020 IFIP

The problem, however, is that it is not clear which factors contribute to this improvement in performance. Mobile browsers are complex, and several factors affect page performance. These factors include browser improvements, network conditions, the device capacity, and the page itself. It is critical to identify the factors that lead to the most improvement in page performance, so that researchers and practitioners can determine not only where to focus their attention, but whether these performance benefits are available to all users who load pages on devices and networks with diverse capacities.

To this end, we present a historical study of mobile browser performance over a four-year period, across network conditions, browser versions, devices, and Web pages. To isolate the effect of each of the four factors, we vary one factor while holding the other factors constant. Overall, we collected and analyzed more than $250,000$ measurements, by loading 150 websites over varying conditions. Our analysis is built over an automated and scalable mobile experimentation platform [1] that allows large-scale experimentation across browsers and Web pages, while abstracting away the specifics of individual mobile browsers [24].

In order to measure the page load performance experienced by users over the years, we use Internet's Archive Wayback machine to access old page contents, and load them on devices and network conditions prevalent in 2015, 2016, 2017, and 2018. As expected, there is a large difference in page performance in these four years, with a median PLT reducing from 23.4 seconds in 2015, to 6.4 seconds in 2018. Two other popular page-load metrics, Time To Interactive (TTI) [15] and Speed Index [25] also see an improvement of 45% and 47% respectively over the four years.

On further investigation, we make three key findings. First, we find that newer browser actually perform poorly compared to older browsers. A browser released in 2018 loads a page *slower* compared to the same browser released in earlier years, across all browsers we tested. Over the years, browsers have added new security and other features but the performance of the browsers have deteriorated.Although newer browsers have more features than older ones, our study shows that older browsers are perfectly capable of rendering current web pages faster than the more recent versions. All of the evalu-

[1] By *mobile*, we are referring to devices with compute constraints. However, our goal is not to study browser performance under mobility.

ated versions support important security mechanisms (such as TLS) hence, the observed slowdown can not be explained by increased security alone.

If the users upgrade their phone and OS to the most recent version, the relatively poorer performance of the browser is masked. However, constant platform upgrade cannot be expected across the world. In developing regions, lower-end phones with lower memory and compute capacity are popular [21], and these users will disproportionally be affected by bloated browsers.

Our second finding is that the improvement in page performance offered by network upgrades is saturating. The network bandwidth and round trip times have been progressively improving since 2015. The download speed for mobile devices has increased from 2 Mbps in 2015 to 11 Mbps in 2018 [6], and the round trip time has reduced from 300ms to 50ms during the same period. The page performance improved 92% between 2015 and 2016 due to improvement in network speeds, but since then the improvement has diminished. For example, Chrome's page load time has improved 30% from 2016 to 2017 while it had only 3% change from 2017 to 2018.

Our third finding is that the Web pages are becoming more complex resulting in increasing page load times. The size of JavaScript code has increased by 50% in 2018 compared to 2010, and by 10% compared to 2015. If a user loads a 2010 version of a page, the page load is 15 seconds faster in the median case compared to a 2015 version of the same page, under the same environment conditions. However, the difference in performance due to page complexity is masked by the improvements in the device. There is little difference between loading a 2010 or a 2018 Web page today, even though the newer Web pages are, by all metrics, more complex.

Overall, we find that the improvement in page load performance from 2015 to 2018 is largely because of improvement in the device and operating system. The newer browsers do not result is improved performance, the network improvements are saturated, and newer pages are more complex. However, pages still load faster today compared to 4 years ago because the device and OS has improved considerably. Since browser developers and web developers rely on the presence of high-end mobile devices to mask their decreased performance, we argue that users who do not wish (or cannot afford) to constantly upgrade their devices are experiencing a slower web as their browsers update and the web evolves. We share the code at https://www.github.com/jnejati/tma2020.

II. Platform

The main design goal of our platform is to provide a web performance measurement testbed that can automatically collect performance metrics across different mobile devices, browsers, networks, and Web pages. This framework must be able to conduct these tests without manual intervention so that it can scale to a large number of browsers and devices, without needing constant human supervision. Figure 1 illustrates the architecture of our testbed.

1) A General Framework.: Most browsers provide a management interface so that an automation script can programmatically control the browser's behavior. This method works well when dealing with a single type of browser, such as, Google Chrome, where existing automation tools (e.g. Selenium [1]) can be used to navigate the browser to different webpages and collect performance data. Unfortunately, popular automation tools only support the most popular browsers on any given platform and as such cannot be used to evaluate the multitude of browsers that are available on mobile devices.

In order to build a testbed that works for different kinds of browsers, we require a more general approach that is not dependent on a single browser's features. Our idea is to use a browser-agnostic testing framework to install different mobile browsers, under different Android versions, automatically visit URLs, and collect metrics about the page load. We use a security tool called HindSight [24] that identifies the closest browser instance for a given device and operating system. We then build a browser-agnostic record-and replay proxy to run the Web page and collect metrics. We describe our framework next.

2) A Controlled Environment.: We employ Google's Web Page Replay tool [16] and Linux's `tc` (Traffic Control) tool to control content and network variations respectively. Since Wi-Fi links can incur a variable delay based on the number of connected users and frequency interferences, we opted to using a USB cable configured in *reverse tethering* mode, to act as an Ethernet link.

Replaying recorded pages is not a straightforward task particularly when dealing with HTTPS Web sites. To convince the evaluated browsers to accept the self-signed certificates presented by the proxy, we had to add a new root Certificate Authority (CA) to each browser's trusted CA store. Chrome and Chrome-based browsers respect a system-wide installed root certificate but other browsers, such as Firefox, manage their own certificate stores and require customized processes for adding a new root CA. Note that this process must be repeated for every tested mobile device and, for the browsers with custom CA stores, for every version of each evaluated mobile browser.

For our study were we load pages from Wayback machine, we load all pages across all browser over HTTP/1.1.

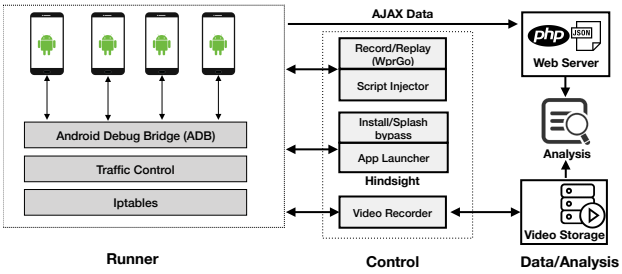


Fig. 1: *Testbed architecture: A controlled environment is built using WprGo and `tc` managed by Hindsight. Performance metrics are collected using ADB and Boomerang and then sent to the Web server.*

TABLE I: *Browser versions and device specifications used in our platform representing the device and Android version popular in 2015, 2016, 2017, and 2018. The corresponding browser versions were released in the beginning of each year.*

Year	OS	Device	CPU	RAM	Firefox	Chrome	Baidu	Ksmobile	Opera	Yandex	UCBrowser	Explore
2015	Android4	Nexus 4	Quad-core 1.5 GHz Krait	8GB	34	37	5.2	5.0	22	14.8	10.7	2.5.2
2016	Android5	Nexus 5	Quad-core 2.3 GHz Krait 400	16GB	40	46	6.2	5.1.98	32	15.12	10.9.5	2.5.9
2017	Android6	Nexus 6P	4x1.55 GHz Cortex-A53 & 4x2.0 GHz Cortex-A57	32GB	52	53	6.3	5.21	34	17.3	12	2.6
2018	Android7	Nexus 6P	4x1.55 GHz Cortex-A53 & 4x2.0 GHz Cortex-A57	32GB	61	64	6.4	5.22	37	17.4	12.11	3.16

3) A Metric Collector: In order to collect browser performance data, we need a general technique that is supported by most browsers and works with current as well as past versions of these browsers. We use Akamai Boomerang [2], a JavaScript library that can be injected by our proxy during the replay phase. Boomerang is a JavaScript library that measures the performance characteristics of real-world page loads and interactions. It supports various type of metrics like overall page load times, DNS, TCP, request and response timings.

To make sure that all Web pages are loaded correctly, we check that the `OnLoad` event is fired. We also perform random checks on the video recordings of Web pages, to verify that the last frame of the video shows the entire page load.

An unexpected complication of incorporating a new JavaScript library in the replay of existing sites is the interaction of that library with a website's Content Security Policy (CSP). CSP is an opt-in security mechanism that websites can use to instruct browsers to declare the allowed origins of remote content so that browsers can refuse to load content that does not come from these origins. To maintain the fidelity of our replay, instead of disabling CSP, we chose to on-the-fly modify outgoing CSP policies and whitelist our endpoint which serves Boomerang library and receives the collected performance metrics from each page load.

To account for any remaining variance (e.g. different scheduling of the browser processes on the mobile devices), we load each evaluated page 5 times and use the median value for each metric. It is important for our tests that browsers do not load any cached contents during the page loads (i.e. we require "cold start" conditions). Since the cache management techniques differ from one browser to another, for each run of the experiment, we install a fresh copy of the browser and use a splash screen bypass technique to automatically skip welcome messages and configuration pages.

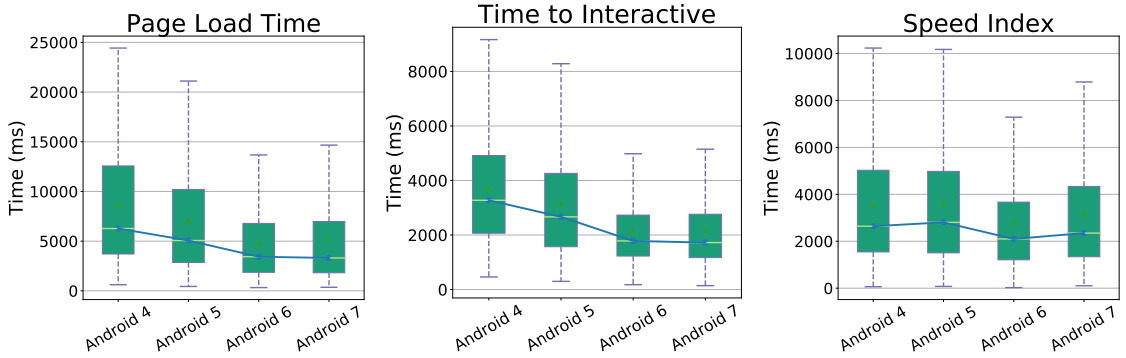
In summary, a typical scenario to run experiments on our platform follows these steps: After setting up the desired network profiles, necessary port forwarding, and USB connections, we record the desired set of pages using Web Page Replay in record mode. After that, we start the Web Page Replay in replay mode which in turn reads the recorded archive file and act as a Web proxy to serve all incoming HTTP requests from mobile devices. For each page, the Boomerang script is injected into that Web page in order to collect performance metrics and transmit them to our Web server via Ajax call. This transmission happens after `OnLoad` event is fired on the page. These metric data are received on our Web server by a PHP program and stored locally as JSON files. In meantime, Hindsight is continuously managing the experiments by installing new instances of mobile browsers on devices, bypassing splash screens and loading pages by launching the relevant activities through ADB. In parallel, a video of each page load is also recorded using ADB and then pulled onto local storage for further analysis.

We quantify page load performance using three most popular metrics: **Page Load Time (PLT)** [9], **SpeedIndex** [25], and **Time To Interactive (TTI)** [15]. PLT is the time between when the page is requested and when the `Load` event is fired by the browser. PLT is the most common metric used to measure page load performance. Recently, SpeedIndex and TTI have been introduced to better quantify page performance from the user's point of view. SpeedIndex is a visual metric that measures the average time for visible content to appear on the browsers viewport. SpeedIndex is measured over a video of the page load process. TTI measures how long it takes for a page to become interactive [15]. Example interactions include clicking on a link or using the auto-complete feature of a search bar.

III. Methodology

Our methodology is as follows: **Experiment with 8 mobile browsers:** There are a large number of mobile browsers on the Android Play Store. We experiment with eight popular browsers that have over 10,000,000 installations each: Chrome, Firefox, Baidu, Ksmobile, Opera, Yandex, UCBrowser, and Explore. We choose browsers that are popular in different parts of the world. For example, UC Browser is more popular than Google Chrome in countries like India and Indonesia [31] and Yandex and Opera are the second most popular browsers in Russia and Africa respectively [10]. We focus on mobile browsers in the Android ecosystem because Android has over 85% of the marketshare worldwide [13], and it is open source that allows a large number of mobile browsers implementations.

Web pages: We choose 150 Web sites out of 2K pages from Alexa's top 1M sites from 2017 to cover different page sizes. We randomly choose 50 pages each of heavy ($>$ 3MB), medium (1-3 MB), and light ($<$1MB) pages. We record mobile version of the pages in the cases where a mobile version exists.



(a) Page Load Time *(b) Time to interactive* *(c) Speed Index*

Fig. 2: *Mobile Web is improving. Web performance metrics for the same set of Web pages when loading pages using todays network speeds, for Android versions and devices popular in 2015, 2016, 2017, and 2018 (represented as Android 4 to Android 7 in the figure). The figure show that in the median case, performance is improving.*

Device and mobile OS: Our testbed employs four Google Nexus mobile devices that were popular in 2015, 2016, 2017, and 2018, each running the most popular Android version in that year ranging from Android 4 to Android 7. This spread allows us to do historical studies. Table I shows a summary of all devices and browsers that we evaluate in this paper. We chose the most popular version of Android OS for each year [12] and the browser version that was released at the beginning of each year. To ensure that one browser version per year is sufficient to capture overall performance trends, we run the Kraken JavaScript benchmark to compare the performance of neighboring versions of each browser. We used t-tests and verified that the neighboring versions performed similarly to the chosen version.

Except for Firefox, all other browsers use Chromium's open source blink rendering engine. FireFox uses its Quantum engine. Even though the browsers (except Firefox) use the same engine, they differ in terms of features. For instance, Yandex runs antivirus software, [17], and KSmobile has a built in Browser Intrusion Prevention System. Baidu, first released in 2011 includes an integrated video and audio downloading tool and a built-in torrent client. UCbrowser, specifically targeting asian markets, delivers local news and video services to its users. Opera browser, which was pioneer in introducing many features adopted by other web browsers, has moved from its own Presto engine to a Chromium Blink engine in 2013. In our experiments, we use the original Opera, not the Opera mini version. Opera mini is a cloud-based browser which receives a pre-rendered page from its cloud endpoint [22] while the original Opera acts like other browsers used in our experiments.

For experiments that recreate the environment of each year (§V), we load pages with the version of the Web page and network conditions representative of each year, from 2015 to 2018. We use the Internet archive's wayback machine to load pages from their respective years. In all we conducted over *250,000 Web page* loads as part of this study.

IV. Current State of the Web

We first capture the mobile browsers experience of the *current* web. We used the four different devices and 8 different browsers listed in Section III to visit the selected Alexa websites served by our record-and-replay proxy, while setting the upload data rate to 4 Mbps, download rate to 3Mbps and delay to 20ms. This upload rate emulates a regular 4G connection in 2019 [4]. Since current pages are loaded in these experiments, we load the page according to the protocol that the page supports, which is either HTTP/1.1 or HTTPS. In total, 23 are used HTTP and 127 use HTTPS as their protocol. None of the pages loaded using HTTP/2. We note there that for the analysis of historical performance (below), all pages were loaded using HTTP.

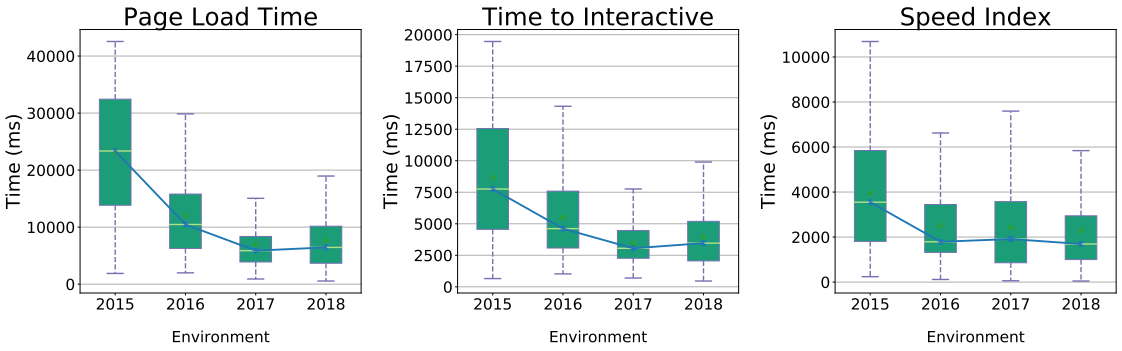
Figure 2 shows the distribution of our three performance metrics (Page Load Time, Time to Interactive, and Speed Index) across all browsers and devices. The combination of newer devices and newer browsers results in a faster performance for all three metrics, even though the Speed Index improvement is less marked than that of the PLT and TTI.

Even though these results indicate that mobile Web performance has improved over the past 4 years (as captured through the use of different mobile devices), it still leaves a number of questions unanswered. Is this performance increase due to better platforms or due to more streamlined browsers? In the next section, we answer these questions by evaluating browser performance in the context of an ever-changing web.

V. Analysis of Historical Performance

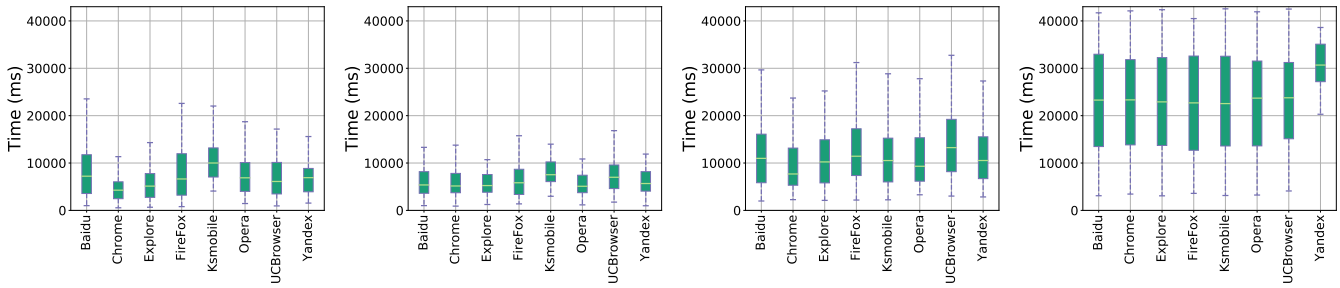
In our previous experiment, the page contents and network speed had been configured to be identical for all platforms. Even though that configuration allowed us to quantify how users owning old vs. new devices experience the web *today*, it does not express how users experience the web over time.

To quantify this user experience in a historical fashion, we obtained the versions of the same Web sites from 2015 to 2018 from Internet archive's [7] Wayback machine. Similarly, we emulate the appropriate network speeds from 2015



(a) Page Load Time *(b) Time to interactive* *(c) Speed Index*

Fig. 3: *Web performance as experienced by a user in each year, when loading contemporary pages over the network, device, and OS popular in that year. Page performance continues to improve every year. The graphs shows the performance of all browsers together.*



(a) PLT, 2018 *Environment* *(b) PLT,* 2017 *Environment* *(c) PLT,* 2016 *Environment* *(d) PLT,* 2015 *Environment*

Fig. 4: *Page Load Times when loading contemporary pages over the network, device, and OS popular in each year from 2015 to 2018 for each browser. PLT improves over the years, but the performance across browsers are largely the same.The similarity is confirmed by t-test.*

to 2018, as the users would have experienced it at those times.These network speeds represent the average speed across the world [6]. The experiment therefore emulates the performance experienced by an average user in the world. Table II lists the different browsing environments we have used for these historical experiments. For example, '2015Env' indicates an environment in which the network is the average network available in year 2015, browser versions are those released in 2015 (described earlier in Section III) running on top of an Android device that was widely used in 2015 and Web pages are from mid-2015 (as captured by the Internet archive).

Figure 3 shows the general trends when contemporary profiles are applied while Figure 4 focuses on the PLT metric and presents a breakdown by specific browser. Overall, we observe the same improvements over time like we did in our earlier experiments, even though the performance increases are more dramatic due to the poorer network conditions associated with older environments.

Environment	(Download, Upload, RTT)	Year of Pages	Device	Android version
2018Env	(11.1Mbps, 5Mbps, 50ms)	2018	Nexus 6P	7
2017Env	(8.7Mbps, 4Mbps, 100ms)	2017	Nexus 6P	6
2016Env	(6.8Mbps, 3Mbps, 150ms)	2016	Nexus 5	5
2015Env	(2Mbps, 800Kbps, 300ms)	2015	Nexus 4	4

TABLE II: *Contemporary environments used for historical analysis spanning over 4 years.*

Over the years, Chrome has the highest improvement in PLT, improving 446% 2015 to 2018, followed by Explore that improved PLT by 347% and Yandex by 344%. In 2015, Yandex is 33% slower than other browsers but stays on par with other browsers from 2016 to 2018. In 2015, on average, there is a 5.39% variance from the median PLT for all browsers. In 2016, we observe a 10% variance, in 2017, there is a 11.7% variance and finally in 2018, on average, there is a 16.4% difference from the median of PLTs among all browsers. As the network and devices improve over the years, PLT decreases making the changes between the browsers more pronounced. So even though many of these browsers use the same rendering engine, their performance is slightly varying because of additional features they provide.

Further in Figure 4 we observe that there is little difference in median Web page load performance across browsers (barring a few outliers in the 2015 environment). However, we plotted the difference in median PLT between the best performing and worst performing browsers for each website. Figure 5 shows that, even though for 13% of the webpages, all browsers have similar median performance, for 87% of the webpages, the difference in browser performance ranges from 2 seconds to 12.9 seconds. This observation is aligned with Rajiullah *et al.* [33] findings. However, we could not find any correlation between the performance difference across

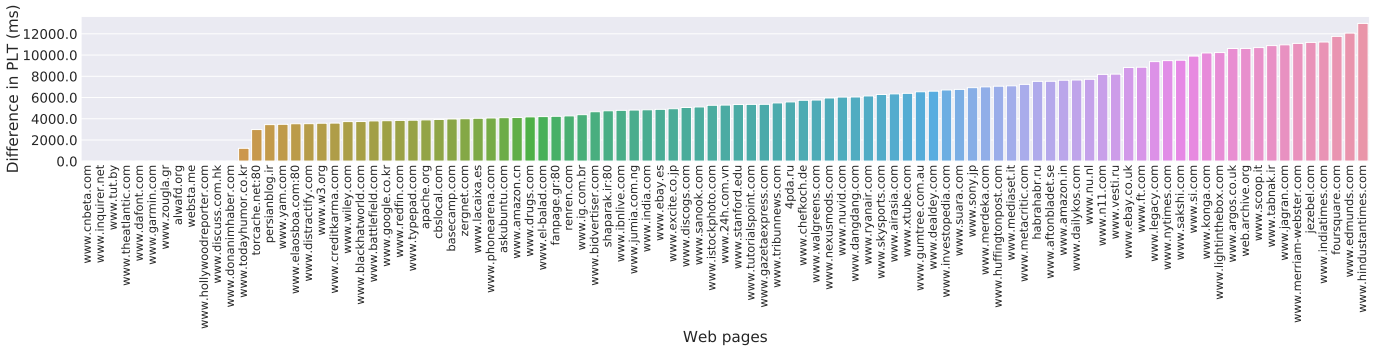


Fig. 5: *Difference between the median best performing and worse performing browsers for each page for 2018 environment. Although for 13% of the cases, the difference is almost zero, for the remaining cases, we observe differences ranging from 2 to 12.9 seconds.*

browsers and Webpage features.

But irrespective of the browser, the results demonstrate that users are clearly experiencing a more performant mobile web as time passes. We conduct further experiments over these contemporary profiles to determine which of the factors—device, browser, network, or page content—provide the most benefit in terms of performance. To this end, we load Web pages by keeping one or more of the factors a constant and varying others. Our findings are described below.

A. Performance consequences of feature creep

Every update of a mobile browser presumably aims at improving support for new mechanisms and protocols, as well as increased usability, security, and performance.To understand how new versions of browsers contribute to the trend of better web performance (as shown in Section 3) we conducted the following experiment. We chose the two "edges" of our dataset (2015/2018 mobile devices) and swapped the versions of browsers, while keeping all other factors constant. We evaluated the performance of a 2015 device running 2018 browsers and a 2018 device running a 2015 browsers.

Figure 6 shows that for virtually all metrics, the 2015 browser outperformed the 2018 browser, when running on the 2018 mobile device. Similarly, the 2018 browser was slower than the 2015 browser, when both were running on top of a 2015 mobile device. We combine the browsers other than Chrome and Firefox as *Others* rather than show each individual browser performance because they all use the same underlying engine and perform similarly.

Since all other factors were kept constant, this demonstrates that newer versions of browsers are *slower* than their older counterparts. We verified that the trend continues when we swap 2018 browsers for 2016 and 2017 browsers as well; i.e., running an older browser on a newer device improves performance. Many browser vendors push their new releases as faster and more performant browsers. However, our results suggest that, if users do not upgrade their phones, they will see poorer performance with newer browsers.

Given our experimental controls, the reason for this slowdown can only be the feature creep that happens to software over time where more and more features are slowly added and increase the footprint of the running application. Figure 7 shows how the size of browser APKs has increased over the years. We argue that the size of an APK is a proxy of the size of its code and therefore the complexity of the running applications. Ksmobile, Explore, and Baidu exhibit a significantly smaller footprint compared to the rest of the browsers. This is because these browsers are using the existing WebView packages already present on the phone, and do not need to "ship" with their own rendering engines.

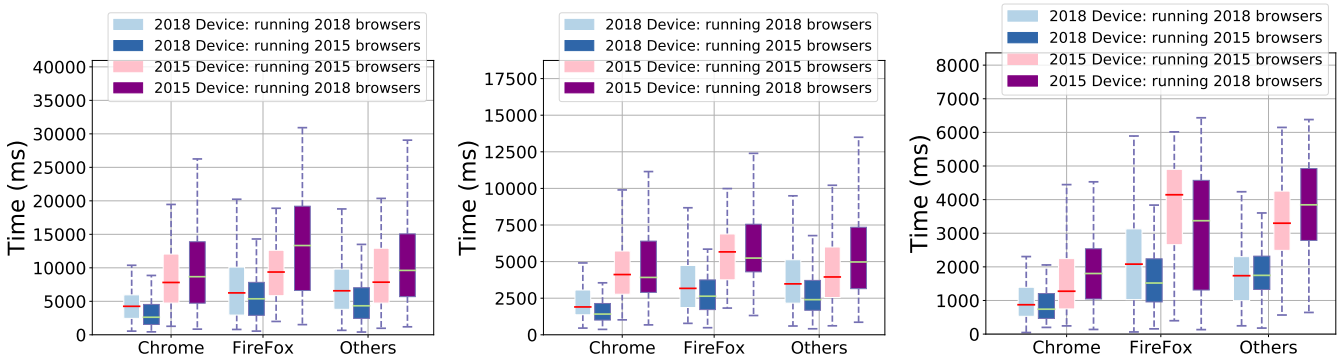
To further validate our observations, we use Kraken [8], a JavaScript-based performance benchmark from Mozilla, to measure the performance of our "swapped" browsing environments. Kraken focuses on realistic workloads and includes 14 benchmarks in 5 different categories: Search, Audio, Imaging, JSON, and Encryption. Fig 8 shows that the combination of device's CPU and OS is the main factor for its performance while the browser version has a small effect which usually is in favor of the older browser versions.

B. Saturation of network impact

Next, we study the effect of improved network on the page load performance. To this end, we vary the network conditions by applying four different network conditions prevalent in the years 2015 to 2018. We keep all other factors a constant and load Web pages from 2018 on 2018 devices and browser versions. Figure 9 shows the median PLT. The performance jump between the 2015 and 2016 network is the highest, where the median PLT reduces by nearly 10 seconds. With further improvement in network conditions from 2016 to 2018, there is only a modest improvement in PLT. The impact of network saturates when the downlink bandwidth is about 6.8Mbps and delay is less than 150ms.

Our findings are aligned with previous studies [27] which discovered that when network is very slow, it is the dominant bottleneck. But beyond a certain network speed, computation becomes the bottleneck for mobile browsers. As a result, improving the network further only has a relatively smaller impact on Web performance. We see similar trends for the TTI and SpeedIndex metrics (not shown here for brevity).

This result is especially significant given that network speeds are improving in many parts of the developing world.



(a) Page Load Time *(b) Time to interactive* *(c) Speed Index*

Fig. 6: *Older browsers results in faster page loads compared to newer browsers when run on newer devices. The* Other *legend combines the performance of all browsers except Chrome and Firefox.*

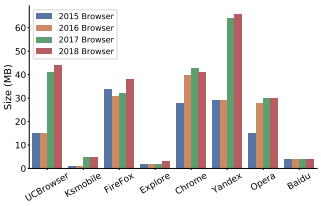


Fig. 7: *APK size for each browser per platform. The size of mobile browsers' installers are increasing over the years.*

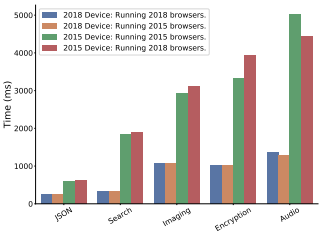


Fig. 8: *Kraken benchmark on 2015 and 2018 Environments show that older browsers run better than newer browsers on new devices for the benchmarks.*

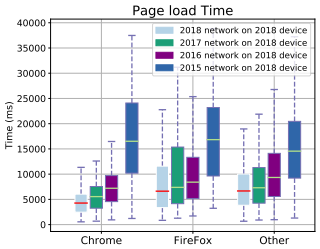


Fig. 9: *Page load times when loading Web pages using the prevalent network conditions in 2015, 2016, 2017, and 2018 [6] (see Table II). The Web page, device, and browser was held constant. The largest improvement in performance is between 2015 and 2016, after which PLT improvement is modest in response to network improvements.*

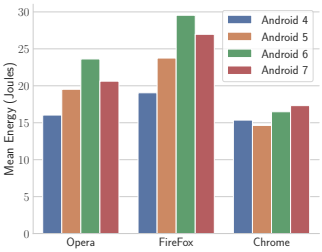


Fig. 10: *Mean energy consumption when loading https://www.npr.org for different browsers across different devices. Energy usage has increased over years as new devices ship with larger screens and more powerful compute resources.*

For example, in India, average network bitrate has increased from 1.7Mbps in 2013 [14] to 4.6Mbps for mobile in 2018 and is projected to reach 16.3 Mbps in 2023 [5]. However, hardware is still a bottleneck, a large number of phones are still low-end devices. [21].

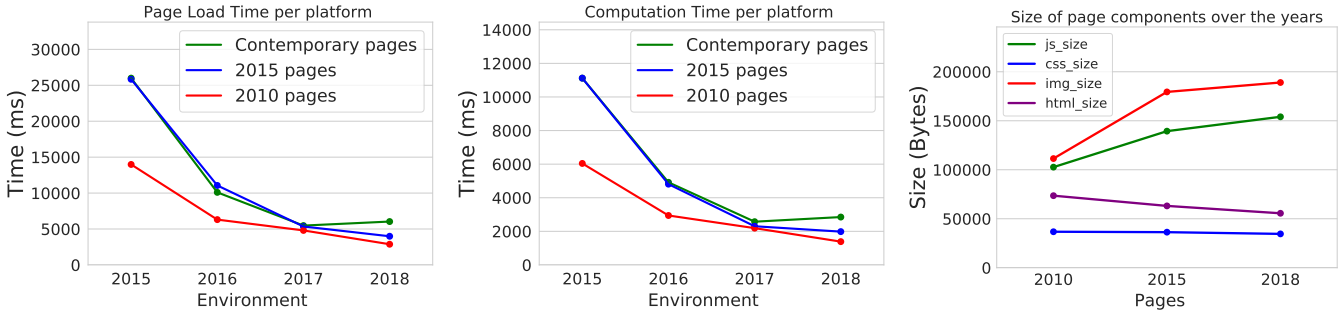
C. Pages are complex

Next we isolate the effect of the Web page. Figure 11a shows the PLT trend when loading versions of the same Web page from 2010, 2015, and 2018.The points in these trend lines are representing the median value for each distribution. We load the Web page in the 2015 to 2018 environment (Table II). Even though the performance for all sets of pages is becoming better as the underlying environment improves, we can clearly see the difference between the 2010 set of pages and the 2015/2018, particularly in older browser environments. For example, a 2010 page loads in 10 seconds but a 2015/2018 page takes 25 seconds to load, in the 2015 environment. This is primarily because the 2015 version of the same Web page is more complex compared to the 2010 version. Figure 11c shows steady increase in the size of JavaScript code and images with only a slight decrease of HTML code and CSS size in Web pages. This makes intuitive sense since websites rely more and more on JavaScript and dynamically populate pages with content that is fetched asynchronously via AJAX requests.

Finally, Figure 11b shows the computation time for the same experiment for different pages across different devices. Computation time is total time of compute activities on the critical path. Critical path is the longest path on the page dependency graph composed of compute and network activities. We employ WProfX [29] to calculate the computation time. We use the term "contemporary" to refer to using the version of pages that correspond to the underlying environment, i.e., 2016 pages were "contemporary" to the 2016 browsing environment. There, we again observe that the mobile devices spends less and less time on computation, even though pages are becoming increasingly complex.

D. Improved Web performance is largely because of improved mobile platforms

Overall, our study finds that Web pages are becoming more complex, the network impact has saturated, and browsers are becoming more bloated and less performant. Yet, the performance of the Web is improving over the years. Combining



(a) Page Load Time *(b) Computation time* *(c) Page complexity trend from 2010*

Fig. 11: *A 2010 version of a Web page loads faster compared to a 2015 version of the same page, under older network and device conditions. This is largely because pages are becoming more complex; the image size has nearly doubled in 2018 compared to 2010 (c).*

our observations over real Web-page loads and JavaScript benchmarks, we conclude that the increased performance that users experience is due to the more powerful underlying hardware and operating system which *masks* the slowdown of newer browser versions and complex Web pages.

This observation is important because not every user is capable of constantly upgrading their mobile device. A software-development approach that necessitates ever more powerful hardware is an approach that excludes cost-sensitive users and entire geographical regions that cannot afford the price of high-end devices. This not only furthers the existing digital divide between developed and developing countries but can also lead users to avoid updating their browsers, out of fear of decreased performance. Moreover, the improvement in device computation power often comes at a cost. Figure 10 shows that newer devices are becoming more power hungry for the same page workload. In an experiment , we loaded the same *https://www.npr.org* Web site 30 times and calculated the mean energy consumption for that duration. We observe that as mobile platforms are getting more powerful, energy consumption increases as well. This would introduce a a trade-off between performance and duration of usage.

VI. Related work

Mobile Web performance is a well-studied research topic. Rajiullah *et al.* [33] run experiments over two million Web-pages using 11 different network operators in Europe. They find page complexity has a big impact on the browsing performance, but their study is restricted to two browsers in the current environment. Butkiewicz *et al.* [19] finds the number of objects requested have the most impact on page load time in 2012 on desktop browser.

Several tools have analyzed the performance of Web page loads by studying dependencies. WebProphet [23] implemented a tool to extract dependencies in order to predict performance for Web services. WProf [34] extended the idea by instrumenting Chromium to capture low-level browser information to build the dependency graph. WProf's dependency relations have been used to evaluate effect of different performance optimizations such as caching and HTTP/2 [35]. Polaris [30] builds on top of WProf to get more dependency relations beyond the lexical dependencies. However, these tools are specific to Chrome versions and cannot be easily extended to other browsers for historical studies.

There has also been work on studying the bottlenecks in Web performance. Nejati *et al.* [28] finds that the main bottleneck in mobile Web performance is the computation part of the page load process. In another study, Dasari *et al.* [21], [20] studied the impact of device performance on mobile Web QoE. Their study shows that Web browsing is more impacted by device's CPU when compared to video applications. Narayanan *et al.* [26] observe that Web page load time does not incur a notable difference when network is upgraded from 4G LTE to 5G. Our work corroborates the findings of these studies across different browsers and years.

In the context of security, unnecessary complexities in Web applications have been found to add vulnerabilities to the Web sites. [32], [18]. We show that the same argument applies to the bloated Web browsers. While new features and enhancements add more complexities to the code base, they do not necessarily increase the performance.

VII. Conclusion

In this paper, we investigated the evolution of mobile web performance over a period of four years. By using multiple devices, browsers, Web site versions, and network conditions, we sought to quantify the improvement of page-load performance and identify the factors behind it. We discovered that even though the web is becoming faster across all browsers, the underlying hardware of newer mobile devices is the key driver of these improvements. Specifically, we showed that older browsers run faster than their newer counterparts on the same hardware and that network improvements provide diminishing returns in terms of page performance. Our findings highlight the fact that web developers and browser vendors rely on ever-more performant hardware to mask their overheads which can harm both the user experience and potentially the security of users who cannot afford to constantly upgrade their mobile devices.

VIII. Acknowledgements

We would like to thank the anonymous reviewers for their valuable feedback. This work has been partially funded by the National Science Foundation, through CNS-1566260 and CNS-1617593 grants and Office of Naval Research (ONR), grant N00014-17-1-2541.

References

[1] Automating web browsers, https://www.seleniumhq.org
[2] Boomerang: a javascript library for real user monitoring, https://developer.akamai.com/tools/boomerang/
[3] Chrome mobile version history, https://en.wikipedia.org/wiki/Google_Chrome_version_history
[4] Chrome on ios, https://umaar.com/dev-tips/66-network-throttling-profiles/
[5] Cisco annual internet report, https://www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report
[6] Cisco visual networking index, https://www.cisco.com/c/m/en_us/solutions/service-provider/vni-forecast-highlights.html
[7] Internet archive, https://archive.org/
[8] Kraken javascript benchmark (version 1.1), https://krakenbenchmark.mozilla.org/
[9] Load event, https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event
[10] Mobile browser market share, https://gs.statcounter.com/browser-market-share
[11] Mobile-only users surpass desktop-only users. http://marketingland.com/mobile-only-users-surpassed-pc-only-users-in-march-comscore-126952
[12] Share of android platforms on mobile, https://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/
[13] Smartphone market share, https://www.idc.com/promo/smartphone-market-share/os
[14] State of the internet report., https://www.medianama.com/2014/07/223-india-internet-speed-akamai-q12014
[15] Time to interactive, https://developers.google.com/web/tools/lighthouse/audits/time-to-interactive
[16] Web page replay, https://github.com/catapult-project/catapult
[17] Yandex browser, https://techcrunch.com/2012/10/01/yandex-gives-google-a-one-two-punch-in-russia-a-new-browser-and-an-app-store-for-the-local-search-giant/
[18] Azad, B.A., Laperdrix, P., Nikiforakis, N.: Less is more: quantifying the security benefits of debloating web applications. In: 28th USENIX Security Symposium USENIX Security 19. pp. 1697–1714 (2019)
[19] Butkiewicz, M., Madhyastha, H.V., Sekar, V.: Understanding website complexity: measurements, metrics, and implications. In: Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference. pp. 313–328. ACM (2011)
[20] Dasari, M., Kelton, C., Nejati, J., Balasubramanian, A., Das, S.R.: Demystifying hardware bottlenecks in mobile web quality of experience. In: Proceedings of the SIGCOMM Posters and Demos. pp. 43–45. ACM (2017)
[21] Dasari, M., Vargas, S., Bhattacharya, A., Balasubramanian, A., Das, S.R., Ferdman, M.: Impact of device performance on mobile internet qoe. In: Proceedings of the Internet Measurement Conference 2018. pp. 1–7. ACM (2018)
[22] Kondracki, B., Aliyeva, A., Egele, M., Polakis, J., Nikiforakis, N.: Meddling middlemen: Empirical analysis of the risks of data-saving mobile browsers
[23] Li, Z., Zhang, M., Zhu, Z., Chen, Y., Greenberg, A., Wang, Y.M.: WebProphet: automating performance prediction for web services. In: Proc. of the USENIX conference on Networked Systems Design and Implementation (NSDI), 2010
[24] Luo, M., Starov, O., Honarmand, N., Nikiforakis, N.: Hindsight: Understanding the evolution of ui vulnerabilities in mobile browsers. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 149–162. ACM (2017)
[25] Meenan, P.: Speed index. https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index (September 2014)
[26] Narayanan, A., Carpenter, J., Ramadan, E., Liu, Q., Liu, Y., Qian, F., Zhang, Z.L.: A first measurement study of commercial mmwave 5g performance on smartphones. arXiv preprint arXiv:1909.07532 (2019)
[27] Nejati, J., Balasubramanian, A.: An In-depth Study of Mobile Browser Performance. In: Proceedings of the 25th International Conference on World Wide Web. pp. 1305–1315. WWW '16, Montreal, Quebec, Canada (2016)
[28] Nejati, J., Balasubramanian, A.: An in-depth study of mobile browser performance. In: Proceedings of the 25th International Conference on World Wide Web. pp. 1305–1315. International World Wide Web Conferences Steering Committee (2016)
[29] Nejati, J., Balasubramanian, A.: Wprofx: A fine-grained visualization tool for web page loads. In: To appear in Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (2020)
[30] Netravali, R., Goyal, A., Mickens, J., Balakrishnan, H.: Polaris: Faster page loads using fine-grained dependency tracking. In: 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16). USENIX Association
[31] Purnell, N.: A Browser You've Never Heard of Is Dethroning Google in Asia
[32] Quach, A., Prakash, A., Yan, L.: Debloating software through piece-wise compilation and loading. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). pp. 869–886 (2018)
[33] Rajiullah, M., Lutu, A., Khatouni, A.S., Fida, M.R., Mellia, M., Brunstrom, A., Alay, O., Alfredsson, S., Mancuso, V.: Web experience in mobile networks: Lessons from two million page visits. In: The World Wide Web Conference. pp. 1532–1543. ACM (2019)
[34] Wang, X.S., Balasubramanian, A., Krishnamurthy, A., Wetherall, D.: Demystify page load performance with wprof. In: Proc. of the USENIX conference on Networked Systems Design and Implementation (NSDI) (2013)
[35] Wang, X.S., Balasubramanian, A., Krishnamurthy, A., Wetherall, D.: How speedy is spdy? In: Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation. pp. 387–399. NSDI'14, USENIX Association, Berkeley, CA, USA (2014), http://dl.acm.org/citation.cfm?id=2616448.2616484