WebTune: A Distributed Platform for Web Performance Measurements

Byungjin Jun* Matteo Varvello† Yasir Zaki‡ Fabián E. Bustamante*

*Northwestern University †Nokia Bell Labs ‡NYU Abu Dhabi

***Abstract*—Web performance researchers have to regularly choose between synthetic and in-the-wild experiments. In the one hand, synthetic tests are useful to isolate *what* needs to be measured, but lack the realism of real networks, websites, and server-specific configurations. Even enumerating all these conditions can be challenging, and no existing tool or testbed currently allows for this. In this paper, as in life, we argue that *unity makes strength*: by sharing part of their experimenting resources, researchers can naturally build their desired realistic conditions without compromising on the flexibility of synthetic tests. We take a step toward realizing this vision with WebTune, a distributed platform for web measurements. At a high level, WebTune seamlessly integrates with popular web measurements tool like Lighthouse and Puppeteer exposing to an experimenter fine grained control on real networks and servers, as one would expect in synthetic tests. Under the hood, WebTune serves "Webtuned" versions of websites which are *cloned* and distributed to a testing network built on resources donated by the community. We evaluate WebTune with respect to its cloning *accuracy* and the *complexity* of network conditions to be reproduced. Further, we demonstrate its functioning via a 5-nodes deployment.**

***Index Terms*—Web, performance, testbed**

I. Introduction

The web is a complex ecosystem with wide range of webpage designs, servers' and clients' configurations, and network settings. Web content is also constantly changing, causing significant differences among webpage loads. This heterogeneity and dynamism are problematic to both researchers and practitioners as they complicate experimentation and cast doubts on their conclusions.

Content providers in control of their hosting servers can explore the effect of different clients' network configurations with web performance tools such as WebPageTest [1], Lighthouse [2], or Puppeteer [3]. Experimenters comparing the performance of different server configurations, on the other hand, find themselves between a rock and a hard place. They can use synthetic webpages on their servers, coming to terms with the fact that their findings may not apply to real-world webpages, which are highly heterogeneous. Alternatively, they can opt for actual webpages and avoid dynamism by recording/replying web access traces [4], but have to limit themselves to in-lab/synthetic network conditions.

None of the options available to experimenters today provides a clear and full picture needed to understand the impact of real sites configurations on the quality of experience of their clients, forcing experimenters to choose either realism or the necessary control to explore, and explain the parameter space. This frustrating state of affairs motivates the design of *WebTune*, a platform that enables experimentation with real-world webpages under real network conditions, without the uncertainty of dynamic content and opaque server-side networking stack.

The goal of WebTune is to reproduce, in a realistic but controlled environment, the entities involved in a webpage load. That is, the set of domains/servers responsible for a webpage are replaced with machines from a *testing network* instrumented to serve such domains. To achieve this, WebTune adopts a cooperative approach where experimenters make hosting servers with their particular connectivity available in the testing network. When deploying an experiment, WebTune selects among the available nodes/servers and instrument them, e.g., by setting the desired TCP version (e.g., CC algorithms), to meet an experimenters' needs.

From an experimenter's perspective, testing a real or a "webtuned" page is equivalent. Web performance tools can be used unmodified by simply prefacing a request to WebTune's API describing the experiment, e.g., *test cnn.com while fully hosted in the US by servers which use the BBR congestion control* (see Figure 3). This request triggers WebTune's *access server* to fully load the desired webpage in Chrome, cache all HTTP(S) headers and bodies, and distribute them to the chosen nodes in the testing network which will be serving them. Finally, the experimenter is provided with a Proxy Auto-Configuration (PAC) file – generated to reflect the nodes selection – which steers traffic towards the testing network transparently, *i.e.,* without any change in both cloned content and web performance tools.

We evaluate WebTune with respect to the *accuracy* of webpage cloning, and the *complexity* of the networking conditions that should be reproduced. Out of Alexa's 100 most popular wepages (*top100*, from now on), we find that 80-90% of pages can be accurately cloned, both in term of their overall size and Structural SIMilarity (SSIM) when fully loaded by the Chrome browser. Some webpages are affected by the presence of dynamic content, which is to be expected and would cause differences even between two consecutive tests using the original webpage. A small percentage of webpages failed during cloning which triggers, for instance, quite visual 404 messages.

With respect to network settings, our results show that a testing network with 5-10 nodes is enough for the majority

978-3-903176-40-9

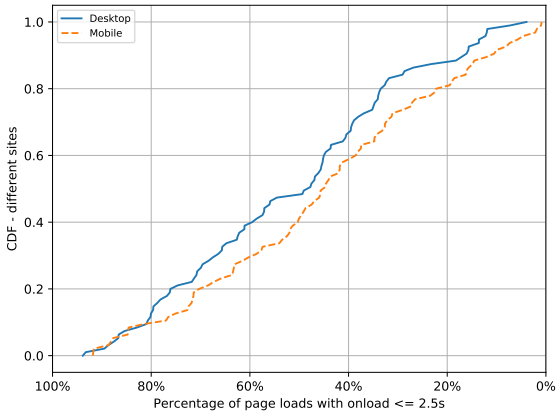


Fig. 1. CDF of percentage of page loads that are "fast", *i.e.,* within 2.5 seconds, as per Chrome UX Report for Alexa top 100 in desktop and mobile. This shows web performance varies ever for the most popular webpages.

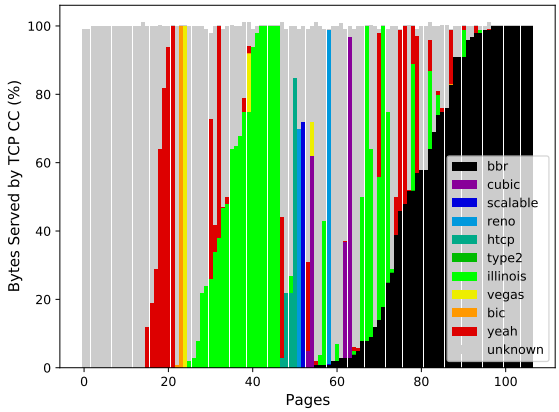


Fig. 2. Traffic contribution per TCP congestion control, as identified by Gordon [7] for Alexa top 100 webpages. In addition to the complex mix of congestion control, its unknown portion makes web performance measurement more opaque and challenging.

of the Alexa top100 pages. However, we also find that some webpages might require tens of nodes to match their more complex RTT distributions. This further motivates WebTune's crowdsourced design, which allows it to scale as more experimenters join. WebTune is open source [5] and can be used with *any* testing network, such as lab machines or cloud nodes.

We demonstrate WebTune's capabilities by exploring two research questions using, as an example, 5-nodes testing network composed of a combination of in-lab and AWS machines. First, we investigate the impact of adding 100% HTTP/2 support to 30 Alexa top100 webpages. Apart from the significant benefits, we show how WebTune's server-side view allows to further explain the results. Last, we use WebTune to quantify the benefits of adding CDN support to 15 Alexa top100 webpages which are still self-hosted.

II. Web Measurement Hazards

Background: Webpages are increasing in complexity, containing hundreds of elements hosted on a variety of domains and servers. Clients' network conditions and devices are also increasingly heterogeneous, with a myriad of new devices being released every year while legacy devices are still operational. Such diversity is a key reason for the significant performance differences we all experience as users.

Figure 1 quantifies these differences focusing on Alexa top100 sites (most popular in the US) plotting the Cumulative Distribution Function (CDF) of percentage of page loads that are *fast*, *i.e.,* within 2.5 seconds as measured by the millions of Google Chrome users in the wild and published in the Chrome User Experience Report (CRuX) [6]. We find that only 20 webpages, out of the top100, have consistently *good* load times on desktop (at least 75% of page loads). Given the expected differences for mobile clients, we plot these pages' metrics separately (dashed line). In this case, the number of pages with consistently good load times drops to just 13 (out of the top 100), even considering mobile-specialised pages, e.g., where client is redirected to `m.` subdomains. Such wide range of performance even for the most popular (and thus one would assume better tuned) webpages emphasises the challenges of measuring and tuning web performance.

Server-Side Opaqueness: There has been many efforts by private companies and researchers to build tools to accurately measure a webpage performance; Puppeteer [3] and Lighthouse [2] are two popular tools for Chromium-based browsers, WebPageTest [1] offers cloud-based tests, and Web Page Replay [8] and Mahimahi [4] focus on recording and replaying HTTP traffic to reproduce network communications.

These tools provide detailed *client-side* view, but no visibility on the *server-side*. Researchers often face an opaque server-side networking stack when measuring webpages in the wild. This complicates experimental analysis as there is a cornucopia of possible server configurations that could impact measurement results. Here we outline a *small* subset to highlight the difficulties faced by researchers.

TCP Congestion Control: While TCP CUBIC is widely used in the web, whether the server uses TCP HyStart or SlowStart can lead to significant changes in a webpage's performance [9]. Additionally, emerging TCP congestion control protocols, such as BBR [10], utilize a delay-based approach which can also lead to different performance results [11]. Understanding which congestion control algorithm a remote server deploys is an active research area which has led to several projects [7], [11], [12].

Figure 2 illustrate the diversity of TCP congestion control algorithms in the wild, showing the percentage of *traffic* carried by each congestion control algorithm involved in a Web page load, focusing on Alexa top 100 [13] sites. To obtained this figure, we paired Lighthouse with Gordon [7], a recent client-side tool for remote TCP version fingerprinting. The figure shows a complex mix of congestion control algorithms, each of which would interact differently with the performance measurement and the network conditions considered. More im-

portantly, the amount of *unknown* (gray barplots) is impressive. This happens because fingerprinting TCP congestion control in the wild is hard and very much dependent on the size of the available content [7], [11]. In other words, even the best-prepared researchers lack the tools to gain full visibility on their experiments.

HTTP/2 Prioritization Implementation: Each browser uses a slightly different prioritization scheme when requesting web objects. Further, HTTP/2 lets a server decide how to handle requests [14]. Given this freedom, not all servers support prioritization or they completely ignore the client-side hints [15]. This is frustrating for researchers measuring the performance of HTTP protocol versions in-the-wild and can lead to inconsistent results.

Server Load-Balancing: When the goal is to measure the performance of web content or protocols, a major roadblock is server-side stack configurations, especially load-balancing. It is possible to load the same webpage multiple times and be served by different machines. The only countermeasure is *DNS prefetching*, *i.e.,* pre-loading the testing machine with the DNS entries to be used in a test to ensure that at least, successive tests will contact the same set of IP addresses. Unfortunately, this approach does not help in presence of IP anycast, a popular solution adopted by some Content Delivery Networks (CDNs) to pair content requests to a near-by cache.

III. WebTune Design

In this section, we outline the design and functioning of WebTune. WebTune focuses on providing the best possible testing conditions for web performance research. To this end, WebTune must ensure (i) that the content served remains constant across runs such that content variability is no longer an uncontrolled variable, (ii) require no modification of today's web performance tools, (iii) provide the whole picture to users, with information from both client and server sides, and finally, (iv) ensure that the content served is as close as possible to the *base* webpage, so performance metrics for both versions remain within normal bounds.

Figure 3 shows a visual representation of WebTune. At a high level, it consists of three main components: experimenter, access server, and testing network. The *experimenter* (client) is a member of WebTune which runs *any* web measurement tool as described in Sec. II. The web measurement is equivalent to today's measurements with the caveat that the experimenter can further specify server-side configurations via WebTune's API. This can be easily paired, for instance, with Lighthouse as shown in Fig. 3. The *access server* is responsible for managing both testing network and undergoing experiments. The *testing network* is the ensemble of resources donated by experimenters which are used to serve *cloned* copies of real webpages to experimenters.

A. Testing Network

WebTune's testing network consists of resources donated by a community of Web experimenters. This is motivated by

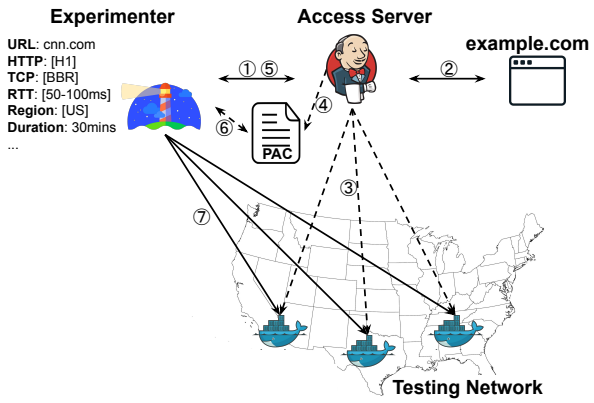


Fig. 3. Visual representation of WebTune and its workflow. First, an experimenter requests to test *example.com* over HTTP/1.1 and BBR as a transport; nodes should be in the US, with latency comprised between 50 and 100ms (1). The access server scrubs *example.com* for its content (2), which it then shares with serving nodes from the testing network selected given an experiment requirements (3). Next, it generates a PAC file reflecting the node selection (4). Finally, it returns to the client the PAC file it should use for this test (5). Consulting with the PAC file (6), a classic web experiment is then conducted (7) using, for example, Lighthouse [2].

the fact that, given the size and diversity of the Web, it is quite challenging (and expensive) to build a centralized architecture or to make a single emulated machine handle requests from the multiple clients to many more servers. Instead, a distributed approach can rely on the natural diversity of the connectivity offered by experimenters. However, WebTune is open source [5] and can thus be used with *any* testing network, such as a collection of cloud servers.

Researchers interested in joining WebTune donate a spare machine of their lab resources where to run WebTune's *node* code (see below). There are only three requirements to be met. First, the selected machine should be publicly reachable from the outside world on a configurable port which will be used to serve content over HTTPS. Second, the researcher must grant `ssh` access to WebTune's access server (via pubkey only). Last, the machine should run Docker [16], the OS-level virtualization we have chosen for our testing node. Docker was chosen because of it is well-known, portable, easy to deploy, and offers high performance.

WebTune's Docker image is based on a minimal Ubuntu equipped with common Linux utilities such as `tc` [17] to allow control on available bandwidth, and (extra) network latency or losses. The image also comes with a `MySQL` database where to store cloned versions of webpages, and `mitmproxy` to serve such cloned content, as we will discuss later. Finally, the image includes WebTune-specific code needed for communication with the access server and to maintain up-to-date information about the rest of the testing network. According to the experiment needs, a node's TCP stack (e.g., TCP version and parametrization) can also be customized. Given that Docker relies on the host kernel, such configurations apply to the whole machine.

Each testing node also maintains an *RTT table* reporting

Parameter	Description	Values
HTTP	Version of HTTP	HTTP/1.1, HTTP/2, HTTP/3, QUIC
TCP	TCP congestion control and settable features.	Cubic, BBR, Yeah, Hybrid slow start, etc.
Regions	Geographical locations of preferred testing nodes	Europe, US, ITA, etc.
Delay	RTT distribution between experimenter and testing nodes	Min/max RTT
Bandwidth	Bandwidth distribution between experimenter and testing nodes	Min/max bandwidth

TABLE I
WEBTUNE'S PARAMETERS DESCRIPTION.

on measured RTT between itself and the rest of the network. Further, it maintains fresh information about its available bandwidth obtained via the Speedtest CLI [18]. The RTT table and the upload bandwidth are shared with the access server and used when deciding which node to associate with a target domain.

Last but not least, testing nodes can be instrumented to collect TCP logs (e.g., via Linux socket statistics [19]) which allow to combine the client-side view, e.g., performance metrics like SpeedIndex [20], with the server side view, e.g., evolution over time of the congestion control window (see Fig. 8). This greatly improves an experimenter visibility in the set of conditions which can determine the outcome of an experiment, helping untangling the intricacy of web performance measurements we discussed in Sec. II.

B. Access Server

The main role of the access server is to manage WebTune's testing network and enable experiments. We built it atop of the Jenkins continuous integration system [21] since it offers many of the functionalities we need. First, it manages access control, e.g., `ssh` key management, to the testing network. Second, it keeps up-to-date statistics, e.g., RTT and availability, for each testing node. Third, it handles end-to-end test pipelines while supporting multiple users and concurrent timed sessions.

WebTune's access server runs in the cloud (Amazon Web Services). Nodes from the testing network are added by WebTune's system administrator via IP lockdown and security groups. `Ssh` is used to allow the access server to communicate with a node via public key and IP white-listing, which are configured at a node during registration.

In the following, we discuss more key features of WebTune's access server.

Restful API: WebTune offers restful APIs which an experimenter can use to request a detailed configuration of the servers (testing nodes) responsible of a "Webtuned" webpage. This boils down to a JSON object, as shown in Fig. 3, `{"URL":"example.com", "HTTP":"H1", "TCP":, "BBR", "RTT":"50-100ms", "Location":"US", "Duration":"30mins"}`, which indicates an experimenter aims to study the performance of *example.com* while served using HTTP/1.1 and BBR as HTTP and TCP algorithms, from servers located in the US and with RTT between 50-100ms range, between experimenter and testing nodes.

Once a request as the above is received, the *webtuning* of a webpage starts. This consists of generating a cloned version of the page, which is then distributed in the testing network to meet the experimenter requirements. Finally, the access server informs the client that the webpage has been webtuned and "how" to reach it. As we will discuss later, this is as simple as pointing the testing device to a remote Proxy Auto-Configuration (PAC) file – located at the access server – generated for this experiment.

Webpage Cloning: A "Webtuned" page is a *cloned* webpage served by multiple nodes from the testing network. There are several tools for cloning a webpage (e.g., wget, puppeteer, mitmproxy). We have evaluated those tools and found each of them to present different issues, such as lack of maintenance for WprGo, low accuracy for wget, and engineering requirements to adapt to our tool (Mahimahi), that would require significant dedication to improve the cloning result. We investigate Web Page Replay (WprGo, part of catapult [8]), wget [22], and two scrapers built on Puppeteer [3]. WprGo returns a pointer-related error even on their example (with an associated open issue) and it is not well maintained anymore. The other tools work – although wget requires careful setup[1] – but, overall, miss more content than the solution we have adopted which we will describe shortly. Mahimahi's Web replay is a potential candidate, but it requires engineering work to untangle the record/replay operations and adapt them to WebTune.

The solution we have adopted combines a real browser (Chrome) with a man-in-the-middle proxy (mitmproxy [23]). The usage of an actual browser ensures all network requests associated with a page are executed. Mitmproxy intercepts these requests – even HTTPS with the setup of a root CA at the testing client – and fetches their associated content while making a local copy for future requests. We wrote a python script for mitmproxy which checks each requested content for availability in a local database (clone). If available, the content is loaded (both header and actual content), compressed, and returned. Otherwise, the request is forwarded to the original source. The response is then intercepted, appropriately stored in a database, and forwarded back to the client. If a webpage utilizes domain sharding [24], *i.e.,* embedded web objects are distributed across multiple (sub)domain names, the (by default) independent sharded subdomains are distributed to different testing nodes. An experimenter can also request test un-sharded version of the same page, where sharded subdomains are *coalesced* – as currently done by HTTP/2 [25].

Webpage cloning can also be delegated to an experimenter. This is useful for two reasons: to reduces the load on the access server, and when needed to estimate the RTT between experimenter and the domains used by a webpage under test. This might be required when an experimenter's goal is to test a

[1]Several uncommon wget's basic flags are essential to comprehensively clone a webpage: *–span-hosts*, *–convert-links*, *–random-wait*

webpage over WebTune under the same network conditions as its original counterpart but, for instance, different TCP flavors. An experiment might run this test independently and then use WebTune's API to enforce the desired RTT conditions.

Nodes Selection: Testing nodes are selected based on the intersection of user needs, e.g., only nodes in the US, and particular nodes status such as reachability and conflicting experiment status. For example, if a node is involved in a test while using `bbr` congestion control, it cannot also be used in a test requesting `cubic`.

RTT is a key feature among the other criteria in the node selection – given a target RTT for a domain, only certain nodes can be considered as candidates for serving a domain. Under the assumption that an experimenter is also co-located with a node from the testing network, its RTT table – periodically returned to the access server – contains all the information needed for this selection. Using information from the RTT table, the access server identifies nodes in the testing network whose RTT closest to that of the domains to be tested.

Once the node selection is concluded, the cloned content of a webpage is distributed to the selected testing nodes and stored appropriately in their databases. Each node also runs mitmproxy to eventually serve cloned content. Finally, a PAC file is generated to reflect the node selection, *i.e.,* inform the experimenter which mitmproxy to use for each domain involved in the webpage under test. This allows transparent experimenting without any changes in both cloned content, e.g., URL rewriting, and testing code.

C. Experimenter

Last but not least, an experimenter is needed to perform actual web experiments with WebTune. There is no restriction on the experimenter *type*, *i.e.,* desktop or mobile as well as a commercial browser or some research Python code. The only requirements are: 1) use WebTune APIs to prepare an experiment, 2) speak HTTP(S), 3) setup the device with the PAC file produced by the server for this test. In presence of HTTPS, WebTune's root CA should be installed on the testing device. Alternatively, the client can be setup to ignore certificate errors, e.g., using flag `-ignore-urlfetcher-cert-requests` in Chrome.

IV. Evaluation

In the following paragraphs, we present results from an evaluation of WebTune. We start by analyzing the *accuracy* of webpage cloning, and the *complexity* of the networking conditions that WebTune should aim at reproducing. Next, we show how WebTune can be beneficial for Web performance research.

A. Cloning Accuracy

We integrate *cloning* and *testing* in a single tool. Given an input webpage, the tool loads it via Lighthouse while collecting `devtools` [26] data such as size of the objects retrieved during a load. Meanwhile, it triggers the cloning procedure to generate the Webtuned version of the webpage under test, for now using a single node hosted at our premises. Next, Lighthouse is instrumented to load the newly created Webtuned webpage. We run this experiment for Alexa top-100 pages using Chrome.

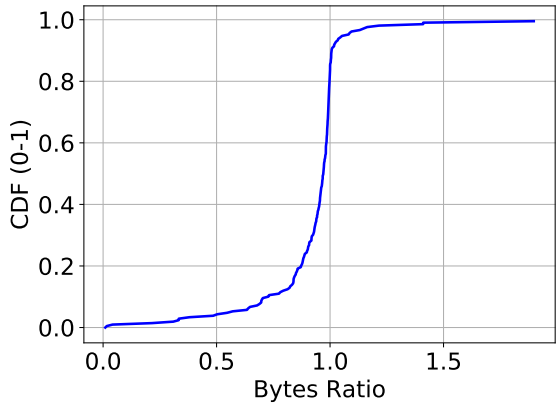


Fig. 4. CDF of bytes ratio (cloned/original). For 55% of pages, the cloned and original versions are very similar in size, with a ratio in [0.9, 1.1]. The differences presented by the top 5% pages with larger cloned pages than the respective original and the other 20% with a ratio in [0.8, 0.9] are predominantly due to dynamic content.

Figure 4 shows the CDF of *bytes ratio* across Alexa top 100 webpages, i.e., the ratio between the bytes received during the load of the cloned and the original webpages, respectively. The figure shows that for 55% of the pages, the cloned and original versions are very similar in size, with a size ratio between 0.9 and 1.1. About 5% of the cloned pages are bigger than the original. This is due to differences between the original and cloned pages due to dynamic content and potentially slightly difference (gzip) compression. This is not unexpected given the hazards of web measurements; even reloading the same original webpage multiple times would produce a page with slightly different sizes each time. Next, 20% of the pages have a ratio between 0.8 and 0.9; while both versions are still quite similar, the smaller cloned versions are due to the presence of dynamic content and other errors, as we discuss in the next paragraphs.

Next, we focus on the *appearance* of a cloned webpage or how similar it looks to the original one. We compute the Structural SIMilarity (SSIM) index – a method for measuring the similarity between two images – between screenshots obtained post the `onLoad` event, *i.e.,* when the page is considered fully loaded by the browser.[2] Figure 5 shows the CDF of the SSIM index for Alexa top 100 pages: 0 means completely different pages and 1 equal pages. The figure shows a SSIM higher than 0.95 for about 60% of the webpages. These are webpages where cloning works perfectly and only very minor differences exist. Next, 30% of the pages have an SSIM between 0.6 and 0.95; in these cases, we can

[2]The webpage similarity can be computed considering either their structure (generally static) or aesthetics (can be more dynamic). Our cloning copies the structure fairly accurately in general, but cloning the "look" is much more challenging. Therefore we chose SSIM as a similarity metric.

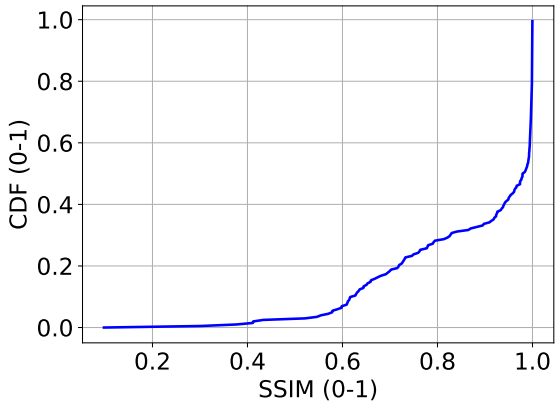
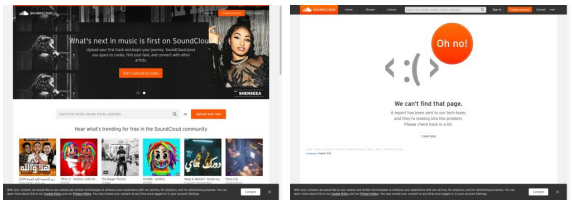
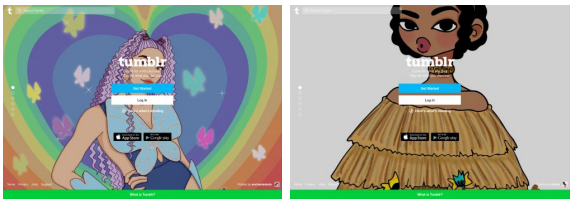


Fig. 5. CDF of SSIM index between cloned and real webpages. About 60% of pages have a SSIM higher than 0.95.



(a) A-original (b) A-clone



(c) B-original (d) B-clone

Fig. 6. Examples of challenging pages. SSIM is 0.31 for A, due to a 404 result that alters the page appearance, and 0.50 for B due to the change of background image.

observe slightly higher differences caused by things like image compression or position of warnings (e.g., cookies accepts or privacy consent), however the overall appearance of the webpages was intact.

The bottom 10% of webpages show significant difference (SSIM<0.6), highlighting a need to improve our cloning tool. Fig. 6 shows examples of two webpages (original on the left and cloned on the right). For the case A, for which we measured a SSIM of 0.31, some contents are missing causing a 404 which dramatically modifies the page appearance, yield such low SSIM. The case B (SSIM 0.50) is different as both the original and the cloned webpages are correctly displayed, but the original server serves different content (the background image in this case) for each access. In future work we plan to improve WebTune to address these two problematic cases, as they are the main responsible for low SSIMs in our measurement.

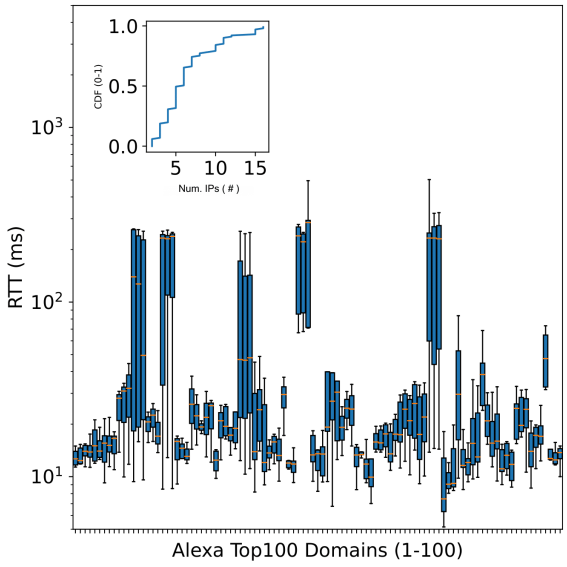


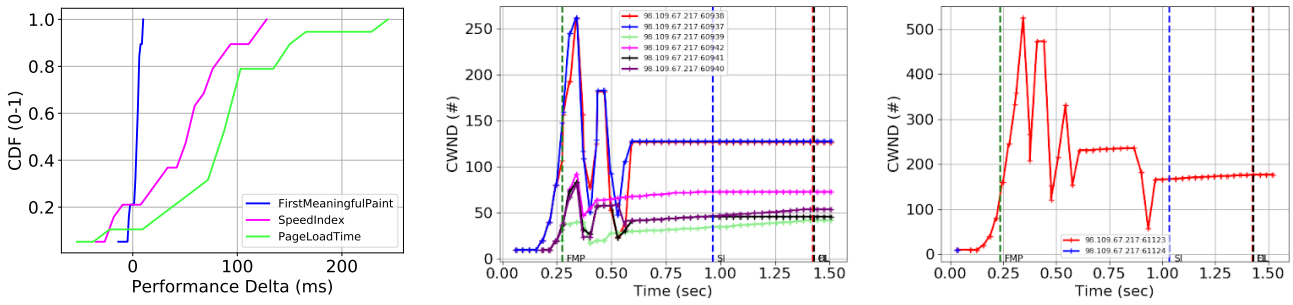
Fig. 7. Boxplots of latency distribution per Alexa top 100 ordered by rank. The inserted plot shows the CDF of the number of unique IPs per webpage load. These two results suggest the number testing nodes necessary to reproduce a pageload with varied delays.

Node Distribution with RTT: Figure 7 reports both on the RTT distribution (one boxplot per webpage) and number of IPs involved in these experiments. Intuitively, both RTT and number of IPs are good indicators of the footprint that WebTune should target. For example, the figure shows that 5 WebTune nodes can cover 50% of Alexa top 100 webpages, and 15 WebTune nodes are required to serve all the pages we tested. Further, the measured RTTs vary between few milliseconds up to even half a second, in few cases. This result reinforces our motivation of WebTune 's crowdsourced design for the testing network, which allows to reproduce such complex mix of network conditions from which experimenter would be able to chose. Having more participants in the testing network should result on a wider range of RTTs and more accurate node selection.

B. Applications

In this subsection, we demonstrate WebTune's functioning using two example applications: upgrading to HTTP/2, and adopting CDN. These are illustrative examples with the goal to showcase WebTune's versatility as a tool for web performance research; we acknowledge that the above research questions are complex and their thorough exploration is beyond the aim of this work.

For these experiments, we use a testing network formed by 5 WebTune proxies (both in-lab machines and AWS instances), which could cover >50% of Alexa top 100 pages in Sec. IV-A, grouping domains to each proxy based on RTT information (unless otherwise noted) in the attempt to reproduce similar network conditions as the one measured towards the original domains used by a webpage. The testing client is a legacy



(a) CDF of the performance delta when webpages are fully served via HTTP/2.

(b) Evolution of `cwnd` over time when using HTTP/1.1 ; Cubic.

(c) Evolution of `cwnd` over time when using HTTP/2 ; Cubic.

Fig. 8. Using WebTune to investigate potential benefits of upgrading HTTP/1.1 to HTTP/2.

Chrome browser instrumented by Lighthouse [2] running on a desktop machine (macOS) connected over a 100 Mbps fiber connection (both in downlink and uplink).

HTTP/2 Upgrade: Several studies have investigated the performance benefits (or penalties) of HTTP evolution, especially when HTTP/2 was standardized back in 2014/2015 [27]–[30]. One common approach consists in instrumenting Chrome to turn on/off HTTP/2, and compare performance using metrics like `SpeedIndex` or `PageLoadTime`. This is because, without browser modifications as in Agarwal et al. [11], the browser does not allow to disable HTTP/2 just for one domain. Also, if the remote server responsible for a domain does not support HTTP/2, the client can not force it to switch.

WebTune allows fine-grained control on which specific HTTP version is supported at the server (proxy) responsible for a domain. We have identified 30 domains from Alexa top 100 with partial HTTP/2 support, and studied the performance impact of a 100% HTTP/2 support using WebTune. Figure 8(a) shows the performance delta using the three most popular metrics, `FirstMeaningfulPaint`, `SpeedIndex`, and `PageLoadTime`, as reported by Lighthouse. The delta of each metric is computed between testing the original page in WebTune, same amount of HTTP/1.1 and HTTP/2 as in the original page, and a page with 100% HTTP/2 support. It follows that values smaller/larger than zero indicate a performance penalty/improvement.

Figure 8(a) shows that, regardless of the metric, a full switch to HTTP/2 offers performance benefits to 80-90% of the webpages. The impact is more prominent on "later" metrics, *i.e.,* metrics which are triggered later in time during the page load, such as `SpeedIndex` and `PageLoadTime`, with median benefits of up to 90ms. Note that these results refer to a single network conditions (fast fiber), and might not extend to other conditions like "jittery" mobile networks [11], [31]. Again, our goal is to showcase WebTune flexibility rather than a thorough exploration of this specific research question.

Following up with our goal, Fig. 8(b) and 8(c) demonstrate the level of visibility achieved by WebTune. Each figure plots the evolution over time of the `cwnd` (server-side) when serving a webpage hosted on a single domain using HTTP/1.1 and HTTP/2, respectively. Each line in each plot refers to a different TCP connection (identified by src ip and port of the testing client); the plot further shows as vertical lines each performance metric collected by lighthouse, e.g., `SpeedIndex` labelled as SI. We chose a webpage hosted on a single domain for increased visibility.

The figures visualize one key novelty of HTTP/2: the usage of a single connection, versus 6 connections used by HTTP/1.1 (at least in Chrome). Note that the second short lived connection shown in Figure 8(c) is just a trick Chrome uses to speedup connection setup. The plots show typical TCP `cubic` (default in the Linux kernel) behaviors: rapidly growing the `cwnd` until a loss is detected, followed by a slower increase in the attempt to avoid congestion. Note that the `cwnd` is (mostly) flat towards the end of each plot because the server has no data to send, e.g., because the browser is "stuck" parsing and rendering with no new content requested. These plots show a scenario where HTTP/1.1 manages to be overall competitive: slower `FirstMeaningfulPaint` but faster `SpeedIndex` – which is the metric better representing user experience – and equivalent `PageLoadTime`. Although not shown due to space limitations, the explanation of the slower `FirstMeaningfulPaint` with HTTP/1.1 derives from the time spent in setting up the 6 TCP connections along with TLS (a fresh visit was considered, *i.e.,* no session resumption was used in this example). With respect to `SpeedIndex`, Fig. 8(c) shows that an additional loss is responsible for the slower `SpeedIndex` in case of HTTP/2. This is a probabilistic event rather than a protocol difference, thus suggesting a potential outlier. This showcases the importance of full-stack (both client and server) visibility offered by WebTune to fully understand Web performance experiments.

CDN Adoption: Next, we use WebTune to quantify the benefits of adding a CDN to *CDN-less* webpages, *i.e.,* webpages which are still self-hosted. We have identified 15 webpages from Alexa top100 which are *CDN-less* by analyzing the hosts of the requests, and we tested them for potential performance improvements when adding a CDN. To do so, we test the

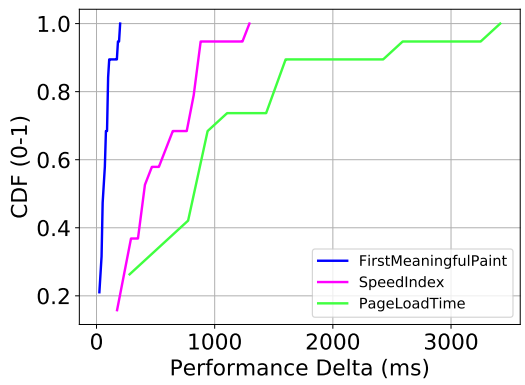


Fig. 9. Using WebTune to quantify the performance benefit (in delta) introducing CDN.

original webpage in WebTune by mapping their domains to our available proxies with most similar RTTs. Next, we run a second test where we make sure a maximum RTT of 15 ms is used, similar to what achieved by most CDNs today [32]. To achieve this, we deployed multiple proxies in our LAN and used Linux `tc` to add some artificial delay.

Figure 9 shows the performance delta from introducing a CDN in these webpages. In this case all webpages benefit from increased performance by adopting a CDN. Further, performance improvements are significant, e.g., a median of half second when considering `SpeedIndex`.

V. Related Work

A number of efforts from both industry and academia aimed at improving web performance has yielded several performance measurement tools. Puppeteer [3] and Lighthouse [2] load webpages and report on their performance across a large set of metrics in Chromium-based browsers. WebPageTest [1] further provides cloud-based tests which allows webpage performance tests from a variety of different devices and network locations. Tools such as Web Page Replay [8] or Mahimahi [4] allow to record and replay HTTP traffic over a local testbed using network emulation to reproduce a single (Web Page Replay) or multiple (Mahimahi) domains. WebTune is similar in spirit to Mahimahi, but that brings the fundamental innovation to achieve a similar outcome over a real network. This is key given that recent work has demonstrated that network emulation, even via accurate traces, can be far from reality [33].

Although different, these tools share a common limitation: lack of server-side visibility. This limits an experimenter's understanding of web performance tests to information gleaned from the client-side. Unless we are a content provider, the only way to gather server-side data is to independently create and host web content, often synthetic, to measure. Having only half of the picture can be catastrophic to the understanding of performance and why a webpage loaded the way it did.

VI. Conclusion and Future Work

The wide range of webpage designs, network settings, client and server configurations, complicates Web measurements. Further, the constantly changing web content renders these measurements often inconsistent. This heterogeneity and dynamism are a challenge for experimenters as they complicate evaluations and cast doubts in their conclusions. Our proposal to this problem is *WebTune*, a distributed platform for Web performance measurements that (i) seamlessly integrates with existing web measurement tools, and (ii) allows fine grained control on real networks and servers. In practice, webpages are "Webtuned" or *cloned* and *distributed* to a testing network which is instrumented as per an experimenter need, e.g., TCP stack and network delay.

We have developed and open sourced WebTune [5]. We presented evaluation results on the accuracy of webpages clonning and estimates of the scale of the testing network needed to achieve realistic testing conditions. We find that WebTune achieves high accuracy, e.g., accurately cloning 80-90% of Alexa's top 100 webpages, and can handle most webpages with a testing network composed of 5-10 nodes. Finally, we demonstrated WebTune capabilities while investigating two illustrative research questions with a 5-nodes deployment.

As part of future work we plan to improve webpage cloning to eliminate some of the issues we have identified and make it more consistent. In particular, resolving 404 error and dynamic contents in the same page would be important to increase accuracy. We also plan to work on avoiding cloning of illicit contents, as it can put researchers/resource host under risk. More importantly, we plan to build a community around WebTune. We will bootstrap the testing network using machines at our premises and colleagues' who have already shown interest in WebTune.

Acknowledgments

We thank our shepherd Roland van Rijswijk-Deij and the TMA anonymous reviewers for their insightful suggestions and feedback. We also would like to thank Andrius Aucinas and Neil Agarwal for their help with the background and motivation of this work.

References

[1] P. Meenan, "Webpagetest: Test a website's performance," accessed on 04.13.2020. [Online]. Available: https://webpagetest.org
[2] Google, "Lighthouse," accessed on 04.13.2020. [Online]. Available: https://developers.google.com/web/tools/lighthouse
[3] Google., "Puppeteer," accessed on 04.12.2020. [Online]. Available: https://developers.google.com/web/tools/puppeteer
[4] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: Accurate record-and-replay for HTTP," in *Proc. USENIX ATC*, 2015.
[5] M. Varvello and J. B. Jun, "Webtune codebase." [Online]. Available: https://github.com/svarvel/webtune
[6] Google, "Custom site performance reports with the crux dashboard." [Online]. Available: https://developers.google.com/web/updates/2018/08/chrome-ux-report-dashboard
[7] A. Mishra, X. Sun, A. Jain, S. Pande, R. Joshi, and B. Leong, "The great internet tcp congestion control census," *Proc. ACM SIGMETRICS*, vol. 3, no. 3, pp. 1–24, 2019.

[8] Catapult-project, "Web page replay," accessed on 04.12.2020. [Online]. Available: https://github.com/catapult-project

[9] A. S. Asrese, E. A. Walelgne, V. Bajpai, A. Lutu, Ö. Alay, and J. Ott, "Measuring Web Quality of Experience in Cellular Networks," in *Proc. of PAM*, 2019.

[10] Y. Cao, A. Jain, K. Sharma, A. Balasubramanian, and A. Gandhi, "When to use and when not to use BBR: An empirical analysis and evaluation study," in *Proc. of IMC*, 2019.

[11] N. Agarwal, M. Varvello, A. Aucinas, F. E. Bustamante, and R. Netravali, "Mind the delay: the adverse effects of delay-based tcp on http," in *Proc. ACM CoNEXT*, 2020.

[12] U. Naseer and T. Benson, "InspectorGadget: inferring network protocol configuration for web services." in *Proc. ICDCS*, 2018.

[13] Amazon, "Alexa top 100," https://www.alexa.com/topsites.

[14] P. Meenan, "Better http/2 prioritization for a faster web," accessed on 05.19.2020. [Online]. Available: https://blog.cloudflare.com/better-http-2-prioritization-for-a-faster-web/

[15] A. Davies, "Tracking http/2 prioritization issues0." [Online]. Available: https://github.com/andydavies/http2-prioritization-issues

[16] https://www.docker.com/.

[17] https://man7.org/linux/man-pages/man8/tc.8.html.

[18] https://github.com/sivel/speedtest-cli.

[19] https://man7.org/linux/man-pages/man8/ss.8.html.

[20] Google, "Speed index," accessed on 03.22.2020. [Online]. Available: https://developers.google.com/web/tools/lighthouse/audits/speed-index

[21] https://www.jenkins.io/.

[22] I. Free Software Foundation, "Gnu wget," accessed on 04.30.2020. [Online]. Available: https://www.gnu.org/software/wget/

[23] M. Project, "Mitm proxy," accessed on 03.25.2021. [Online]. Available: https://mitmproxy.org

[24] U. Goel, M. Steiner, M. P. Wittie, S. Ludin, and M. Flack, "Domain-sharding for faster http/2 in lossy cellular networks," *arXiv preprint arXiv:1707.05836*, 2017.

[25] D. Stenberg, "Http/2 connection coalescing." [Online]. Available: hhttps://daniel.haxx.se/blog/2016/08/18/http2-connection-coalescing/

[26] Google, "Chrome devtools," accessed on 07.04.2020. [Online]. Available: https://developer.chrome.com/docs/devtools/

[27] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How speedy is SPDY?" *Proc. of USENIX NSDI*, 2014.

[28] U. Goel, M. Steiner, M. P. Wittie, S. Ludin, and M. Flack, "Domain-Sharding for Faster HTTP/2 in Lossy Cellular Networks," 2017. [Online]. Available: http://arxiv.org/abs/1707.05836

[29] H. De Saxce, I. Oprescu, and Y. Chen, "Is HTTP/2 really faster than HTTP/1.1?" *Proc. IEEE INFOCOM*, vol. 2015-Augus, pp. 293–299, 2015.

[30] "Modeling HTTP/2 speed from HTTP/1 traces," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9631, pp. 233–247, 2016.

[31] U. Naseer and T. Benson, "Configtron: Tackling Network Diversity with Heterogeneous Configurations," *arXiv preprint*, 2019. [Online]. Available: https://www.usenix.org/system/files/conference/hotcloud17/hotcloud17-paper-naseer.pdf

[32] M. Williams, "Website latency with and without a content delivery network." [Online]. Available: https://www.keycdn.com/blog/website-latency

[33] F. Y. Yan, H. Ayers, C. Zhu, S. Fouladi, J. Hong, K. Zhang, P. Levis, and K. Winstein, "Learning in situ: a randomized experiment in video streaming," in *Proc. of USENIX NSDI*, 2020.