An In-depth study of Mobile Browser Performance

Javad Nejati
jnejati@cs.stonybrook.edu

Aruna Balasubramanian
arunab@cs.stonybrook.edu

ABSTRACT

Mobile page load times are an order of magnitude slower compared to non-mobile pages. It is not clear what causes the poor performance: the slower network, the slower computational speeds, or other reasons. Further, most Web optimizations are designed for non-mobile browsers and do not translate well to the mobile browser. Towards understanding mobile Web page load times, in this paper we: (1) perform an in-depth pairwise comparison of loading a page on a mobile versus a non-mobile browser, and (2) characterize the bottlenecks in the mobile browser *vis-a-vis* non-mobile browsers. To this end, we build a testbed that allows us to directly compare the low-level page load activities and bottlenecks when loading a page on a mobile versus a non-mobile browser. We find that computation activities are the main bottleneck when loading a page on mobile browsers. This is in contrast to non-mobile browsers where network activities are the main bottleneck. We also find that the composition of the critical path during page load is different when loading pages on the mobile versus the non-mobile browser. A key takeaway of our work is that we need to fundamentally rethink optimizations for mobile browsers.

1. INTRODUCTION

Recent reports estimate the number of smartphone subscriptions worldwide to be over the 2.6 *billion* mark [4]. Already, the number of users for whom the mobile phone is their only *computer* has exceeded the number of PC-only users [10]. As a result, mobile pages are becoming the primary portal for content, and mobile browsers are one of the most popular applications on smartphones [18]. Unfortunately, the page load performance on mobile devices does not match up to its importance: mobile page load times are an order of magnitude slower compared to loading pages on a desktop browser, often taking 10s of seconds to load just the landing page [17].

It is not easy to improve mobile Web performance: most of the Web optimizations [12, 25, 24] are designed for desktop browsers[1]. For example, HTTP/2 [12, 9] was designed to improve page load performance for primarily non-mobile browsers, and their effect on mobile pages have not been significant [20]. The problem is that mobile and desktop browsers have different bottlenecks and resource constraints. As a result, Web optimizations designed for desktop page loads cannot be directly ported to mobile page loads.

[1]We use the term non-mobile browsers and desktop browsers interchangeably

Copyright is held by the International World Wide Web Conference Committee (IW3C2). IW3C2 reserves the right to provide a hyperlink to the author's site if the Material is used in electronic media.
WWW 2016, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4143-1/16/04.
http://dx.doi.org/10.1145/2872427.2883014.

Some recent research works [16, 30, 17] have specifically optimized mobile pages. FlyWheel [16], Google's data compression proxy, significantly reduces data usage on mobile devices, but its effect on page load time is mixed. Others target specific aspects of the page load process such as the network latency [30] or user QoE [17] instead of the page load time.

The problem is that the mobile pages are varied; an optimization that works well for one page may not work well for another. In addition, mobile page load bottlenecks are not well understood. Mobile devices are scaled down versions of desktops: they have limited CPU, poorer network capacities, and lower memory. It is unclear what the bottleneck is, and how this bottleneck compares to loading pages on non-mobile browsers. It is critical to understand the bottlenecks in mobile browsers to design appropriate optimizations.

Towards understanding mobile page load times, in this paper we: (1) perform an in-depth pairwise comparison of loading a page on a mobile versus a non-mobile browser, and (2) characterize the bottlenecks in a mobile browser *vis-a-vis* a non-mobile browsers. This in-turn can inform us on how and when to port desktop optimizations to mobile.

Unfortunately, it is not straightforward to isolate the effect of the browser and the mobile device on the page load performance. Even if we load the same page on the desktop and the mobile device, the difference in their performance could be due to several factors. Web page performance is influenced by the underlying network, changes to the Web page, the latencies at the Web server, the device, and the browser. It is challenging to determine what part of the page load performance is affected by the mobile device and the mobile browser, and what part is affected by other factors.

Further, extracting the browser dependencies is challenging. The browser is a complex piece of software consisting of a series of interdependent network and computation activities [31]. Because of the interdependence, some of the activities can be performed only after a previous activity finishes, leading to bottlenecks, and creating a critical path [6]. Identifying the bottlenecks require knowledge of the internal browser structure and browser policies.

To address the bottleneck problem, we leverage WProf [31]. WProf is an in-browser profiler designed for desktop browsers that infers the dependency policies of browsers. WProf instruments the browser to log fine-grained timing information. It combines the timing information and the dependency policies to build the dependency graph and to identify the bottlenecks. We port WProf to mobile by instrumenting a mobile browser. We call this instrumented mobile browser WProf-M. Similar to WProf, WProf-M extracts the dependency graph and identifies the bottlenecks, but does so for mobile page loads.

To better isolate the effect of the browser and the mobile device on the page load performance, we build an experimental testbed to perform controlled experiments. The testbed uses several virtualized Web servers that serve local pages under a controlled network environment. The controlled environment reduces the variances caused by external factors and helps us isolate the effect of the mobile browser/device. We use a WProf instrumented desktop browser, and a WProf-M instrumented mobile browser to log low level page load activities. By combining the testbed and the low level activity logging, we are able to compare the desktop and the mobile page load performance at a fine-grained level.

We experiment with 200 Web pages, choosing pages from a mix of popular and not-so-popular pages from Alexa [1]. Our key finding is that it is computation activities, not network activities, that are the main bottleneck in mobile browsers. On a mobile browser, computation activities such as HTTP parsing constitutes for over 60% of the page load bottleneck in the median case. In contrast, computation activities constitutes less than 40% of the page load bottleneck on a desktop browser under the same network condition. When network conditions worsen, network activities become even more of a bottleneck when loading pages on the desktop browser. In contrast, on the mobile browser, even when the network conditions get worse, computation activities continue to be the primary bottleneck. We verify that, even when loading pages *in-the-wild*, without our controlled testbed, computation is the primary bottleneck on the mobile browser.

When loading mobile versions of a page (e.g., loading `m.cnn.com` instead of `cnn.com`), we find that computation activities are still the bottleneck during page load time. This is not surprising because mobile versions of the page reduces object size, but it does not simplify the computation required to load the page. As a result, computation activities become even more of a bottleneck when loading mobile versions of the page.

Next we compare the activities on the critical path (i.e., the bottleneck path), when loading pages on the mobile versus the desktop browser. We find that, even when loading the same page under the same conditions, the specific activities on the critical path vary between the mobile and the desktop browser. Our observation indicates that optimizing a specific URL corresponding to a Javascript or image may not provide improvements on both desktop and mobile browsers, since the specific URL may not be on both critical paths. However, we find that the type of activities on the critical path, for example, in terms of the percentage of Javascript evaluation versus CSS evaluation activities, remain largely similar for the desktop and the mobile browser.

Finally, we find that for small pages that are less than 250KB in size, the performance difference between the desktop and the mobile browser is insignificant. The performance difference is more significant in larger pages, but the difference is because the time taken to complete the computation activities on the mobile browser is much longer when compared to the desktop browser.

Our findings have several implications for browser vendors and mobile page developers. It shows that one of the keys to improving mobile page load performance is to address the computation bottleneck. While several existing optimizations such as FlyWheel [16], Parcel [30], and HTTP/2 [12], attempt to reduce the network latency, there has only been limited effort in improving the computation latency. Our observations show that optimizations may not help mobile and desktop browsers equally because of the differences in the critical path, even when loading the same page. Finally, our findings show that when loading smaller pages, the performance on the mobile browser is similar to the performance on the desktop browser. Therefore, we should focus on optimizing larger pages.

2. BACKGROUND

The Page load Process: Figure 1 shows the various activities involved in the page load process. The page load process is a combination of network and computation activities. The page load starts with *Object loading*, a network process that downloads the HTML page for the input URL. Upon receiving the first chunk of the HTML page, *HTML parsing* starts to iteratively parse the page. Parsing is a computation process. As the page is being parsed, more objects are loaded by the Object loader, thus creating a dependency between the network and the computation activities. If the object is a Javascript (JS) or a style sheet, it is *Evaluated*. The evaluation may in turn trigger more object loading. In parallel, the *Rendering* process progressively renders the page. As the HTML is being parsed, the Document Object Model, or DOM [14] is constructed. The DOM is the intermediate structure constructed by browsers; it provides a common platform for rendering. Both HTML parsing and the JS evaluation manipulate the DOM, creating potential write conflicts.

Critical path: If there were no dependencies, the four activities run in parallel, and page load performance is determined by the slowest of the four processes. However, there are several dependencies imposed by browser policies, resource limitations, and potential write conflicts [31]. Because of these dependencies, certain activities can only be performed after a previous activity completes, creating bottlenecks and a dependency graph. The critical path is the longest path in the dependency graph, such that reducing the latency of any activity not on the critical path cannot reduce the page load time.

The critical path is the bottleneck path. Only optimizations that reduce the activities on the critical path can reduce the page load time. This makes critical path analysis crucial. For example, if most of the activity on the critical path is HTML parsing (a computation activity), we can conclude that the page load time cannot be significantly improved by speeding up the network.

WProf WProf [31] is an in-browser profiler that captures the dependencies and identifies the critical path during page load. Figure 2 shows the dependency graph for an example page extracted by WProf. The different page load activities such as HTML parsing, JS downloading and evaluation, and image downloads, are shown in the figure. The critical path (shown in red) is the longest bottleneck path. In this example, HTML downloads, HTML parsing, Javascript loading and evaluation, and an image download are all part of the critical path.

3. DESIGN AND METHODOLOGY

The goal of this work is to study the root cause of slow page load times on the mobile browser in comparison to the desktop browser. First, we leverage WProf to build an instrumented mobile browser called WProf-M. Together, WProf and WProf-M identify the low level activities during page load times on the desktop and the mobile browser, respectively. For any page we load, either on the mobile or the desktop browser, we get a detailed dependency graph similar to the one shown in Figure 2.

Next, we use an experimental testbed to run controlled load experiments. The controlled environment allows us to isolate the page load performance differences caused by the mobile device better, by reducing the variances caused by external factors.

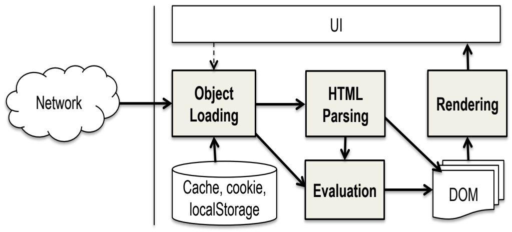


Figure 1: The page load process involving four activities (shown in gray). Figure from [31]

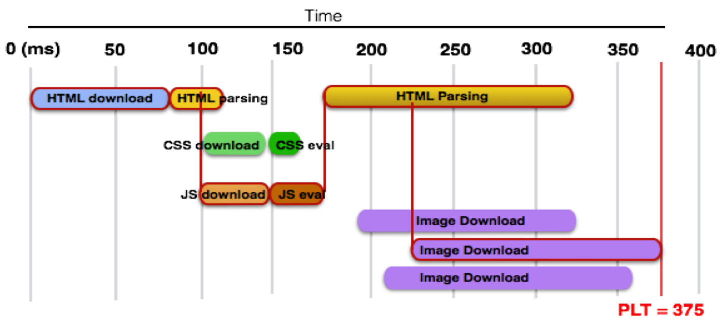


Figure 2: The dependency graph when loading an example page. The figure shows all the activities during the page load process. The line in red shows the critical path. The Page Load Time (PLT) for this page load is 375ms.

3.1 WProf-M

To perform critical path analysis on mobile devices, we built a version of WProf [31] on Android Chromium Version 31.0.1626.0. As a first step, we infer the set of dependency policies. Dependencies occur because of the underlying browser policy. To uncover these dependencies, we use the same technique as described in WProf [31]: we load a set of test Web pages and observe the browser behavior when loading the Web pages. For example, to infer the browser dependency policies in loading Web objects, we create test pages with two objects. We then artificially create delays in loading the first object. If the subsequent object is also delayed, we infer that the two objects are dependent. The full set of test pages we use to infer the dependency policies can be found at `wprof.cs.washington.edu/tests/`.

From our analysis we find that the set of dependency policies we observed in Android Chromium Version 31.0.1626.0 are the same as the dependency policies observed in Chromium Version 31.0.1626.0 that we instrument for WProf. Therefore, we port the WProf instrumentation to the Android Chromium version directly. The instrumentation logs the timing information for various browser activities, and then uses the dependency policies to construct the dependency graph and the critical path. Minor modifications were made to accommodate for changes between the desktop and the mobile browser versions.

We call the instrumented Android Chromium application WProf-M. Most of the instrumentation was done on Webkit, the open source browser engine used by Chromium. In total, the difference in lines of code between Android Chromium and WProf-M is 4191 lines, as indicated by the patch file.

The WProf-M browser application has been released and can be downloaded from `wprofx.cs.stonybrook.edu`.

3.2 Testbed Design

The key to our analysis is to load pages on the desktop and the mobile browser *under the same conditions*, to perform pairwise comparisons. However, this is non-trivial: page load performance exhibits high variance [32]. This means, the difference in page load times may not only because of loading the page on the desktop versus the mobile device. It could be because of several other reasons including network variations, changes to the Web page, or delay at the Web server.

We design an experimental testbed that minimizes external variances when loading pages over the desktop and the mobile browser. We perform the experiments in a controlled environment by:

- serving pages from our local server. We download the entire page locally on the server and convert all the external links to local links. This minimizes variances caused by changes to the page.

- loading the same page on the mobile and the desktop browser, rather than loading the mobile version of the page (or *mpages*) to perform more direct comparisons.

- emulating different network conditions using a traffic controller to ensure that both the mobile and the desktop browser load pages under the same network conditions.

To ensure that the results we get from our controlled setting applies more broadly, we perform additional experiments where the three restrictions are removed; i.e., we load pages directly from the Web server, we load *mpages*, and we use real networks. Our additional experiments show that the conclusions we derive from our controlled experiments also apply more generally (§5.5).

Figure 3 shows our experimental testbed. At the client side, we load Web pages on a phone using the WProf-M browser, and on the desktop we load pages using the WProf browser. All Web page loads go through the experiment manager. The manager stores the logs generated by WProf and WProf-M. The manager also configures the traffic controller and the Web server according to the experiment. We use reverse USB tethering to connect the mobile device to the experiment manager [33] rather than connect using WiFi because we observed large variances in WiFi latencies.

On the server side, we leverage virtual machines to run multiple web servers on the same platform. To isolate the different network stacks for the difference virtual servers, we use Linux network namespace [11], similar to [26]. To emulate different network conditions, we use Linux Traffic Control (TC). Before each experiment, we use the *ping* and *iperf* tools to test that the emulated network has the expected bandwidth and delay values.

3.3 Methodology

We load the same set of Web pages on the desktop and the mobile browser and collect logs of the low level browser activities as shown in Figure 2. We use the logs to identify the bottlenecks and the critical path during the page load process. By comparing the two log files, we are able to study the differences in page load bottlenecks when loading a page on the mobile versus the desktop browser.

We divide the activities on the critical path into computation and network activities as follows:

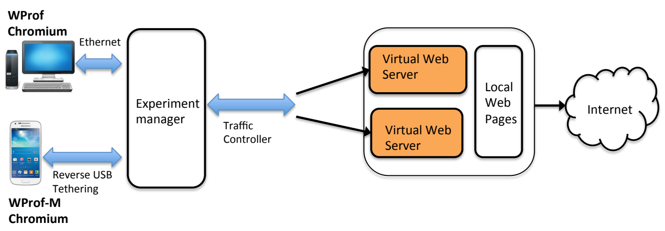


Figure 3: Testbed architecture. All results presented in this paper (except the *"in-the-wild"* experiments) are run on this testbed.

- Computation activities: HTML Parsing, Javascript/CSS evaluation, and rendering.
- Network activities: Downloading objects.

For example, in Figure 2, the critical path is a mix of computation and network activities: JS evaluation and HTML parsing are computation activities, and downloading the HTML, Javascript, and the images are network activities. Note that several activities occur off of the critical path. Unless stated, we do not consider the activities that are not on the critical path in our analysis because they are not bottlenecks.

4. EXPERIMENTAL SET UP

Server and Client: All Linux instances are virtual machines running inside a VMware ESXi 6.0 Bare-Metal Hypervisor. On average, 4 cores at 2.6GHz and 2GB of RAM has been assigned to each virtual machine.

We use two Samsung Galaxy S4 phones each with 2GB of RAM and powered by a 1.9 GHz Quad-core Krait 300 CPU running Android KitKat. We also experiment with a Samsung Galaxy S6 phone with 3GB of RAM and Octa-core 1.5/2.1 GHz Cortex-A53, Cortex-A57 CPUs running Android Lollipop. By default, we present experiments conducted on Samsung Galaxy S4 phones on the controlled testbed.

Network: We run the emulated network experiments on 6 different network profiles under the following bandwidths: 1Mbps, 5Mbps and 20Mbps. We experiment with two round trip delays with means 50ms and 150ms. The delays are varied according to a normal distribution. We inject up to 2% packet loss rate based on real world studies [19] and the losses are chosen from a random distribution.

	lab_WiFi	lab_3G	lab_4G
Average	b20-d50	b1-d50	b5-d50
Poor	b20-d150	b1-d150	b5-d150

Table 1: Network profiles used in the experiments. "b" stands for bandwidth in each direction and "d" stands for mean round trip delay. For example, an *Average lab_WiFi* is denoted by *b20-d50* and stands for a network whose bandwidth is 20Mbps and whose mean round trip delay is 50ms.

Table 1 shows the different network profiles used in our experiments. The network profiles are named after typical WiFi, 3G, and 4G conditions for ease of exposition. We map the profiles to the closest network condition based on lab experiments.

Webpages: We experiment with 200 Web pages. We randomly chose 40% of the Web pages from the top 200 Websites in Alexa [1], 30% from the pages from the bottom of Alexa's 1 million Websites, and the remaining 30% from news Websites on Alexa. We choose a mix of Web pages for the following reason: typically the popular top 200 Web pages on Alexa are smaller (for example, google.com) and are highly optimized. The performance of such Web pages may not be typical. Instead, we include unpopular pages in our mix because they are likely to not be optimized. We also choose news Websites because they tend to be complex pages.

In the common case, we load the original page on both the desktop and the mobile browser. We perform addition experiments where we load the mobile version of the page, that we call *mpage*. For example, `m.cnn.com` is an mpage, where the original page is `www.cnn.com`. By default, mobile browsers always redirect to the *mpage*. We modify the user agent field to force the mobile browser to load the original page. We do this to directly compare the performance differences between the mobile and the desktop browser.

Metrics: We measure page load performance using the Page Load Time (PLT) metric. The Page Load Time metric is commonly defined as the time between when the page is requested and when the *DOMLoad* event is fired [31]. The DOMLoad event is fired when all objects are fetched, processed, and added to the DOM. There has been several alternate metrics to define page load performance such as the above-the-fold metric [15]. However, these alternate metrics are not easy to compute and are not yet widely used.

Limitations of the testbed: While our testbed allows us to perform controlled experiments, there are several limitations. First, the PLT metric does not take into account scrolling speeds, refresh rates, and other user-perceived Quality of Experience metrics. These are outside the scope of this work. Second, the page load performance is not only affected by the network bandwidth and delay, but also by network variance. By controlling the network, we lose the network variances seen in real environments. However, there is a trade-off between doing controlled experiments and doing experiments that are close to the real world. In this work, we lean towards controlled experiments so that we can perform meaningful comparisons between the desktop and the mobile browser.

Finally, our findings are specific to the version of the Chromium browsers we choose. We believe that the findings from this work will apply broadly and can help influence future browser versions.

5. COMPUTATION IS THE BOTTLENECK

In this section, we present the page load times and bottlenecks in loading pages on the desktop and the mobile browser. We load both the original version of the page and *mpages* under all the network profiles for all Web pages, as discussed in §4.

Our key findings are as follows:

- On the mobile browser, on almost all the network profiles starting from *Average lab_WiFi* to *Poor lab_3G*, the time spent on computation activities on the critical path is significantly higher than on network activities. On the desktop browser, the phenomenon is reversed; irrespective of the network profile, almost always network activities are more of a bottleneck.
- Computation is the key bottleneck even on newer Samsung Galaxy S6 phones with arguably better computational capacity.
- Our results are not specific to our controlled experimental testbed. When pages are loaded *in-the-wild*, i.e., from the original Web server on real WiFi links, computation is still the bottleneck during page load on the mobile browser.

5.1 Page load times

Figure 4 shows the page load times to load pages on the desktop browser and the mobile browser. In the median case, page load times are two times slower on the mobile browser compared to the desktop browser, even when both browsers see similar restrictions in bandwidth and delays. The difference holds even in the top 20% of the cases, not shown in the figure because of the long tail. Note that the same pages are being loaded under the same network conditions; the changes in page load performance are largely because of the mobile versus the desktop device/browser.

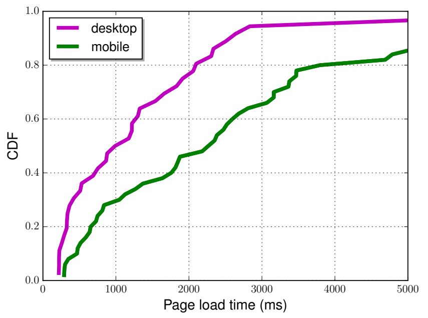
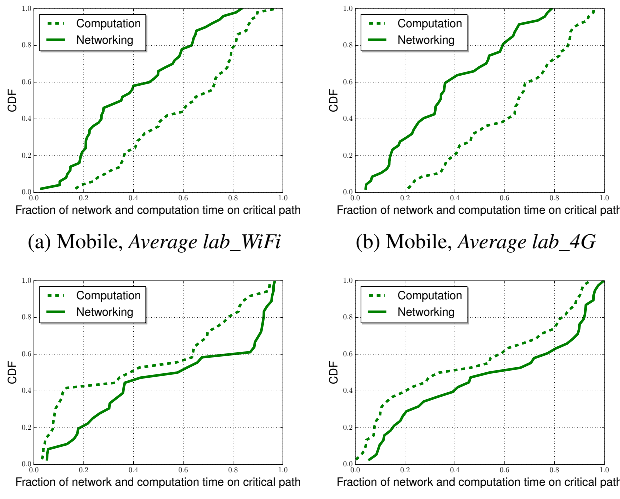


Figure 4: Page load times when loading original pages on the mobile browser and the desktop browser. Results for *Average lab_WiFi*.



(a) Mobile, *Average lab_WiFi* (b) Mobile, *Average lab_4G*

(c) Desktop, *Average lab_WiFi* (d) Desktop, *Average lab_4G*

Figure 5: Fraction of network and computation time on critical path when loading pages on the *Average lab_WiFi* and the *Average lab_4G* networks. Under average connectivity, computation activities is more of a bottleneck for the mobile browser, while network activities is the primary bottleneck for the desktop browser.

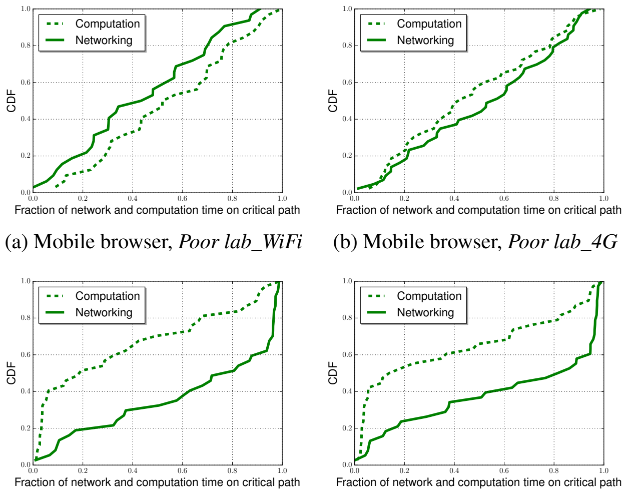
5.2 Bottleneck in mobile versus desktop browsers

Figure 4 showed significant difference in page load performance between the mobile and the desktop browser even when pages are loaded under similar conditions. To understand this further, we study the bottleneck.

Figure 5 shows the fraction of computation and network activities on the critical path when loading pages on the mobile and the desktop browsers. The results are for average network conditions, under *Average lab_WiFi* and *Average lab_4G* networks. The definition of computation and network activities can be found in §3.3 and the network profiles are specified in §4.

Figure 5a and 5b show the fraction of computation and network activities when loading the page on the mobile browser for the two network profiles. In both cases, in the median case, computation activities occupy more than 60% of the critical path. Network activities occupy less than 40% of the critical path.

Figures 5c and 5d show that, in contrast, on the desktop browser, network activities occupy 60% of the critical path. The results for the *Average lab_3G* network is quantitatively similar (not shown here).



(a) Mobile browser, *Poor lab_WiFi* (b) Mobile browser, *Poor lab_4G*

(c) Desktop browser, *Poor lab_WiFi* (d) Desktop browser, *Poor lab_4G*

Figure 6: Fraction of network and computation time on critical path when loading pages on the *Poor lab_WiFi* and the *Poor lab_4G* networks. Even under poor network connectivity, computation activities is the primary bottleneck for the mobile browser. Under poor connectivity, network activities becomes even more of a bottleneck for the desktop browser.

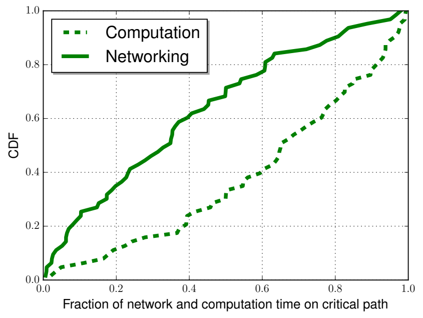
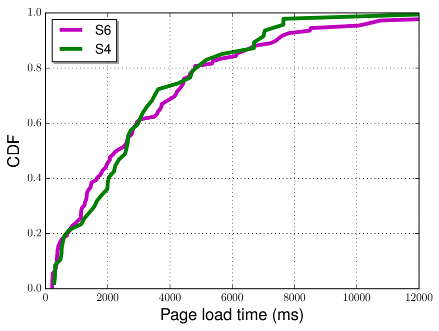
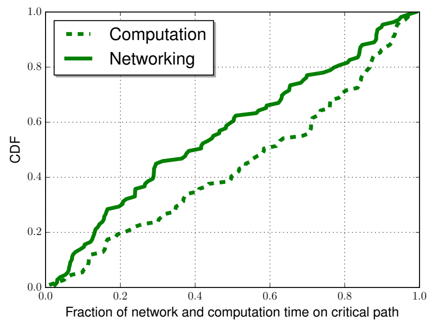


Figure 7: Fraction of network and computation time on critical path when loading *mpages* on the *Average lab_WiFi* network.

Next, we look at poor network environments. Figure 6 shows the fraction of computation and network activities on the critical path, when the mean round trip delay is 150ms. On the mobile browser, Figure 6a shows that computation activities continue to be the main bottleneck for page load, accounting for 55% of the critical path in



(a) PLTs when loading Web pages on Samsung Galaxy S4 and S6 phones. The pages were loaded on the *Average lab_4G* network.



(b) Fraction of network and computation time on critical path when loading pages in Samsung Galaxy S6 phones on the *Average lab_4G* network.

Figure 8: Results from experiments on Samsung Galaxy S6

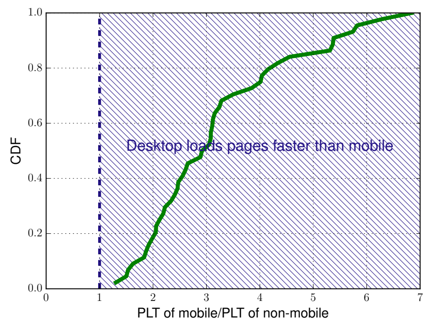


Figure 9: The ratio of page load times on the mobile browser versus the desktop browser, when loading pages *in-the-wild*.

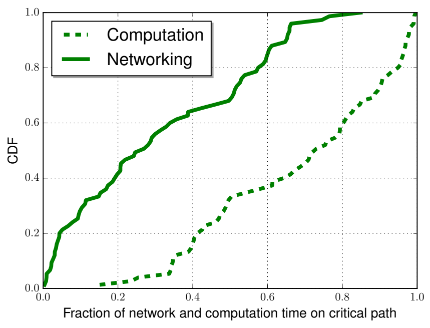


Figure 10: Fraction of network and computation time on critical path when loading pages on mobile browser *in-the-wild*.

the median case. It is only in the *Poor lab_4G* environment that the network activities become more of a bottleneck compared to computation activities. As the network conditions worsen, network activities start becoming the primary bottleneck during page load.

On the other hand, on the desktop browsers the poor network condition only makes the bottleneck due to network activities more pronounced (Figures 6c and 6d).

5.3 Bottleneck when loading mpages

Figure 7 shows the bottlenecks when loading *mpages* on the mobile browser. Continuing the trend, we find that when loading *mpages*, computation activities are the primary bottleneck. In the median case, 65% of the critical path constitutes computation activities, and only 35% of the critical path constitutes network activities.

Mobile pages such as `m.cnn.com` are smaller versions of the original `cnn.com` page, with smaller object sizes. Therefore, the network activities for loading an *mpage* should not take as long compared to loading the original page. Our analysis of the number of bytes downloaded when loading an *mpage* versus loading the original page confirms this hypothesis (§6.3). However, *mpages* do not significantly reduce the computation activities required to load the page. Therefore, it is not surprising that the computation activities continue to be the bottleneck when loading *mpages*.

5.4 Experiments on Samsung Galaxy S6

We repeated the controlled page load experiments on Samsung Galaxy S6. The goal of this experiment is to observe if the bottleneck due to computation activities in mobile browsers are a function of a weaker processor. The S6 phone has four ARM Cortex-A57 cores clocked at 2.1GHz, and four Cortex-A53 cores clocked at 1.5GHz, compared to the quad-core, 1.7GHz Samsung S4 phones. First we look at the page load time differences: Figure 8a shows that the page load times on S6 is not significantly different compared to the page load times on S4.

As before, we next look at the bottlenecks in S6. Figure 8b shows that computation activities is still the primary bottleneck, constituting 62% of the critical path in the median case. There are two possible reasons for this result. Either, the browsers are not able to use the additional CPU capacity effectively, or the increase in CPU is not enough to change the bottleneck. We will explore this further as part of future work.

5.5 Experiments in-the-wild

Finally, we load web pages on the desktop and the mobile browser outside of our experimental testbed. The web pages are fetched from the Web server. The desktop browser uses the campus Ethernet connection (bandwidth 250Mbps), while the mobile browser uses the campus WiFi connection (bandwidth 30Mbps). The goal of this experiment is to study if the observations we make in the controlled setting also hold true in general.

We first perform a pairwise comparison of loading pages on the desktop versus the mobile browser. Figure 9 shows the ratio of the time to load the page on the mobile browser versus the desktop browser. The line $x = 1$ shows the points when the page load times of the mobile and the desktop browsers are equal; all points to the right show cases when the desktop browser is faster. Figure 9

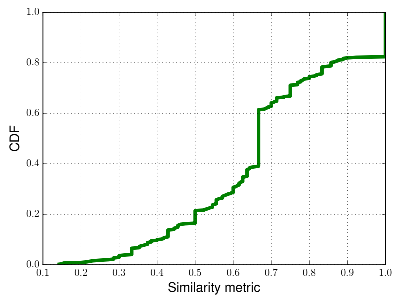


Figure 11: Similarity metric: the fraction of time the same <URL, activity> pair occur on the critical path when loading the page on the mobile browser and the desktop browser. Each activity is associated with a unique URL. Results from all network profiles.

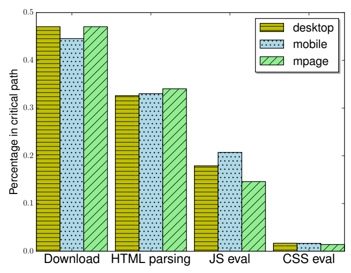


Figure 12: Looser similarity metric: the fraction of time an activity occurs on the critical path when loading the page on the mobile and the desktop browser. The URL associated with the activity may be different. Results from all network profiles.

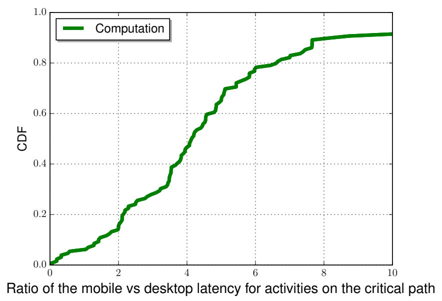


Figure 13: Ratio of the time taken to complete computation activities on the mobile browser versus the desktop browser. The activity should be on the critical path of at least one of the two browser loads. Pages loaded on the *Average lab_WiFi* network.

shows that in all cases, the desktop browser is faster than the mobile browser to load the same page. In the median case, loading a page on the mobile browser is three times as slow compared to the desktop browser. Note that these experiments are not performed in a controlled setting. So the difference in performance between loading the pages on the desktop browser versus the mobile browser may be due to several factors.

Figure 10 shows the fraction of computation versus network activities on the critical path when loading pages on the mobile browser *in-the-wild*. We find that computation activities are even more of a bottleneck; in the median case, close to 70% of the critical path is composed of computation activities. On the desktop browser, as before, we find that network activities continues to be the primary bottleneck (not shown here). Our *in-the-wild* experiments show that our observations about the mobile and the desktop browser bottlenecks holds true more generally, and is not specific to our testbed.

6. CRITICAL PATH ANALYSIS

In the previous section, we looked at the differences in critical path between the mobile and the desktop browser at a macro-level. Next, we study the critical paths at a more micro-level. Our key findings are as following:

- Even when loading the same page on the same network profile, the critical path on the mobile browser and on the desktop browser are different in terms of the exact objects downloaded, the Javascript executed, etc. However, in terms of the fraction of page load activities on the critical path, the two critical paths are similar.
- For each computation activity on the critical path, it takes 4 times as long to perform the activity on the mobile browser compared to the desktop browser.
- On the mobile browser, of all the computation activities, HTML parsing is the most dominant on the critical path; rendering activities are the least dominant.

6.1 Comparing critical paths on the desktop and the mobile browser

The critical path consists of a series of activities such as HTML loading, HTML parsing, downloading images, JavaScript, and CSS, and evaluating the CSS and Javascript (see Figure 2). Going one level deeper, each of the activities are associated with a unique URL. For example, two activities that are both downloading an image are downloading different images associated with different URLs.

Our goal is to study the similarity between the critical path when loading a page on the desktop versus the mobile browser. We define the similarity metric as follows: the fraction of time the same <URL, activity> pair occurs on the critical path when loading the page on the desktop browser and the mobile browser.

Figure 11 shows the similarity metric across all network profiles. Even when the same page is being loaded under the same network profile, the critical path is identical only for 20% of pages. For another 20% of the pages, only 50% of the critical path is similar. This result has implications for optimization. It shows that optimizing a specific activity, such as making the content of a specific Javascript object smaller, may not have the same effect on the mobile browser as they would on the desktop browser.

Next, we relax the definition of similarity and look at the percentage of time the same activity occurs on both the critical paths, but not necessarily associated with the same URLs. For example, if the desktop critical path contains the image download activity for, say downloading *image1*, but the mobile critical path contains another image download activity, for a different image, *image2*, we consider them to be similar. Note that, in Figure 11, these two activities will be considered dissimilar because they are not downloading the same image.

Figure 12 shows the percentage of each activity on the critical path across all pages and network profiles. For every page, the activities on the critical path in terms of Javascript evaluation, HTML parsing etc are similar on the desktop and the mobile browser. One of the implications of this result is that optimizations that target a class of activities, such as reducing the time to download all Javascript objects, are likely to provide benefits across mobile and desktop browsers.

6.2 Comparing the latency for each activity on the critical path

Figure 12 showed that in terms of the number of the activities on the critical path, the desktop and the mobile browser are similar, even though the URL of the activities may be different (Figure 11). However, we know that the length of the critical path (i.e., the PLT) is much smaller for the desktop browser compared to the mobile

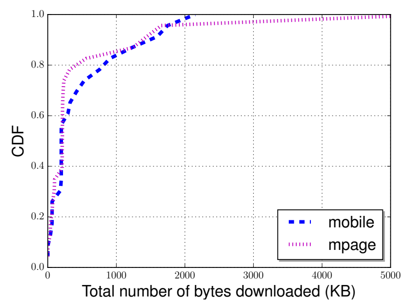


Figure 14: Total bytes downloaded when loading the original versus *mpage* on the mobile browser. Results from all network profiles.

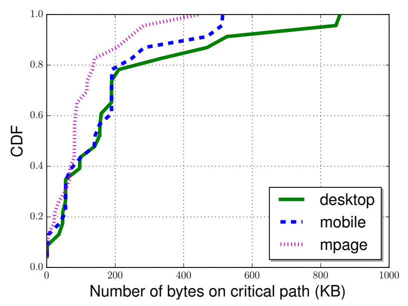


Figure 15: Bytes downloaded on the critical path when loading the original page on the desktop and the mobile, and loading *mpages*. Results from all network profiles.

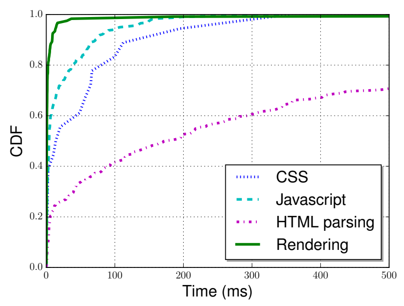


Figure 16: The breakdown of the computation activities on the critical path. Results from experiments on the S6 phone on the *Average lab_WiFi network*.

browser (Figure 4). To understand this, we measure the time taken to complete an activity on the mobile versus the desktop browser.

We consider all activities that are on the critical path of at least one of the two browsers. We then compare the time taken for each of these activities to complete on the mobile and the desktop browser. Since we load the exact same page, each activity has to be performed both by the desktop and the mobile browser.

Figure 13 shows the time take for each computation activity to be performed on the mobile versus the desktop browser. Each activity takes over 4 times longer to perform on the mobile browser compared to the desktop browser. This is one of the reasons for the mobile browser performance to be much worse compared to the desktop browser performance even under similar network conditions.

Surprisingly, the time taken for network activities is also smaller on the desktop browser compared to the mobile browser (not shown in figure). In 40% of the cases, the difference is insignificant, but for the remaining 60%, the difference is higher, even though the pages are loaded under the same network conditions. This can possibly be because of a less optimized network stack on the mobile browser compared to the desktop browser. We will investigate this further as part of future work.

6.3 The breakdown of network and computation activities on the critical path

Figure 14 shows the total bytes downloaded when loading the original page and loading the *mpage* on the mobile browser. Even though *mpages* are designed to be smaller, for 60% of the pages, the total objects downloaded remains the same. But for 30% of the pages, the difference in size is over 60%. The results suggests that *mpages* significantly reduces the size of a small number of pages, but for a large fraction of pages, there is not much difference between the *mpage* and the original page.

Figure 15 shows the bytes downloaded on the critical path when loading the original page on the desktop and the mobile browser, and when loading *mpage*. Here we find that *mpages* do reduce the number of objects loaded in the critical path for over 40% of the pages. In a separate experiment (not shown here), we find that loading *mpages* does not reduce the computation time on the critical path.

With respect to computation, Figure 16 shows the breakdown of the various computation activities on the critical path. HTML parsing is the dominant activity on the critical path, far outweighing the time spent on Javascript and CSS evaluation. Rendering only occupies a small part of the critical path, as also shown by other researchers [33].

7. EFFECT OF PAGE TYPE AND NETWORK

In this section, we present our results on the effect of page type and network on the page load time. Our key findings are as follows:

- For page sizes of less than 250 KB, there is no significant difference in performance between the mobile and the desktop browser. For larger pages, there is significant performance difference.
- Under poor bandwidth conditions, changes in round trip delays does not affect page load times. Similarly, when the round trip times are high, bandwidth does not have significant effect on page load times.
- The desktop browser is able to use good network conditions better than the mobile browser.

7.1 Effect of page type

In this section, we look into desktop and mobile page load times based on the page size. In our experiments, 25% of analyzed Web pages had size less than 250KB. We define these Web pages as *small* pages and the rest as *large* pages.

Figure 17 shows that there is no significant difference in page load time when loading small pages on the mobile versus the desktop browser. But for large Web pages, the page load times on the mobile browser is almost twice as high compared to the desktop browser in the median case. The results were similar for other network settings.

Figure 18 shows the time spent on network activities on the critical path to load the small versus large pages. For both small and large pages, the time spent on network activities on the critical path is not very different on the mobile and the desktop browser.

Figure 19 shows the time spent on computation activities on the critical path to load small versus large pages. In this case, as before, for small pages, the computation time on the critical path is not significantly different when comparing page loads on the desktop and the mobile browser. But for the larger pages, we find that the computation time when loading the page on the mobile browser is almost two times that of loading the page on the desktop browser in the median case.

In effect, the difference in page load performance when loading pages on the mobile versus the desktop browser is more pronounced for larger pages. This difference is because of the addi-

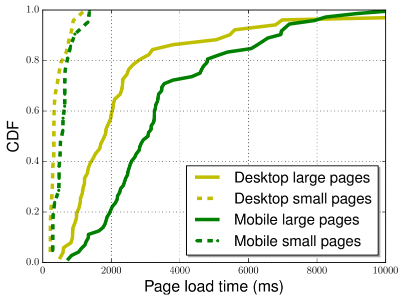


Figure 17: Page load time for *small* and *large* Web pages in mobile and desktop browsers on the *Average lab_WiFi* network.

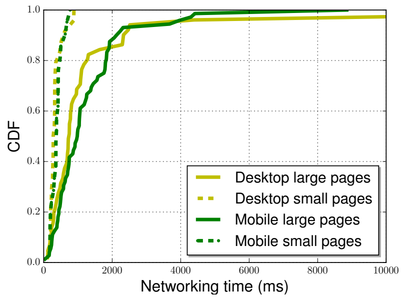


Figure 18: Time spent on network activities on the critical path to load *small* and *large* Web pages on the *Average lab_WiFi* network.

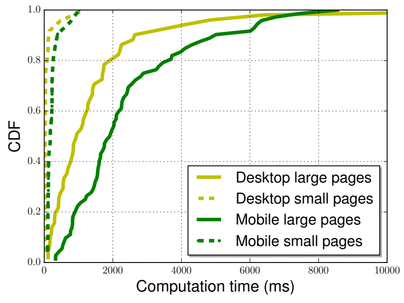


Figure 19: Time spent on computation activities on the critical path to load *small* and *large* Web pages on the *Average lab_WiFi* network.

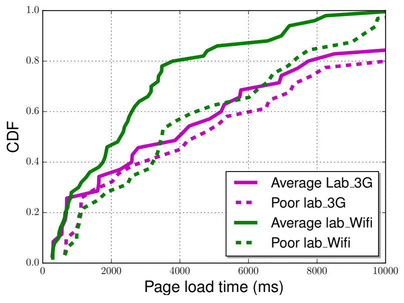


Figure 20: Page load time under *Average lab_3G*, *Poor lab_3G*, *Average lab_WiFi*, and *Poor lab_WiFi* networks when loading pages on the mobile browser.

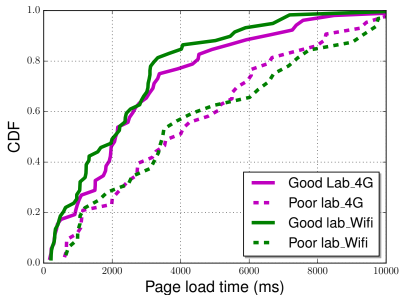


Figure 21: Page load time under *Good lab_4G*, *Poor lab_4G*, *Good lab_WiFi*, and *Poor lab_WiFi* network profiles when loading pages on the mobile browser.

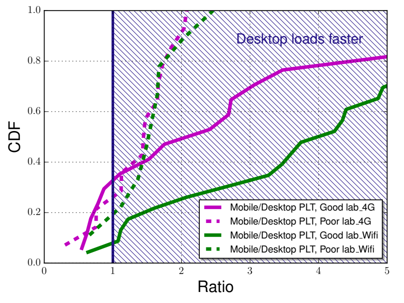


Figure 22: The ratio of the page load time when loading pages on the mobile and the desktop browser under *Good lab_4G*, *Poor lab_4G*, *Good lab_WiFi*, *Poor lab_WiFi* network.

tional time taken by the mobile browser to complete computation activities on the critical path.

7.2 Effect of network

Figure 20 shows the page load times under various network conditions on the mobile browser, starting from *Average lab_WiFi* to *Poor lab_3G*. When the bandwidth is poor, as in the case of *lab_3G* (with 1Mbps bandwidth), round trip times does not affect the page load time; for instance, the page load time under *Poor lab_3G* (RTT = 150ms) and *Average lab_3G* (RTT = 50ms) is similar.

Figure 21 shows that, similarly, when the network experiences high RTTs, the bandwidth does not affect page load time. Under high RTT of 150ms, both *lab_WiFi* (bandwidth = 20 Mbps) and *lab_4G* (bandwidth = 5 Mbps) perform similarly. In these graphs, we define *Good lab_Wifi* and *Good lab_4G* to be the same as *Average lab_WiFi* and *Average lab_4G* networks, respectively, in terms of bandwidth, but with mean round trip delays of 5ms.

Finally, Figure 22 shows that ratio of the page load time on the mobile browser and the desktop browser for different network profiles. Points to the right of $x = 1$ line shows the area where the desktop browser is faster. The difference between the desktop browser and the mobile browser is highest for the *Good lab_WiFi* network, which is the network with the best bandwidth and round trip times across our network settings. In this setting, the page load time on the mobile browser is 3.5 times higher than the desktop browser in the median case. As the network conditions worsen, as in the case of *Poor lab_4G*, the difference in performance between the mobile and the desktop browser decreases. One possible hypothesis is that the desktop browser is able to utilize good network conditions much better than the mobile browser.

8. RELATED WORK

In this section, we will discuss related literature on mobile browsers. We will also discuss literature on desktop browsers when applicable, since mobile browsers have not yet seen the kind of extensive research as compared to desktop browsers.

Improving mobile browsers: Erman et al. [20] show that unlike desktop browsers, optimizations such as SPDY/HTTP2 does not improve performance of Web pages on mobile browsers. They show that this is because of the negative interactions between the cellular state machine and the transport protocol. Similarly, Qian et al. [27] show that caching does not provide page load improvements for mobile browsers.

Many of the research on explicitly improving mobile browser performance has seen mixed results. Flywheel [16] is Google's compression proxy that compresses web content to significantly reduce the use of expensive cellular data. The authors note that while FlyWheel succeeds in reducing data usage, its effect on page load performance is more mixed; it helps performance of certain pages and hurts performance of others. FlexiWeb [29] is built over Google's compression proxy to ensure that the proxy does not hurt page load times. But FlexiWeb is not designed to explicitly improve page load performance. Wang et.al [34] show that speculative loading is one of the only client-only approaches that can improve mobile browser performance. However, speculative loading requires knowledge of what objects are likely to be requested by the user.

Other research works have looked at metrics orthogonal to the page load time metric. Parcel [30] uses a proxy approach to divide the page load process between the mobile device and the proxy. Because Parcel is a network approach, the evaluations are largely with respect to reduction in network latencies. Klotski [17] focusses on increasing the number of objects rendered in the first 5 seconds to improve user Quality of Experience.

While there has been several recent efforts on improving mobile browser performance, they have not been uniformly successful. In our work, we are studying the fundamental bottlenecks in mobile browsers as a first step towards designing effective mobile Web optimizations. Importantly, our goal is to compare the page load performance between the desktop and the mobile browser, to understand why and how mobile browsers differ from desktop browsers.

Measurement Studies: The browser study presented by Wang et.al [33] in 2011 is the closest to our measurement set up. Similar to our set up, their work also uses an instrumented Webkit and extracts dependency relations in the browser load process. Our findings are largely consistent with the *What-If* analysis presented in the paper [33] for a small set of 10 Web pages. Notably, they find that rendering does not significantly contribute to browser delays, and a more powerful operating system can improve browser performance.

Different from our findings, their work finds that higher bandwidth does not effect performance because Web page sizes are small. However, Web pages have become more complex now compared to 2011 [36]; as a result, we find that for current Web pages, higher bandwidth does improve page load times. Our work expands on this study by performing a more extensive dependency analysis for a larger number of pages. Importantly, we enable pairwise comparisons between mobile and desktop browsers to understand the critical differences between the two.

Qian et al. [28] provide one of the first detailed measurement study of mobile browsers, but focus on cellular data and energy usage rather than page load times. Zaki et. al [38] measure the mobile browser performance in Ghana, focussing only on the network delays. They find that the network delays in Ghana are largely caused by DNS, HTTP redirections, and SSL handshake.

Industry tools: All major browsers, including Chrome, Firefox, and Internet explorer, provide developer tools to study browser performance [5, 21, 3]. Many of these tools are designed for desktop browsers, but they are starting to be developed for mobile. While the developer tools provide network timings, they ignore the timings for the computational activities, and cannot be used to construct the critical path [31]. New tracing tools such as `chrome://tracing` [13] provide more fine-grained information about chrome internals, but again, do not have enough information to extract the critical path.

Google Octane [7] benchmarks the performance of the Javascript engine of a browser by running a suite of tests. Browsermark [2] is a browser benchmarking tool which helps a user to decide which browser offers the best Web experience. We, on the other hand, are focused on characterizing the bottlenecks during mobile page loads. We believe that once the bottlenecks are identified, each of the above mentioned benchmarking tools can play a role in improving page load performance.

Measurement platforms such as WebPageTest [35] and HTTP Archives [22] allow researchers to perform Web page measurements from several vantage points. They provide measurement data from a large number of networks and devices. However, we still lack tools to directly compare the performance of desktop and mobile browsers, or to analyze the critical path.

Finally, Google's PageSpeed Insight [8] and Yahoo's YSlow [37] are industry tools that take a Webpage URL as input and suggest optimizations. The tools apply static rules to the page. For example, if an object is not compressed, the tool may suggest that the object be compressed, even if loading the object is not the bottleneck. Analyzing page load bottlenecks can greatly improve the optimization suggestions of these tools.

Testbeds: In terms of experimental testbed, Mahimahi [26] is the closest to our work. Mahimahi uses an HTTP-based record and replay tool for repeatable page load experiments on desktop browsers. Our goals on repeatability are similar. But we focus on studying the critical path, and therefore record the page load time at the object level rather than at the HTTP level.

WebProphet [23] was one of the first works to discuss dependencies for desktop browsers. WebProphet uncovered dependencies in desktop browsers, assuming the computational activities to be a black box. Our previous work WProf [31], improved over WebProphet by uncovering both network and computation dependencies. In this work, we port WProf to a mobile browser.

9. IMPLICATIONS/CONCLUSIONS

In this work, we perform an in-depth study of bottlenecks in mobile browsers vis-a-vis desktop browsers. To identify the low level page load activities and extract the bottlenecks when loading a mobile page, we leverage our past work to build WProf-M, an in-browser profiler for Android Chromium. We build an experimental testbed that allows us to directly compare the low-level page load activities and bottlenecks when loading a page on a mobile versus a desktop browser. We use the testbed to better isolate the differences in page load performance caused by the mobile device and the mobile browser.

Our results have several implications for mobile browser optimization. First we find that computation activities are the main bottleneck is mobile browsers. This is in contrast to desktop browsers where network activities are the main bottleneck. This has implications for mobile browser optimizations which have focussed more on reducing the networking latency during page load, and not enough on reducing the computational bottlenecks. Second, we find that the activities on the critical path can be different when loading pages on the mobile browser compared to the desktop browser. Therefore, optimizing a specific object on the critical path may not provide similar benefits on the desktop and the mobile browser. Finally, smaller pages do not see significant difference in performance when loading on the mobile versus the desktop browser. Therefore, we should focus on optimizing larger pages.

Acknowledgements

First, a special thanks to University of Washington undergraduate student Tyler Jacoby who helped a great deal in porting WProf to the Android version. Thank you to Xiao Sophia Wang for providing helpful feedback and comments during the porting process. This work was supported by the National Science Foundation, through the grant CNS-1551909. Sincere thanks to Stony Brook Masters students Naveen Nuthalapati (for help with experiments) and Pavan Muguluri (for help with debugging the WProf visualization). We gratefully acknowledge the anonymous reviewers.

10. REFERENCES

[1] Alexa. `http://www.alexa.com/`.

[2] Browsermark. `http://web.basemark.com/`.

[3] Chrome developer tools. `https://developer.chrome.com/devtools/`.

[4] Ericsson mobility report june 2015: `http://www.ericsson.com/res/docs/2015/ericsson-mobility-report-june-2015.pdf`.

[5] Firefox Developer Tools. `https://developer.mozilla.org/en-US/docs/Tools`.

[6] Google developers: Analyzing critical rendering path performance. `https://developers.google.com/web/fundamentals/performance/critical-rendering-path/?hl=en`.

[7] Google Octane. `https://developers.google.com/octane`.

[8] Google Pagespeed Insights. `https://developers.google.com/speed/pagespeed/insights`.

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

[10] Mobile-only users surpass desktop-only users. `http://marketingland.com/mobile-only-users-surpassed-pc-only`.

[11] Network Namespace. `http://blog.scottlowe.org/2013/09/04/introducing-linux-network-namespaces`.

[12] SPDY. `https://www.chromium.org/spdy/spdy-whitepaper`.

[13] The Trace Event Profiling Tool . `https://www.chromium.org/developers/how-tos/trace-event-profiling-tool`.

[14] W3C: Document Object Model. `http://www.w3.org/DOM/`.

[15] Above the fold time. `http://www.webperformancetoday.com/`.

[16] 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 *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, Berkeley, CA, USA, 2015. USENIX Association.

[17] M. Butkiewicz, D. Wang, Z. Wu, H. V. Madhyastha, and V. Sekar. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, Oakland, CA, 2015. USENIX Association.

[18] X. Chen, A. Jindal, N. Ding, Y. C. Hu, M. Gupta, and R. Vannithamby. Smartphone background activities in the wild: Origin, energy drain, and optimization. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, MobiCom '15, pages 40–52, New York, NY, USA, 2015. ACM.

[19] N. Dukkipati, M. Mathis, Y. Cheng, and M. Ghobadi. Proportional rate reduction for tcp. In *Proceedings of the 11th ACM SIGCOMM Conference on Internet Measurement 2011, Berlin, Germany - November 2-4, 2011*, 2011.

[20] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan. Towards a spdy'ier mobile web? In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 303–314, New York, NY, USA, 2013. ACM.

[21] Firebug. `http://getfirebug.com/`.

[22] HTTP Archive. `http://httparchive.org/`.

[23] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. Greenberg, and Y.-M. Wang. WebProphet: automating performance prediction for web services. In *Proc. of USENIX NSDI, 2010*.

[24] L. A. Meyerovich and R. Bodik. Fast and parallel webpage layout. In *Proc. of the international conference on World Wide Web (WWW), 2010*.

[25] J. Mickens. Silo: Exploiting JavaScript and DOM Storage for Faster Page Loads. In *Proc. of USENIX conference on Web Application Development (WebApps), 2010*.

[26] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan. Mahimahi: Accurate Record-and-Replay for HTTP. In *USENIX Annual Technical Conference 2015*, Santa Clara, CA, July 2015.

[27] F. Qian, K. S. Quah, J. Huang, J. Erman, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Web caching on smartphones: Ideal vs. reality. In *MobiSys '12*, pages 127–140, New York, NY, USA, 2012.

[28] F. Qian, S. Sen, and O. Spatscheck. Characterizing resource usage for mobile web browsing. MobiSys '14, pages 218–231, New York, NY, USA, 2014. ACM.

[29] S. Singh, H. V. Madhyastha, S. V. Krishnamurthy, and R. Govindan. Flexiweb: Network-aware compaction for accelerating mobile web transfers. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, MobiCom '15, pages 604–616, New York, NY, USA, 2015. ACM.

[30] 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 *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, New York, NY, USA, 2014. ACM.

[31] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. Demystify page load performance with wprof. In *Proc. of the USENIX conference on Networked Systems Design and Implementation (NSDI)*, 2013.

[32] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. How speedy is spdy? In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 387–399, Berkeley, CA, USA, 2014. USENIX Association.

[33] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie. Why are web browsers slow on smartphones? In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, pages 91–96. ACM, 2011.

[34] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie. How far can client-only solutions go for mobile browser speed? In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 31–40, New York, NY, USA, 2012. ACM.

[35] WebPagetest. `http://www.webpagetest.org/`.

[36] WebsiteOptimization. Web growth, 2014.

[37] YSlow. `http://yslow.org/`.

[38] Y. Zaki, J. Chen, T. Pötsch, T. Ahmad, and L. Subramanian. Dissecting web latency in ghana. In *Proceedings of the Internet Measurement Conference (IMC). Vancouver, Canada, November 2014.*, 2014.