Web-LEGO: Trading Content Strictness for Faster Webpages

Pengfei Wang[†]*, Matteo Varvello[‡], Chunhe Ni[§], Ruiyun Yu[¶], Aleksandar Kuzmanovic[‖]
[†]School of Computer Science and Technology, Dalian University of Technology, China
[‡]Nokia Bell Labs, USA
[§]Amazon, USA
[¶]Software College, Northeastern University, China
[‖]Department of Computer Science, Northwestern University, USA
wangpf@dlut.edu.cn, matteo.varvello@nokia.com,
nichunhe@amazon.com, yury@mail.neu.edu.cn, akuzma@northwestern.edu

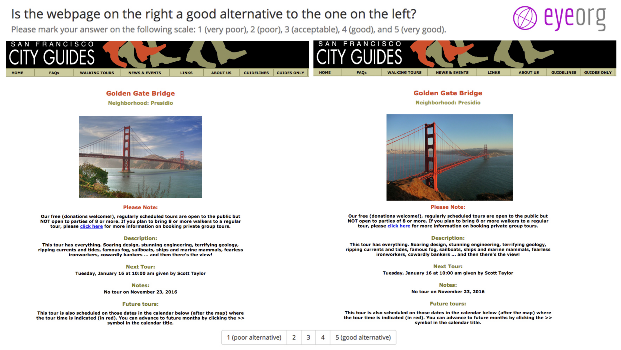
***Abstract*—The current Internet content delivery model assumes strict mapping between a resource and its descriptor, e.g., a JPEG file and its URL. Content Distribution Networks (CDNs) extend it by replicating the same resources across multiple locations, and introducing multiple descriptors. The goal of this work is to build Web-LEGO, an opt-in service, to speedup webpages at client side. Our rationale is to replace the slow original content with fast similar or equal content. Further, we perform a reality check of this idea both in term of the prevalence of CDN-less websites, availability of similar content, and user perception of similar webpages via millions of scale automated tests and thousands of real users. Then, we devise Web-LEGO, and address natural concerns on content inconsistency and copyright infringements. The final evaluation shows that Web-LEGO brings significant improvements both in term of reduced Page Load Time (PLT) and user-perceived PLT. Specifically, CDN-less websites provide more room for speedup than CDN-hosted ones, *i.e.,* 7x more in the median case. Besides, Web-LEGO achieves high visual accuracy (94.2%) and high scores from a paid survey: 92% of the feedback collected from 1,000 people confirm Web-LEGO's accuracy as well as positive interest in the service.**

***Index Terms*—Web acceleration, quality of experience, crowdsourcing, network performance**

I. Introduction

High Quality of Experience (QoE) on the Web is essential for both content providers and end-users. QoE directly affects content providers' business revenues and clients' willingness to wait for and revisit a webpage [1], [2]. Amazon has reported that every 100 ms increase in page load time costs them 1% in sales [3]. As a result, significant efforts have recently been made from both industry (e.g., QUIC, SPDY, and HTTP/2) and academia [1], [4], [5], [6], [7] to design tools and novel protocols which can reduce page load times.

Despite these efforts from both industry and academia, Content Distribution Networks (CDNs) are still the most valuable asset to speed up web content delivery. CDNs disrupt the Internet content delivery model from *device-centric* to *content-centric*, and move popular content close to the users by leveraging hundreds of thousands of servers distributed worldwide [8], [9]. To further improve user's QoE, CDNs redirect clients to different servers over a very short time scale by leveraging extensive network and server measurements.


Is the webpage on the right a good alternative to the one on the left?
Please mark your answer on the following scale: 1 (very poor), 2 (poor), 3 (acceptable), 4 (good), and 5 (very good).

Golden Gate Bridge
Neighborhood: Presidio

Please Note:
Our free (donations welcome!), regularly scheduled tours are open to the public but NOT open to parties of 8 or more. If you plan to bring 8 or more walkers to a regular tour, please click here for more information on booking private group tours.

Description:
This tour has everything. Soaring design, stunning engineering, terrifying geology, ripping currents and tides, famous fog, sailboats, ships and marine mammals, fearless ironworkers, cowardly bankers ... and then there's the view!

Next Tour:
Tuesday, January 16 at 10:00 am given by Scott Taylor

Notes:
No tour on November 23, 2016

Future tours:
This tour is also scheduled on those dates in the calendar below (after the map) where the tour time is indicated (in red). You can advance to future months by clicking the >> symbol in the calendar title.

Golden Gate Bridge
Neighborhood: Presidio

Please Note:
Our free (donations welcome!), regularly scheduled tours are open to the public but NOT open to parties of 8 or more. If you plan to bring 8 or more walkers to a regular tour, please click here for more information on booking private group tours.

Description:
This tour has everything. Soaring design, stunning engineering, terrifying geology, ripping currents and tides, famous fog, sailboats, ships and marine mammals, fearless ironworkers, cowardly bankers ... and then there's the view!

Next Tour:
Tuesday, January 16 at 10:00 am given by Scott Taylor

Notes:
No tour on November 23, 2016

Future tours:
This tour is also scheduled on those dates in the calendar below (after the map) where the tour time is indicated (in red). You can advance to future months by clicking the >> symbol in the calendar title.

Fig. 1. User opinions on trading content strictness for faster webpages.

While CDNs undoubtedly help content providers, only popular content providers can afford their services. Despite the availability of free (basic) CDN services from providers like CloudFlare [10] , 36% of Alexa's top 1,000 websites (and 77% of the top one million websites) do not use a CDN. Yet, the same constraints regarding user experience and revenue loss for associated content providers hold for such websites. In a way, this creates a negative cycle where websites fail to attract clients due to their poor performance, yet improving performance depends on revenues from the clients.

The numbers above suggest that "server side" technologies to improve page load time–such as CDN adoption, update to HTTP/2, or other–are unlikely to be deployed *where* needed the most. Motivated by this observation, the goal of this work is to build a client side mechanism (*Web-LEGO*) to speedup webpages in the absence of server side cooperation. Our key idea is to trade content strictness for performance. By enabling users to download *similar*, yet not necessarily the same content they requested (e.g., a similar but not the same JPEG file as shown in Figure 1), we generate the opportunity to locate faster content thus implicitly speeding up a webpage. Nevertheless, Web-LEGO can be coupled with all current popular web accelerating methods (e.g., optimizing dependency graph and caching [1], [4], [5], [6], [7], [11], [12], [13]).

In this paper, we went one step further and relaxed the latter constraint proposing a novel content delivery model where

TABLE I
FRACTION OF WEBSITES USING CDN SERVICES AS A FUNCTION OF WEBSITE POPULARITY (ALEXA RANKING).

top-100	top-1K	top-10K	top-100K	top-1M
70.0%	64.3%	49.3%	35.4%	22.9%

similar content can be downloaded from arbitrary servers. We realized this abstract idea into Web-LEGO, a concrete solution which requires no changes to the existing Internet infrastructure. Web-LEGO identifies similar content on the fly and offers the best available *copy* with minimal distortion of content exactness. Toward a comprehensive solution for speeding up webpages leveraging widely existed similar content, we make the following contributions:

- With millions of automated tests and thousands of real user feedback, we present a large scale reality check to understand the feasibility of Web-LEGO in term of the prevalence of CDN-less websites, availability of similar content, and user perception of similar webpages.
- We design and implement the idea as Web-LEGO, an *opt-in* speedup service where users can trade faster webpages for slightly different content at client side. As far as we know, this is the first webpage accelerating service leveraging the widely existing similar (and equal) content widely deployed on the Internet.
- Our final evaluation with large scale tests shows that Web-LEGO brings significant improvements in term of reduced PLT (up to 5.5s) and uPLT (up to 5s).
- We also discuss the potential problems which can meet when Web-LEGO is deployed in the wild in the future, e.g., advanced alternative content selection, security, and economic sustainability problem, etc.

II. REALITY CHECK

Web-LEGO is motivated by three assumptions that we set out to verify: 1) a significant fraction of websites do not use a CDN, 2) the Internet abounds of *similar* content, and 3) users are willing to trade content strictness for faster websites.

Prevalence of CDN-less websites Web-LEGO aims to improve the performance of *not-so-popular* websites that currently do not use a CDN (*CDN-less*), either to avoid extra costs or due to a lack of technical expertize. This raises the question: *how many websites are CDN-less today?* Table I shows the fraction of websites that are hosted by at least one CDN from the Alexa's top 100 up to Alexa's top 1 million websites. This data was obtained by querying `whatsmycdn.com`, a public service which reveals the underlying CDNs for an arbitrary URL [14]. The table shows that while the majority of popular websites are hosted by CDNs, e.g., 70-64.3% if we focus on the top 100 and 1,000 websites, only 22.9% of the Alexa top 1 million websites use at least one CDN.

Content similarity Our work relies on the underlying assumption that the Internet abounds of *similar* content, e.g., images of the same bridge from a different angle (see Figure 1). We investigate this assumption by extracting all image URLs embedded in 100 CDN-less webpages randomly selected

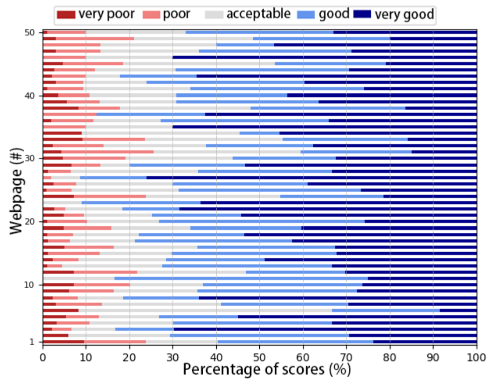


Fig. 2. Visualization of user feedback on Web-LEGO alternative websites ; 50 websites, 1,000 paid participants, 6,000 responses.

from Alexa and searching for similar images using Google Images [15]—whose reverse image search allows to locate similar images across billions entries. We find that Google Images returns at least 10 similar images for 92% of cases, and at least two similar images for 97% of cases. The procedure above was repeated also for 100 CDN-hosted websites and no significant difference was observed. This result suggests that similar content does exist, as we further verify in Section IV-B.

User Feedback Web-LEGO is designed as an *opt-in* service, and its users are informed to be trading faster webpages for slightly different content. Estimating how many users would accept this compromise is hard, as it requires launching the service and attracting a substantial user-base. We thus resort to crowdsourcing to shed some light on this question.

We use our previous work – Eyeorg [16] , which is a system for crowdsourcing Web QoE measurements, to collect user feedback on whether they would consider Web-LEGO-generated webpages as alternatives to an original page, granted that they can be served faster. We generated a new experiment type where two snapshots (instead of videos) of fully loaded webpages are presented side by side (see Figure 1). Prior to the test, participants are informed that the page on the right was automatically generated by our *algorithm* with the goal to be similar to the original one but faster. Participants are also informed that the text was not altered, and that only images have been potentially altered. Participants are finally asked to rank the proposed alternative websites on a scale 1 (very poor), 2 (poor), 3 (acceptable), 4 (good), and 5 (very good).

Using the methodology above, we run an Eyeorg campaign where we ask each paid participants to *score* 6 side-by-side website comparisons. We recruited 1,000 users on Appen (cost: \$120), and collected 6,000 evaluations of 50 webpage comparisons ($\sim$120 scores per webpage). Paid participants are selected as “historically trustworthy,” a Appen's feature which guarantees trustworthy participants at the cost of a longer recruitment time. We further discard participants with potential low quality, *i.e.,* participants who skipped our detailed instructions, spent less than one second on a task, provided always the same score across six website comparisons. This filtering accounts for about 7% of the paid participants.

Figure 2 visualizes the scores collected per webpage. The majority of the scores (92.5%) are positive, *i.e.,* acceptable, good, or very good. On average, a website only had ∼12% of the scores which were negative. We visually inspected websites where more than average low scores were observed, e.g., 25% for website-31. A recurrent pattern was observed: slightly different image sizes cause small variations to the page layout. Participants could spot these minor variations because of the two pages being displayed side-by-side which is necessary for the study but also an adversary condition for Web-LEGO which is not to be found in regular browsing. This study indicates that Web-LEGO is doing a good job in selecting alternative content (see Section III). Further, it shows encouraging results with respect to the potential adoption of the novel content delivery model here proposed.

III. Design and Implementation

A. Design

How does Web-LEGO work? Figure 3 shows Web-LEGO's three main components: client, similarity storage server (3S), and reverse file search server (RFSS). The Web-LEGO's client has the goal to intercept regular user requests and replace them, if possible, with equal or similar content it has obtained from alternative resources. Such alternative resources are provided by 3S which, very much like DNS, responds to Web-LEGO's client queries for extra content resources. Finally, the RFSS is responsible to collect similarity information, *i.e.,* identify acceptable alternative content resources.

Web-LEGO's client intercepts web requests and 1) forwards them to its target server, 2) forward requests for web objects supported by Web-LEGO (images, JS, and CSS) to 3S, seeking for alternative resources. Upon the reception of a content request (URL), 3S looks for such URL in its database. In case of a hit, N alternative resources for the requested content are provided. In case of a miss, a negative response is returned to the client while the original request is forwarded to the RFSS. The RFSS performs a reverse file search for this content, *i.e.,* it finds alternative same or similar files, and returns alternative URLs to 3S so that future requests for the same content will not result in another miss. During the reverse file search progress, RFSS will consider the content providers' preferences of the original request and similar alternatives based on the similarity control headers to avoid the possible content inconsistency and copyright problems. Reverse file search is an active research area [15], [17], [18], [19], [20], and many operational systems exist in this domain [15], [18], [20]. Among them, we utilize Google Images and TinEye; we detail our use of these services below.

Upon the reception of N alternative resources for some content, the Web-LEGO client attempts contacting each of them. Finally, whichever of the $(N+1)$ requests—the original requests plus these N—completes first, its associated object is returned to the application. Note that the N extra requests for similar content pay an extra latency compared to the original request due to the communication with 3S; however, Web-LEGO aims at finding alternative content hosted on better-

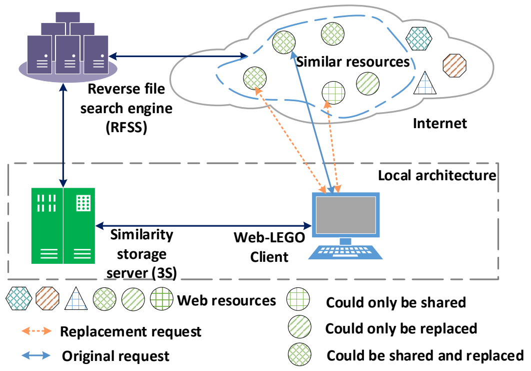


Fig. 3. Web-LEGO architecture.

provisioned and/or closer servers. Further, this latency can be minimized by caching and/or prefetching popular resources for alternative content.

How many alternative resources does Web-LEGO query? We here discuss N, or the number of alternative resources that Web-LEGO queries for each original request.A similar problem has been analyzed in depth in the context of reducing latency through redundant requests [21], [22], [23]. While Web-LEGO effectively extends this approach by allowing *similar* content retrievals, the same cost vs. benefit trade-off holds for Web-LEGO and redundant requests. In [21], the authors develop a simple, yet insightful, model to quantify the break-even benefit for a client choosing to send extra requests, measured in terms of the amount of improved performance (milliseconds) to the amount of added traffic (KBytes). As an example, the model implies that for a broadband wireline service, as long as Web-LEGO generates the benefit of at least 0.029 ms/KB, it pays off to send extra requests. For a mobile broadband plan, however, this benefit threshold is necessarily higher (0.74 ms/KB) because the cost of such service is higher. We experimentally evaluate this issue in Section IV by varying N in the range 1—3. While different networks may require different level of request replication, we find that sending up to 3 alternative requests is beneficial on broadband wireline and mobile networks alike.

Which web objects does Web-LEGO support? Modern webpages contain hundreds of different objects such as HTML, JavaScript (JS), CSS, images, XHR, etc. Currently, Web-LEGO only supports (*i.e.,* attempt to replace) images, JS, and CSS files. The remainder object types should either never be replaced (e.g., HTML) or require extra care (e.g., media objects). Substituting HTML content is not allowed since it can trigger the download of extra/unwanted content. In principle, Web-LEGO can handle media objects since they also share high similarity, e.g., two videos of the same event from a slightly different angle, and reverse video search services are indeed widely available [15], [18], [20]. However, media objects further complicate the understanding of user QoE. We thus opted to focus on media-free webpages, but we plan to remove this limitation in our future work.

Web-LEGO handles JS and CSS files differently than images. For images, Web-LEGO opportunistically attempts to replace them with the same or a *similar* content hosted elsewhere. For JS and CSS files, Web-LEGO only looks for *identical copies* hosted at alternative servers. In fact, while two similar images can convey the same semantic meaning (see Figure 1), even few different lines in JS/CSS can dramatically change the appearance and behavior of a website.

Why is Web-LEGO designed as an *opt-in* service? Web-LEGO speeds up webpages by giving up the content strictness, therefore, Web-LEGO service works for a web object only when both users and content providers agree to use it. For users, they can just disable the service on the client-side. For content providers, they can explicitly adding similarity control headers to HTTP(S) response, and RFSS will eliminate the associate content for sharing or replacing (See Section III-B3). In this way, content providers could retain fine-grained control of their web resources, and the potential content inconsistency and copyright issues could also be solved at the same time. This is somewhat similar with the web caching technique [24] where content providers are able to "opt-out" by setting cache control headers.

B. Implementation

1) Web-LEGO Client: There are three potential candidates for a (realistic) Web-LEGO client implementation, each with its own pros and cons. First, we could modify Chromium [25] adding support of Web-LEGO's client side logic. Clearly, this solution is realistic but at the expense of a significant engineering challenge. Nevertheless, because of the rapid evolution of the Chromium source base our implementation could rapidly become obsolete. We thus discarded this solution with the hope that the larger Chromium community can consider Web-LEGO's adoption in the future.

The next approach consists in implementing Web-LEGO's logic as a browser extension. This approach is also realistic, although some potential performance overhead due to the introduction of the extension. After exploring the Chrome extension capabilities—this is without loss of generality since all browsers follow a similar architecture for their extensions—we have also discarded this solution. In fact, via an extension we can intercept, block, or modify in-flight requests using `webRequest` API. For security reasonsthe same cannot be done for received responses. This would limit our design to $N = 1$, *i.e.,* "guessing" only one potentially faster alternative.

Finally, we discuss the solution we opted for: a web *proxy*. This allows to build an application agnostic (e.g., browsers and mobile apps) solution while minimizing the engineering effort. This solution also allows realistic performance testing since any modern browser is supported. The main drawback of this solution is the performance penalty caused by the proxy, which in turn implies that our evaluation represents a lower bound of Web-LEGO performance. To minimize such penalty, we have resorted to a careful implementation which we detail next. Note that HTTPS is not an issue since, in a lab setting, TLS interception can be adopted.

Concurrency The high level goal of Web-LEGO is to speedup web applications like browsers and mobile apps. It follows that the extra operations required, *i.e.,* contact 3S and lookup extra resources, should have minimal impact on the regular application execution. Our goal is thus to implement a multi-threaded proxy where the extra operations required are independent from the traffic relay operation. Beyond system-level concurrency, it is also essential to avoid the degradation of user-perceived performance due to the potential network congestion induced by added traffic. This issue was analyzed in depth in [22], showing that request replication is beneficial even under heavy load. In Section IV-D, we evaluate Web-LEGO in constrained bandwidth environments.

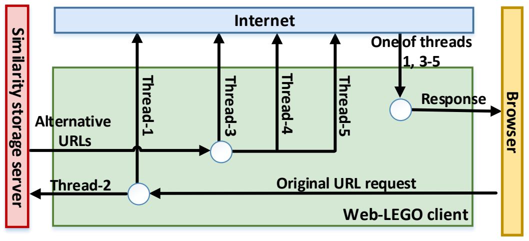


Fig. 4. The concurrent workflow of Web-LEGO's client.

The Web-LEGO's client uses a dedicated thread (thread-1 in Figure 4) to relay HTTP(S) traffic generated and received by an application. Such content requests are also shared with thread-2 which identifies if the content requested is suitable for replacing, *i.e.,* it refers to an image, JS, or CSS. We describe how we realized such content identification in the upcoming subsection. If a request is a candidate for replacing, thread-2 contacts the 3S requesting alternative resources. Assuming at least one source is returned, the Web-LEGO's client starts a thread pool. For example, if three alternative resources are returned, the pool consists of thread-3, thread-4, and thread-5 (as shown in the figure), where each thread is responsible to fetch the desired resources. As discussed above, we limit the size of the thread pool to 1, 2, or 3, to evaluate performance in different networks. Eventually, the first result (out of four) that completes is sent to the application.

We implemented Web-LEGO's client within `goproxy` [26], an HTTP/HTTPS proxy library written in Go [27] which has native concurrency features.

Content Identification A URL is usually composed of four parts: protocol, hostname, pathname, and parameters. In most cases, the suffix of the pathname reveals the file type, e.g., ".png" suggests that the requested content is an image. In some cases, the URL does not unveil which content type is being requested. To address this issue, the Web-LEGO's client analyzes the content received *at the proxy* to identify images, JS, and CSS, and propagate the pair <URL, type> to 3S. *Obfuscated URLs*, or URLs lacking file extensions in the suffix of their pathname, are always forwarded to the 3S where their type can now be retrieved. Finally, we omit sending to 3S advertisement URLs—detected via their hostnames—and social-network and other analytics.

2) Similarity Storage Server: 3S has three main functions: store similarity information about web objects, respond to Web-LEGO clients' requests, query RFSS and cache its responses. We use MongoDB [28]to deploy a non-relational database and store information about URLs and their similarity. For each type (*i.e.,* JS, CSS, and image) we create two collections, which are similar to tables in SQL, yet have no structure. One collection stores all URLs along with their attributes, e.g., for an image URL it stores its category (*i.e.,* image), file size, width, length, type, CDN-flag (*i.e.,* whether it is hosted at a CDN or not), *etc*. The second collection stores the similarity relationships among URLs.

URL Selection Algorithm. Web-LEGO limits the number of candidate URLs to replace a web object to three, as discussed above. When more than three candidate URLs are available, we proceed as follows. First, we give highest priority to CDN-hosted URLs, *i.e.,* URLs having the CDN-flag attribute set.For image URLs, we prioritize identical images, then URLs for which the corresponding image dimensions are closest to the original image size. We find that the file type, e.g.,.png vs .jpg, does not pose a constraint for successful alternative webpage construction, because the CSS files handle the file types automatically. For JS and CSS files, given that the alternative CDN-hosted URLs point to *identical* content to the requested one, we select them randomly. Note that Web-LEGO ignores URLs associated with images that have width or length smaller than 100 pixels. Such images usually do not have enough features for the reverse file search engine to return a same or similar image response. We finally return the top 3 URLs based on the ranking above.

URLs cached at the 3S server might become stale over time, *i.e.,* they might no longer point to the desired content or point to different content. The first scenario is likely to happen over time, yet the consequences for Web-LEGO are not significant; in the worst case, the original content will still be served, because Web-LEGO always requests the original object. The second scenario, *i.e.,* an existing URL pointing to a new object, is unlikely. In either cases, regular automatic cleanups effectively resolve these issues.

3) Reverse File Search Server: RFSS identifies alternative resources for an input web object. Differently from 3S, this operation does not have a real-time constraint, as results are used to populate 3S's database for (eventual) future requests.

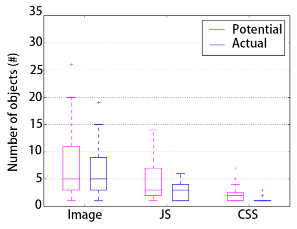
To fully control how to share and replace website resources by the content provider, the sharing flag and replacing flag are defined. Both flags are included in the HTTP(S) header. RFSS will follow the sharing and replacing flags when it looks up alternative resources from the Internet, and further tell the associate information to 3S. The potential content inconsistency and copyright issues could be solved with similarity control headers. Specifically, the resource can be shared when the sharing flag is set to one, otherwise, equal to zero; The resource can be replaced with the similar, same and no resource when the replacing flag is equal to one, two and zero.

Building a reverse file search service is complex, costly and out of the scope of this paper, so we just borrow the power of other existed reverse file search engines. In the following, we first focus on how RFSS deals with images, and later describe how CSS and JS are handled. Although reverse image search is a well explored topic [29], [30], [31], building a reverse image engine is challenging. This is because it requires indexing billions of images and thus a significant engineering and monetary investment. To build a proof of concept of RFSS, we piggyback on two commercial services for reverse image search: TinEye [20] and Google Images [15]. While Google Images is free of use [32], for TinEye we purchased an API with a capacity of 10,000 searches for the price of $300. For each returned image, both engines also report their copyright. To avoid copyright infringement in our evaluations, Web-LEGO only uses "free to use or share" images.

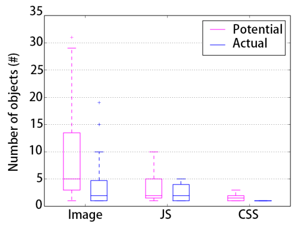
Given an input image/URL, TinEye derives a fingerprint to find equal images within its database (billion of entries). Matching images are returned in the form of a public URL from where they can be downloaded. Instead, Google Images analyzes the input picture, e.g., by identifying objects and/or persons, and return a set of equal and similar images. For each input image, it also returns a *tag*, *i.e.,* a set of keywords describing the content of the image.

Despite Web-LEGO users are informed to be trading performance for potentially different website appearances, our goal is to minimize website *distortion*. Webpage developers use images to convey a message; our goal is to provide an image which captures a similar message, e.g., the Golden Gate bridge from a different angle rather than a different bridge (see Figure 1). Accordingly, for a given input image url, we query both TinEye and Google Images; we then rank the received images based on their *similarity* to the original image. Equal images are ranked the highest; next, similar images are ranked based on how similar their tags are with the original image. Tags for alternative images (provided by Google) are retrieved by performing a reverse file search; tag similarity between original and alternative image is expressed using cosine similarity [33]. Whenever 3S asks for alternative images, RFSS will respond with same images first, then similar images with highest cosine similarities. We find that more than 70% images have equal images with a measurement study in Section IV, so we are sure that we convey the similar message that webpage developers want to deliver.

To the best of our knowledge, no commercial reverse file search service exists for JS and CSS. As above, building such service is out of the scope of this paper. While finding binary-level object matches has been studied in a variety of contexts over the years [34], [35], [36], we resort to a simple mechanism which piggybacks on Google search. Given an input JS/CSS file, we use text snippets from these files to perform Google searches and locate corresponding alternative files. Once *identical* alternative files are found, their URLs is returned to 3S. For JS, two popular CDN-hosted JS frameworks are located: Google-hosted libraries and CDNJS libraries. For CSS, we instead find identical CSS files scattered around independent websites on the Internet.



(a) CDN-less



(b) CDN-hosted

Fig. 5. Evaluation of Web-LEGO's potential vs actual replacement.

IV. Evaluation

A. Metrics and Experiment Settings

Metrics Throughout the evaluation, we use three main metrics: PLT, uPLT, and normalized benefit. The *page load time* (PLT) refers to when a browser returns the *onload* event (see Section VI). This metric is commonly used [1], [4], [5], [16] as a quick method to estimate the performance of a webpage. PLT has been recently criticized because unable to keep up with the evolution of the Web, e.g., it suffers from the presence of JS files [1], [37]. For this reason, we also use the *user-perceived page load time* (uPLT), or the time when a user considers a page "ready to use" as defined in [16], [38], [39].

We also compare Web-LEGO's benefits with respect to the extra network traffic it introduces. We call *normalized benefit* the ratio between milliseconds (of improvement or deterioration) and KBytes (of extra traffic); note that this metric has been previously used in [21]. We compute a webpage's performance improvement (or deterioration) as the difference between the PLT of the original webpage and its PLT when served via Web-LEGO. The extra traffic refers to the additional requests generated by the Web-LEGO's client in the attempt to retrieve (faster) equal or similar content.

Settings We built a testbed composed of a Web-LEGO's client, 3S, and RFSS. The Web-LEGO's client runs on a high-end desktop machine (3.30 GHz, 4 cores,8 GBytes of RAM) with fixed access (up to 100 Mbps in download and upload). The 3S and RFSS run on two regular server machines (3.50 GHz, 8 cores, 8 GBytes of RAM) located in the same LAN as the Web-LEGO's client, *i.e.,* negligible network delay and packet losses. Artificial delay, losses, and bandwidth throttling are introduced using `cell-emulation-util` [40], a script based on Linux TC netem [41] which allows to emulate real-world cellular networks based on traces provided by Akamai.

We selected two sets of 50 websites to be tested: *CDN-hosted* and *CDN-less*. Both sets were randomly selected from Alexa's top 1,000 websites based on whether they are hosted on a CDN or not, according to `whatsmycdn.com` (see Section II). The number of websites under test was chosen to ensure we can collected reasonable feedback from real users; nevertheless, it is twice as large as in [39].

We instrument Chrome via its remote debugging protocol to load a target website while collecting an HTTP Archive (HAR) file, *i.e.,* JSON-formatted file logging the browser's interaction with a site. Each website is loaded directly and via the Web-LEGO's client; each load is repeated 5 times for 30 seconds while a video of the page loading is recorded. As suggested in [16], [42], we ensure a fresh browser by cleaning Chrome state among runs, as well as a "primer" to populate eventual DNS (and 3S) caches at ISP level.

The recordings of website loads, realized with ffmpeg [43], are used for uPLT assessment. We use Eyeorg's *timeline* experiment [16] where a participant is asked to "scrub" the video until when (s)he considers the page to be ready to use. For each website load (original and Web-LEGO) we show to paid participants the video whose PLT is closest to the median of the 5 loads. Each Eyeorg campaign targets 200 paid participants (total cost: $24), who each evaluates 6 videos—thus generating 1,200 uPLT values or about 12 feedback per website's load (original and Web-LEGO).

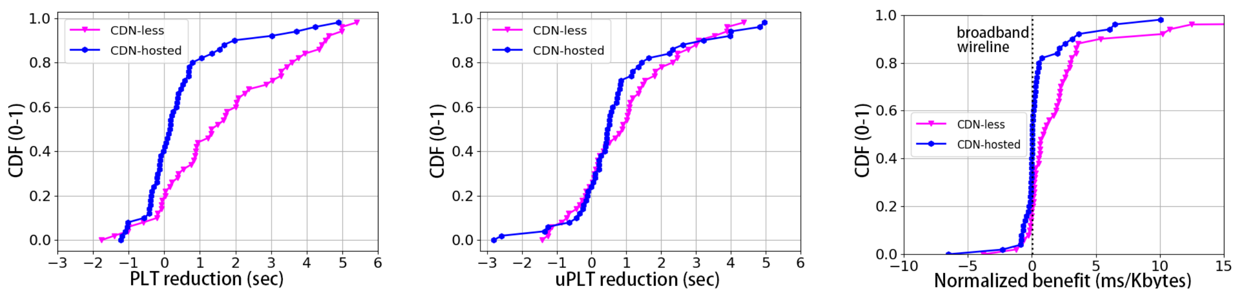
B. Web Objects Characterization

We start by analyzing which web objects have *potential* for replacements and which ones were *actually* replaced in our experiments, *i.e.,* served from alternative resources. Figures 5(a) and 5(b) show boxplots of the number of potential and actual replacements as a function of the object type (image, JS, CSS), differentiating between CDN-hosted and CDN-less websites.

Regardless of whether a website is CDN-hosted or CDN-less, image files hold the most potential for replacement, followed by JS and CSS files. For CDN-less websites, Figure 5(a) shows that, most of the time, potential replacement translates into actual replacement, which means that the identified equal or similar content is received by the Web-LEGO's client faster than the original one. For CDN-hosted websites, Figure 5(b) shows instead that while 50% of these websites contain at least 5 images with candidate replacements, only one of these objects was actually replaced in the median case. This result is not surprising since CDNs are already optimized to serve content close to the user; indeed, this result further strengthens the importance of Web-LEGO for CDN-less websites.

For JS and CSS files, Web-LEGO leverages alternative files which are carbon copies of the original ones; alternative images can instead be either *equal* or *similar*—in terms of cosine similarity (see Section III-B3)—to the original ones. For each original image (1,012 images when combining CDN-hosted and CDN-less websites) we derive the highest *rank* available, *i.e.,* equal or highest cosine similarity, and summarize the results in Table II. The majority of the original images, 73.9%, have at least one equal alternative. Although not shown in the table, more than half (52.1%) of the original images have *three* equal alternative images: since Web-LEGO never attempts to replace an image with more than three alternative images (see Section III), this means that half of the time the user would see no visual change in a Web-LEGO-generated webpage. Further, an extra 11.9% of images have at least one very similar alternative image, *i.e.,* cosine similarity mostly equal to one. Only 14% of the images have low similarity alternative images (cosine similarity <0.5) which have potential to *distort* the semantic of an original image.

Next, we quantify the *accuracy* of cosine similarity computed between image tags as an indicator of how *visually*



(a) CDF of PLT reduction. (b) CDF of median uPLT reduction (c) CDF of normalized benefit

Fig. 6. Performance evaluation ; 50 CDN-less and 50 CDN-hosted websites.

similar two images are (Table II, second row). We resort to visual inspection and find that cosine similarity is a good visual similarity indicator, e.g., 92.4% of the images with perfect cosine similarity are visually similar to their original counterpart. Errors happen when the search engine doesn't correctly tag an input image, triggering the retrieval of bad alternative images which indeed share the (erroneous) tags with the input image. Accuracy decreases as we focus on images with lower cosine similarity; still, 70.3% of the images with cosine similarity $\leq$0.5 are visually similar to the original. By combining the first and the second row of Table II, we can conclude that Web-LEGO has an average accuracy of 94.2%.

C. Baseline Network Experiments

Page Load Time Figure 6(a) shows the Cumulative Distribution Function (CDF) of the (median) *PLT reduction* for 50 CDN-less and 50 CDN-hosted websites, respectively. We compute the PLT reduction as the difference between the median PLT of an original webpage and its median PLT when served via Web-LEGO, *i.e.,* positive values indicate good Web-LEGO's performance.

Figure 6(a) shows that Web-LEGO reduces the PLT for 80% of the CDN-less websites, with reductions in the order of multiple seconds (up to 5.5 secs). Significant PLT reductions are also available for 58% of the CDN-hosted websites. As discussed above, the observed lower benefits for CDN-hosted websites are expected due to the reduced opportunity of content replacement (see Figure 5(b)). Web-LEGO causes *PLT degradations* (negative PLT reductions) for 20% of the CDN-less websites (10 websites) and 42% (21 websites) of the CDN-hosted websites; we analyze such PLT degradations next.

The PLT degradations for 6 of the 10 CDN-less websites impacted are $< 200\ ms$. Such values are, however, within the standard deviation of the PLT measured across 5 runs. It follows that their root causes might lie in non-Web-LEGO related causes such as high(er) load at one of the many servers involved during a webpage load. Conversely, for the remaining 4 websites we observe PLT degradation comprised between 0.6 and 1.7 seconds. HAR inspection suggests that in all cases, third-party ad servers, that Web-LEGO did not interfere with, were the root cause of the degradation.

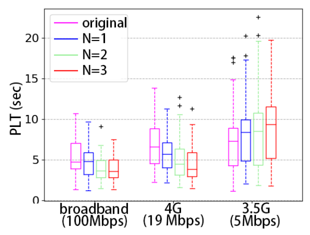
TABLE II
SIMILARITY OF ORIGINAL IMAGES AND ALTERNATIVE ONES ; METRICS: COSINE SIMILARITY AND ACCURACY.

	Equal	**Similar (cosine similarity)**				**None**
		1	**[0.8,1)**	**[0.5, 0.8)**	**[0, 0.5)**	
Ratio	73.9%	7.1%	2.2%	2.6%	14.0%	0.2%
Accu.	100%	92.4%	86.1 %	78.9%	70.3 %	-

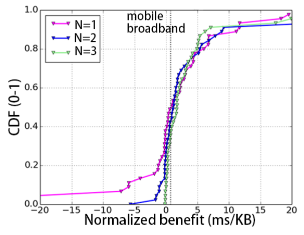
User-perceived Page Load Time Here, we aim to verify that Web-LEGO's performance improvements also hold in terms of *uPLT*, or how fast people perceive a webpage. Accordingly, Figure 6(b) shows the CDF of the median uPLT for both CDN-less and CDN-hosted websites.

Differently from Figure 6(a), the figure shows a similar fraction of CDN-less and CDN-hosted websites (74%) for which users "perceive" an improvement thanks to Web-LEGO. This implies that the verdict of PLT is reverted for 6% of the CDN-less websites, which are felt slower by real users than what PLT indicated, and for 16% of the CDN-hosted websites, which are felt faster than what PLT indicated. This result is due to two reasons. First, as discussed above, PLT is not perfect and does not take into account how the page is actually rendered on screen. Second, humans are not perfect either; note that the two CDFs show that 20% of the uPLT reduction values are comprised within $\pm 200\ ms$, a value very hard to judge by the human eye. The figure also shows large uPLT reduction values, e.g., 60% comprised between half a second and 4-5 seconds where, we can blindly trust the input from the human eye. Similarly, we observe high PLT degradation values for two CDN-hosted websites (2.7 sec); we visually inspected these videos and indeed observed that some advertisements, by design ignored by Web-LEGO, are responsible for the webpage to complete later. Interestingly, few users did not consider this to be an issue and reported much faster "ready to be used" values.

Normalized Benefit To further understand the result above, Figure 6(c) shows the CDF of Web-LEGO's normalized benefit distinguishing between CDN-less and CDN-hosted websites. The normalized benefit was first introduced in [21] to quantify the break-even points for a client choosing to send extra requests, measured in terms of the amount of improved performance (milliseconds) to the amount of added traffic (KBytes). In the most relevant scenario for our experiments here, *i.e.,* a broadband wireline service, the model implies that as long as extra traffic induced by the client generates the benefit of at least 0.029 ms/KB, it pays of economically for the client to generate extra utilization. We plot this value as a vertical line at $x = 0.029$ in Figure 6(c).



(a) CDF of PLT.



(b) CDF of normalized benefit (4G).

Fig. 7. PLT and normalized benefit under challenging network conditions ; CDN-less websites.

The figure shows that Web-LEGO's normalized benefit is above the 0.029 threshold in most cases, *i.e.,* 80% of CDN-less cases and in 58% of CDN-hosted cases. The significant benefits, particularly for CDN-less websites, imply that there is additional room for increasing the number of requests for alternative content. At the same time, the results for CDN-hosted websites imply that a more conservative approach should be taken. This is because normalized benefit is only slightly above the threshold in most cases. While the value of 3 additional requests provides a reasonable trade-off in this particular scenario, we conclude that a more fine-grained method, capable of dynamically adjusting the amount of additional traffic on-the-fly, would provide additional benefits.

D. Challenging Network Conditions

We here evaluate the impact of worsen network conditions, *i.e.,* slower speed, higher latency and losses, on Web-LEGO's performance. We select two scenarios representative of 4G and 3.5G connectivity, respectively, in terms of download bandwidth. For 4G, we limit the download bandwidth to 19 Mbps; we computed this value as the average download speed for 9 LTE networks in the US, based on a recent study [44]. For 3.5G, we set the download rate to 5 Mbps, according to [45]. For this analysis, we focus on the 50 CDN-less websites.

Figure 7(a) shows boxplots of the PLT as a function of the network connectivity (broadband, 4G, and 3.5G) as well as N, or the maximum number of *additional* parallel content requests sent by the Web-LEGO's client for each replaceable object. We denote as "original" the PLT of a website when loaded directly. Differently from above (see Figure 6(a)) we plot the PLT instead of the PLT reduction to highlight the PLT variation in presence of variable network connectivity.

We also can be concluded from Figure 7(a) that Web-LEGO is beneficial in both the broadband and 4G scenarios. In particular, as the number of additional requests grows, the performance improves. Despite the additional traffic downloaded in such scenarios, no significant congestion occurs in the download direction. Hence, the Web-LEGO client manages to utilize a larger number of alternative resources to reduce latency and improve performance. We notice that the improvement relative to the original case is *larger* in the 4G case, which is more constrained than the broadband one. For example, the $N = 3$ median case improves upon original by 20% in the broadband scenario, and by 45% in the 4G scenario. A similar effect was first analyzed in [22], where it was observed that the benefits of request replication increase at medium over low loads.

Differently from [22], which found that request replication is beneficial at high utilization, we found that this does not hold for Web-LEGO. Figure 7(a) shows that in the case of a 3.5G network, characterized with a download speed of 5 Mbps, requesting alternative content can cause the PLT to *increase*. For example, for $N = 1$, it increases by 14% in the median case, and for $N = 3$, it increases by 28%.

In the 3.5G scenario, the additional traffic fetched by a Web-LEGO client creates short-term congestion in the down-link direction, which effectively degrades the overall performance, including the performance of the connections to the origin servers. In these or worse network conditions, Web-LEGO has several options: a Web-LEGO client should either refrain from sending alternative requests. Alternatively, Web-LEGO could implement fine-grained algorithms that would opportunistically select a subset of replaceable objects.

Normalized Benefit. Figure 7(a) shows that Web-LEGO provides clear and substantial performance benefits in the mobile broadband scenario. Here, we explore if such performance improvements are of sufficient *economic* benefit to the clients. We again refer to the cost-benefit analysis of low latency via added utilization [21] which found that, in the case of a mobile broadband plan, the normalized benefit should be above a 0.74 ms/KB threshold in order to be economically viable.

Figure 7(b) shows Web-LEGO's normalized benefit in 4G, *i.e.,* the 19 Mbps LTE mobile broadband service. As a reference point, we add the $x = 0.74$ line in the figure, which corresponds to the economic break-even point discussed above. The figure shows that for all numbers of added requests per object, the benefit is still larger than 50%. Most importantly, the figure shows that $N = 3$ not only brings the largest benefit, *i.e.,* in more than 62% of scenarios, but despite the larger overhead (*i.e.,* on average 3x relative to the $N = 1$ case), it manages to bring the corresponding PLT reduction. As a minor note, we investigated the long negative tail shown in Figure 7(b), for $N = 1$. We found that it is induced by a poorly-responsive ad server, that Web-LEGO did *not* interfere with, associated with one of investigated websites. On top of that, the amount of alternative content fetched by Web-LEGO was only 7 KBytes, which caused the long negative tail.

V. Discussion

Advanced alternative content selection The normalized benefit (milliseconds/KBytes) depends on the amount of extra content requested by the Web-LEGO's client, and the reduction in PLT that such extra traffic brings. While simple heuristics could be easily and safely drawn from our results, e.g., "send 3 alternative requests for a CDN-less website on a broadband wireline connection", more sophisticated and fine-grained algorithms could be developed. Such algorithms would determine the right level of request redundancy based, for example, on the PLT of a particular website, Web-LEGO setup, the underlying network type, and whether a website is hosted on a CDN or not. Moreover, in resource constrained scenarios it might not be beneficial to replace every single object, even if alternative sources are available. In such a case, it would be necessary to select a subset of the replaceable objects, further raising the question which objects are more important than others in the context of user perception [4], [39].

Security One such hypothetical scenario arises due to a discrepancy between time-of-check *vs.* time-of-use of a file used by Web-LEGO. In particular, if an attacker is capable of hijacking an origin server, then it can inject malware to a Web-LEGO client. There are several ways to address such a potential scenario. First, the RFSS server should only rely on reputable websites using information from Alexa [46], WoT [47], etc. The ability for an adversary to promote a malicious website into the group of reputable sites is challenging, and the ability for an adversary to hijack a reputable website is again challenging. Even if such an event happens, the adversary is far more likely to conduct *direct* and more potent attacks (e.g., phishing), than to "hunt" for Web-LEGO's clients—whose requests are hard to distinguish from one originated by more classic clients.

Economic sustainability The key involved entities are: Web-LEGO clients, CDNs, CDN-less websites, and *origin websites* or websites whose content might be offered as an alternative by Web-LEGO. Clients and CDN-less websites directly benefit from Web-LEGO's page load time speedup. CDNs also directly benefit from Web-LEGO because of additional traffic and thus revenues increase. Conversely, there is no obvious benefit for origin websites. However, Web-LEGO could target any web objects that content providers would like to share and replace (see Section III-B3). Further, while the economics behind each website are different and complex, there is one shared goal which directly relates to increased revenues: webpage ratings [46] and search engines rankings. It is important to note that such webpage ratings and search engine rankings are impacted by traffic-based metrics [48], [49]. It follows that Web-LEGO-induced traffic towards origin websites, helps their general ratings and search engine rankings.

VI. Related Work

To the best of our knowledge, the work in [50] (for files) and in [51] (for videos) are the only previous attempts to look into *content similarity* to improve an application performance, *i.e.,* Web-LEGO's core idea. In particular, [50] introduces a system that utilizes file similarity to speed-up large file transfers for multi-source downloads, such as BitTorrent. The authors provide a method to find similar objects and locate chunks within such similar objects that are *identical* to the chunks in the original file. While similar in spirit, Web-LEGO departs from this approach for two key reasons. First, it abandons content *strictness* which is still a requirement for the above solutions. Second, it targets the Web and mobile applications.

Web-LEGO's main goal is to speedup webpage loading time to improve a user's QoE. Many recent solutions share the same goal and achieve so by optimizing/compressing a webpage dependency graph. KLOTSKI [4] is a system that aims at improving user experience by prioritizing the content most relevant to a user's preferences. Shandian [1] is another system whose goal is to remove the intrinsic inefficiencies of webpage loads. It removes such inefficiencies by directly manipulating the dependency graph, *i.e.,* controlling what portions of the page and in which order they are communicated to the browser. Netravali et al. [5] notice that a browser is only partially informed of a page's dependency graph. With the goal to minimize wasted times, the authors propose fine-grained dependency graphs with no missing edges. Prophecy [12] is a system for accelerating mobile webpages. It first precomputes the JavaScript heap and the DOM tree of a webpage, then the server sends a write log including the computed information to the mobile browser. The mobile browser can replay with the log to reconstruct the final page state without intermediate computations to speedup webpages. Vaspol et al. [6] propose VROOM which aims to speedup mobile webpages via server-aided dependency resolution. To preserve the end-to-end nature of the web, the web servers assist clients to find resources when clients fetch every resource from the servers. Nutshell [11] is devised to solve the scaling challenge of the proxy based webpage accelerating solution via minimizing the proxy computational overheads. It can identify and execute only the necessary JS code to push the required objects for loading a webpage, while ignoring other codes.

VII. Conclusion

This paper has proposed Web-LEGO, a client side mechanism which trades content "strictness" for faster webpages. We first validated Web-LEGO's idea via crowdsourcing, *i.e.,* interrogating a thousand users on their interest on slightly modified webpages in exchange for a faster browsing experience. Next, we design and implement Web-LEGO as a high performing opt-in service, for both end-users and content providers. Finally, we use both automated tests and paid participants to demonstrate Web-LEGO's effectiveness. Our results show that Web-LEGO offer speedups of up to 5 seconds in user perceived page load time. Most gains originated in broadband wireline and wireless scenarios, and for websites not using a CDN. We further showed that fetching additional content is economically viable in evaluated broadband scenarios.

Acknowledgement

The corresponding author is Pengfei Wang. This work was supported by the National Key Research and Development Program of China (grant 2018YFC0910500), the National Natural Science Foundation of China (grants U1908214, 61872052, 62071083, 62072094, 61672148), the National Science Foundation NeTS Program (grants CNS-1810582, CNS-1615837, CNS-1526052), the Liaoning Key R&D Program (grants 2019JH2/10100030, 2020JH2/10100046), the Fundamental Research Funds for the Central Universities (grants DUT20RC(3)039, N182608004, N180101028), the CERNET Innovation Project (grant NGII20190504), and China Scholarship Council (grant 201606080049).

References

[1] X. S. Wang, A. Krishnamurthy, and D. Wetherall, "Speeding up web page loads with shandian," in *13th USENIX Symposium on Networked Systems Design and Implementation*, Mar. 2016, pp. 109–122.

[2] P. Wang, M. Varvello, and A. Kuzmanovic, "Kaleidoscope: A crowdsourcing testing tool for web quality of experience," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 1971–1982.

[3] "Shopzilla: faster page load time = 12 percent revenue increase," 2016, http://www.strangeloopnetworks.com/resources/infographics/web-performance-andecommerce/shopzilla-faster-pages-12-revenue-increase/.

[4] 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*, May 2015, pp. 439–453.

[5] R. Netravali, A. Goyal, J. Mickens, and H. Balakrishnan, "Polaris: Faster page loads using fine-grained dependency tracking," in *13th USENIX Symposium on Networked Systems Design and Implementation*, Mar. 2016.

[6] V. Ruamviboonsuk, R. Netravali, M. Uluyol, and H. V. Madhyastha, "Vroom: Accelerating the mobile web with server-aided dependency resolution," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 390–403.

[7] S. Mardani, M. Singh, and R. Netravali, "Fawkes: Faster mobile page loads via app-inspired static templating," in *17th USENIX Symposium on Networked Systems Design and Implementation*, 2020, pp. 879–894.

[8] "Akamai," 2020, http://www.akamai.com/.

[9] "Google CDN Platform," 2020, https://cloud.google.com/cdn/docs/.

[10] "CloudFlare," 2020, https://www.cloudflare.com/.

[11] A. Sivakumar, C. Jiang, Y. S. Nam, S. Puzhavakath Narayanan, V. Gopalakrishnan, S. G. Rao, S. Sen, M. Thottethodi, and T. Vijaykumar, "Nutshell: Scalable whittled proxy execution for low-latency web over cellular networks," in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, 2017, pp. 448–461.

[12] R. Netravali and J. Mickens, "Prophecy: Accelerating mobile page loads using final-state write logs," in *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation*, 2018.

[13] R. Netravali, A. Sivaraman, J. Mickens, and H. Balakrishnan, "Watchtower: Fast, secure mobile page loads using remote dependency resolution," in *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, 2019, pp. 430–443.

[14] "What's My CDN," 2020, http://whatsmycdn.com/.

[15] "Google Images," 2020, https://images.google.com/.

[16] M. Varvello, J. Blackburn, D. Naylor, and K. Papagiannaki, "Eyeorg: A platform for crowdsourcing web quality of experience measurements," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016, pp. 399–412.

[17] "Baidu image," 2020, http://image.baidu.com/.

[18] "Bing Images," 2020, https://www.bing.com/images.

[19] "Musixmatch," 2017, https://www.musixmatch.com/.

[20] "TinEye: Reverse Image Search," 2020, https://tineye.com/.

[21] A. Vulimiri, P. Godfrey, S. V. Gorge, Z. Liu, and S. Shenker, "A cost-benefit analysis of low latency via added utilization," *arXiv preprint arXiv:1306.3534*, 2013.

[22] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, 2013, pp. 283–294.

[23] A. Vulimiri, O. Michel, P. Godfrey, and S. Shenker, "More is less: reducing latency via redundancy," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, 2012, pp. 13–18.

[30] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3128–3137.

[24] "The history of the do not track header," 2020, http://paranoia.dubfire.net/2011/01/history-of-do-not-track-header.html.

[25] Google, "The Chromium Projects," https://www.chromium.org/.

[26] "GoProxy," 2020, https://github.com/elazarl/goproxy.

[27] "The Go Programming Language," 2020, https://golang.org/.

[28] "mongoDB," 2020, https://www.mongodb.com/.

[29] J. Jarvis, *What would Google do?: Reverse-engineering the fastest growing company in the history of the world*. Harper Business, 2011.

[31] M. Gaillard and E. Egyed-Zsigmond, "Large scale reverse image search," in *INFORSID*, 2017, p. 127.

[32] "Best reverse image search engines, apps and its uses (2016)," 2020, https://beebom.com/reverse-image-search-engines-apps-uses/.

[33] A. Singhal, "Modern information retrieval: A brief overview," *IEEE Data Eng. Bull.*, vol. 24, no. 4, pp. 35–43, 2001.

[34] S. C. Rhea, K. Liang, and E. Brewer, "Value-based web caching," in *Proceedings of the 12th International Conference on World Wide Web*, 2003, pp. 619–628.

[35] K. Park, S. Ihm, M. Bowman, and V. S. Pai, "Supporting practical content-addressable caching with czip compression." in *USENIX Annual Technical Conference*, 2007, pp. 185–198.

[36] R. Netravali and J. Mickens, "Remote-control caching: Proxy-based url rewriting to decrease mobile browsing bandwidth," in *Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications*, 2018, pp. 63–68.

[37] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "Demystifying page load performance with wprof," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation*, 2013, pp. 473–485.

[38] Q. Gao, P. Dey, and P. Ahammad, "Perceived performance of top retail webpages in the wild: Insights from large-scale crowdsourcing of above-the-fold qoe," in *Proceedings of the Workshop on QoE-based Analysis and Management of Data Communication Networks*, 2017, pp. 13–18.

[39] C. Kelton, J. Ryoo, A. Balasubramanian, and S. R. Das, "Improving user perceived page load times using gaze," in *14th USENIX Symposium on Networked Systems Design and Implementation*, 2017, pp. 545–559.

[40] Akamai, "A tool to emulated real-world cellular networks," https://github.com/akamai/cell-emulation-util.

[41] Linux Foundation, "netem provides network emulation functionality in Linux." https://wiki.linuxfoundation.org/networking/netem.

[42] M. Varvello, K. Schomp, D. Naylor, J. Blackburn, A. Finamore, and K. Papagiannaki, "Is the web http/2 yet?" in *International Conference on Passive and Active Network Measurement*, 2016, pp. 218–232.

[43] "FFmpeg," 2020, https://www.ffmpeg.org/.

[44] "The fastest wireless networks of 2017," 2020, https://brianchan.us/2017/10/02/the-fastest-wireless-networks-of-2017/.

[45] "How fast is 4g wireless service?" 2020, https://www.lifewire.com/how-fast-is-4g-wireless-service-577566.

[46] "Alexa top sites," 2020, https://aws.amazon.com/alexa-top-sites/.

[47] "Web of Trust," 2020, https://www.mywot.com/.

[48] "Queries and clicks may influence google's results more directly than previously suspected," 2020, https://moz.com/rand/queries-clicks-influence-googles-results/.

[49] A.-J. Su, Y. C. Hu, A. Kuzmanovic, and C.-K. Koh, "How to improve your search engine ranking: Myths and reality," *ACM Transactions on the Web*, vol. 8, no. 2, p. 8, 2014.

[50] H. Pucha, D. G. Andersen, and M. Kaminsky, "Exploiting similarity for multi-source downloads using file handprints." in *NSDI*, 2007.

[51] A. Anand, A. Balachandran, A. Akella, V. Sekar, and S. Seshan, "Enhancing video accessibility and availability using information-bound references," *IEEE/ACM Transactions on Networking*, vol. 24, no. 2, pp. 1223–1236, Apr. 2016.