Improving User Perceived Page Load Time Using Gaze

Conor Kelton*, Jihoon Ryoo*, Aruna Balasubramanian, and Samir R. Das
Stony Brook University
*student authors with equal contribution

Abstract

We take a fresh look at Web page load performance from the point of view of user experience. Our user study shows that perceptual performance, defined as *user-perceived page load time (uPLT)* poorly correlates with traditional page load time (PLT) metrics. However, most page load optimizations are designed to improve the traditional PLT metrics, rendering their impact on user experience uncertain. Instead, we present *WebGaze*, a system that specifically optimizes for the uPLT metric. The key insight in WebGaze is that user attention and interest can be captured using a user's eye gaze and can in turn be used to improve uPLT. We collect eye gaze data from 50 users across 45 Web pages and find that there is commonality in user attention across users. Specifically, users are drawn to certain regions on the page, that we call regions of high collective fixation. WebGaze prioritizes loading objects that exhibit a high degree of collective fixation to improve user-perceived latencies. We compare WebGaze with three alternate strategies, one of which is the state-of-the-art system that also uses prioritization to improve user experience. Our evaluation based on a user study shows that WebGaze improves median uPLT for 73% of the Web pages compared to all three alternate strategies.

1 Introduction

Web performance has long been crucial to the Internet ecosystem since a significant fraction of Internet content is consumed as Web pages. As a result, there has been a tremendous effort towards optimizing Web performance [21, 39, 60]. In fact, studies show that even a modest improvement in Web performance can have significant impact in terms of revenue and customer base [15, 16, 34].

The goal of our work is to improve page load performance, also called the Page Load Time (PLT), from the perspective of the user. PLT is typically measured using objective metrics such as *OnLoad* [11], and more recently *Speed Index* [32]. However, there is a growing concern that these objective metrics do not adequately capture the user experience [2, 5, 42, 49].

As a first step, we define a *perceptual* variation of page load time that we call *user-perceived PLT*, or *uPLT*. We conduct a systematic user study to show what was anecdotally known, i.e., uPLT does not correlates well with the OnLoad or Speed Index metrics. However, almost all current Web optimization techniques attempt to optimize for the OnLoad metric [10, 39, 47, 52, 62] rendering their impact on user experience uncertain. The problem is that improving uPLT is non-trivial since it requires information about user's attention and interest.

Our key intuition is to leverage recent advances in eye gaze tracking. It is well known that user *eye gaze* – in terms of fixation, dwell time, and search patterns – correlate well with user attention [17, 55]. In the human visual system only a tiny portion (about 2°) at the center of the visual field is perceived with the highest visual acuity and the acuity sharply falls off as we go away from the center [57]. Thus the eye must move when a user is viewing different parts of the screen. This makes eye gaze a good proxy for user's attention. Further, the commoditization of gaze trackers allow accurate tracking using low cost trackers [35, 37, 41, 51], without the need for custom imaging hardware.

We design *WebGaze*, a system that uses gaze tracking to significantly improve uPLT. WebGaze prioritizes objects on the Web page that are more visually interesting to the user as indicated by the user's gaze. In effect, WebGaze encodes the intuition that loading "important" objects sooner improves user experience. The design of WebGaze has two main challenges: *(i)* Scalability: personalizing the page load for each user according to their gaze does not scale to a large number of users, and *(ii)* Deployability: performing on-the-fly optimizations based on eye gaze is infeasible since page loads are short-lived and the gaze tracker hardware may not be available with every user.

WebGaze addresses these challenges by first distilling gaze similarities across users. Our gaze user study shows that most users are drawn to similar objects on a page. We divide the page into visually distinctive areas that we call regions and define the *collective fixation* of a region as the fraction of users who fixate their gaze on the region. Our study with 50 users across 45 Web pages shows that a small fraction of the Web page has extremely high collective fixation. For example, of the Web pages in our study, at least 20% of the regions were viewed by 90% of the users. Whereas, at least a quarter of the regions of the page are looked at by less than 30% of the users.

WebGaze then uses the HTTP/2 Server Push [13, 30] mechanism to prioritize loading objects on the page

that exhibit high degree of collective fixation. In fact, WebGaze provides a content-aware means of using the HTTP/2 Server Push mechanism. WebGaze does not require gaze tracking on-the-fly or require that every user participates in gaze tracking, as long as enough users participate to estimate the collective fixation. WebGaze's algorithm not only pushes the objects of interest, but also all dependent objects as obtained using the WProf tool [60].

The goal of WebGaze is to improve uPLT, a subjective metric that depends on real users. Therefore, to evaluate WebGaze, we conduct an extensive crowd-sourced user study to compare the performance of WebGaze's optimization with three alternatives: *Default*, *Push-All*, and *Klotski* [21]. Default refers to no prioritization. The Push-All strategy indiscriminately prioritizes all objects. Klotski is the state-of-the-art system whose goal is to improve Web user experience: Klotski works by prioritizing objects that can be delivered within the user's tolerance limit (5 seconds). We conduct user studies across 100 users each to compare WebGaze with each alternative.

The results show that WebGaze improves the median uPLT over the three alternatives for 73% of the 45 Web pages. In some cases, the improvement of WebGaze over the default is 64%. While the gains over the default case come from prioritizing objects in general, the gains over Push-All and Klotski come from prioritizing the right set of objects. *All user study data and videos of Web page loads under WebGaze and each alternative strategy can be found at* `http://gaze.cs.stonybrook.edu`.

2 Page Load Metrics

To study the perceptual performance of Web page loads, we define a perceptual variation of the PLT metric, that we call uPLT or user-perceived Page Load Time. uPLT is the time between the page request until the time the user 'perceives' that the page is loaded. In this section, we provide a background on traditional PLT metrics and qualitatively describe why they are different from uPLT. In the next section, we use a well posed user study to quantitatively compare traditional PLT metrics and uPLT.

OnLoad: PLT is typically estimated as the time between when the page is requested and when the OnLoad event is fired by the browser. The `OnLoad` event is fired when *all* objects on the page are loaded [8]. There is a growing understanding that measuring PLT using the OnLoad event is insufficient to capture user experience [2, 5, 49]. One reason is that users are often only interested in *Above-the-Fold (AFT)* content, but the OnLoad event is fired only when the entire page is loaded, even when parts of the page are not visible to the user. This leads to the OnLoad measure over-estimating the user-perceived latency. But in some cases, OnLoad can underestimate uPLT. For example, several Web pages load additional objects *after* the OnLoad event is fired. If the additional loads are critical to user experience, the PLT estimated based on the OnLoad event will underestimate uPLT. Variants of the OnLoad metric such as `DOMContentLoaded` [8, 60], are similarly disjoint from user experience.

Speed Index: Recently, `Speed Index` [32] was proposed as an alternate PLT measure to better capture user experience. Speed Index is defined as the average time for all AFT content to appear on the screen. It is estimated by first calculating the visual completeness of a page, defined as the pixel distance between the current frame and the "last" frame of the Web page. The last frame is when the Web page content no longer changes. Speed Index is the weighted average of visual completeness over time. The Speed Index value is lower (and better) if the browser shows more visual content earlier.

The problem with the visual completeness measure (and therefore Speed Index) is that it does not take into account the relative importance of the content. This leads to over- or under-estimation of user-perceived latency. If during the page load, a majority of visual components are loaded quickly, Speed Index estimates the page load time to be a small value. However, if the component critical to the user has not yet been loaded, the user will not perceive the page to be loaded. In other cases, Speed Index overestimates. For example, if a large portion of the page has visual content that is not interesting to the user, Speed Index will take into account the time for loading all the visual content, even though the user may perceive the page to be loaded much earlier.

Motivating Example: Figure 1 shows the `energystar.gov` page, and the three snapshots taken when the page was considered to be loaded according to the Speed Index, uPLT, and OnLoad metrics. In the case of uPLT, we choose the median uPLT value across 100 users who gave feedback on their perceived page load time (§3).

Speed Index considers the page to be loaded much earlier, at 3.2 seconds, even though the banner image is not loaded. For the users, the page is not perceived to be completely loaded unless the banner is loaded, leading to Speed Index under-estimating uPLT. On the other hand, the OnLoad metric estimates the page to be loaded 4 seconds *after* the user perceives the page to be loaded, even though their snapshots are the same visually. This is because the OnLoad event fires only when the entire page, including the non-visible parts, are loaded. This illustrative example shows one case when the traditional PLT metrics do not accurately capture user experience.

(a) Speed Index = 3.7 seconds (b) Median uPLT = 8.2 seconds (c) OnLoad = 12 seconds

Figure 1: Snapshots of the page load of `energystar.gov` shown at the Speed Index, the median uPLT across 100 users, and OnLoad values.

3 uPLT User Study

We conduct a user study to systematically compare uPLT with traditional PLT metrics, with the goal of verifying our observations presented in §2.

3.1 Set Up

Our user study was conducted (1) in the lab, and (2) online using crowd-sourcing. For the lab-based study we recruit subjects from our university. The user subjects belong to the age group of 25 to 40, both male and female. The online study is conducted on the Microworkers [9] platform. We present results from 100 users, 50 from each study. *All user studies presented in this paper were approved by the Institutional Review Board of our institution.*

3.2 User Study Set Up and Task

A key challenge of conducting Web page user studies *in-the-wild* is that the Web page load timings experience high variance [61]. The uPLT feedback from two users for a given page may not be comparable under such high variance. To conduct repeatable experiments we capture *videos* of the page load process. The videos are captured via `ffmpeg` at 10 fps with 1920x1080 resolution as the page loads. The users see the video instead of experiencing an actual page load on their computers. This way, each user views exactly the same page load process.

The primary task of the user is to report their perceived page load time when they are browsing the page. We ask the user to view the Web page loading process and give feedback (by pressing a key on the keyboard) when they perceive that the page is loaded. There is an inevitable reaction time between when a user perceives the page to be loaded and when they enter the key. For all measurements, we correct for the user's reaction time using calibration techniques commonly used in user-interaction studies [6]. To ensure high quality data from the user study, we remove abnormally early or late responses. To do so we utilize the `First Paint` and `Last Visual Change` PLT metrics [31]. The First Paint is the time between when the URL begins to load and the first pixel is rendered, and the Last Visual Change is the time when the final pixel changes on the user's screen. Any responses before the First Paint and after the Last Visual Change events are rejected.

3.2.1 Web Pages

In the default case, we choose 45 Web pages from 15 of the 17 categories of Alexa [3], ignoring Adult pages and pages in a language other than English. From each category, we *randomly* choose three Web pages; one from Alexa ranking 1–1000, another from Alexa ranking 10k–20k, and the other from Alexa ranking 30k+. This selection provides wide diversity in the Web pages. The network is standardized to the accepted DSL conditions [63], 50ms RTT, 1.3Mbps downlink and 384Kbps uplink, using the Linux traffic controller '`tc`' [18].

We conduct additional user studies by varying network conditions using the `tc` tool [18] to emulate: i) WiFi-like conditions: a 12 ms RTT link with 20 Mbps download bandwidth and ii) 3G-like conditions: a 150 ms RTT link with a 1.6 Mbps download bandwidth. We conduct these additional user studies across 30 users and 23 Web pages, half from the top 100 and remaining from between 10k–20k Web pages from Alexa's list [3].

3.2.2 Measurement Methodology

We load the Web page using Google Chrome version 52.0.2743.116 for all loads. We do not change the Web load process, and all the objects, including dynamic objects and ads, are loaded without any changes.

When the Web page load finishes, we query Chrome's Navigation Timeline API remotely through its Remote Debugging Protocol [22]. Similar interfaces exist on most other modern browsers [38]. From the API we are able to obtain timing information including the OnLoad measure. To estimate Speed Index, we first record the videos of the pages loading, recorded at 10 frames-per-second. The videos are fed through the WebPageTest Tool [63] that calculates the Speed Index.

3.3 Comparing uPLT with OnLoad and Speed Index

First, we compare the uPLT variations across lab-based and crowd-sourced studies for the same set of Web pages. Figure 2 shows the uPLT box plots for each Web page across the two different studies. Visually from the plot, we find that the lab and crowd-sourced users report similar distributions of uPLT. The standard deviation of the median uPLT difference between the lab and the crowd-sourced study for the same Web page is small, about 1.1 seconds. This same measure across Web pages is much larger, at about 4.5 seconds. This increases our

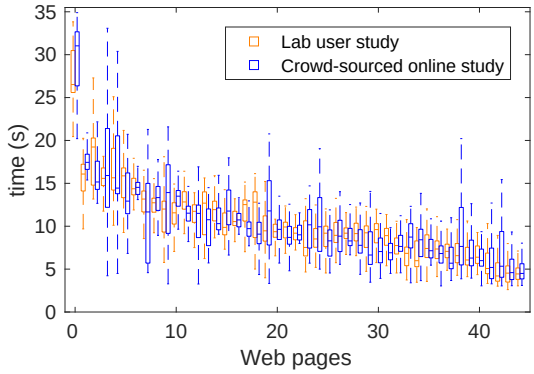

Figure 2: Comparing the uPLT box plot between the 50 lab and 50 crowd-sourced users. Although the uPLT values vary across users, the distributions are similar for the two data sets.

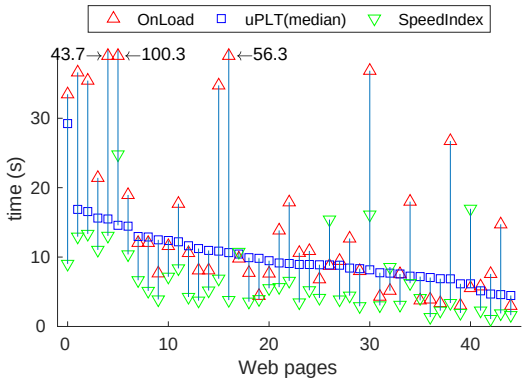

Figure 3: Comparing median uPLT with OnLoad and Speed Index across 45 Web pages and 100 users. The median uPLT is lower than OnLoad for 50% of the Webpages, and higher than Speed Index for 87% of Webpages.

confidence in the results from the crowd-sourced user study; we leverage a similar crowd-sourced study to evaluate WebGaze.

Figure 3 shows median uPLT compared to the OnLoad and Speed Index metrics across the 45 pages and 100 users, combining the crowd-sourced online study and the lab study. The Speed Index and OnLoad values are calculated from the same Web page load in which was recorded and shown to the users.

We observe that uPLT is not well correlated with the Onload and Speed Index metrics: the Correlation Coefficient between median uPLT and the OnLoad metric is ≈ 0.46 while the correlation between median uPLT and the Speed Index is ≈ 0.44. We also find the correlation between uPLT and the DomContentLoaded to be ≈ 0.21.

The OnLoad metric is about 6.4 seconds higher than the median uPLT on an average, for close to 50% of the pages. For 50% of Web pages, the OnLoad is lower than the median uPLT by an average of about 2.1 seconds. On the other hand, Speed Index, estimated over visible AFT content, is about 3.5 seconds lower than uPLT for over 87% of the Web pages. In Section 2 we discussed the cases in which the OnLoad and Speed Index can over and underestimate the user perceived page loads. From our results we see that while cases of uPLT over and underestimation occur in equal proportion for the OnLoad, the case of uPLT underestimation, as shown in Figure 1, occurs more for the Speed Index.

3.4 uPLT Across Categories

The 45 Webpages used in the study have diverse characteristics. In Figure 4, we study how uPLT differs from traditional PLT metrics for different categories. Each point in the plot is the median uPLT across 100 users.

We divide the Web pages across four (4) categories: *(i)* Light html: landing pages such as `google.com`, *(ii)* CSS-heavy; *(iii)* Javascript-heavy; and *(iv)* Image-heavy. To categorize the page into the latter three categories, we look at the types of objects downloaded for each page and count the number of CSS, Javascript, and images. The categories are based on the type of object that is fetched most when the page is loaded.

Light html and *CSS-heavy* pages are simple and see little difference between the uPLT and the OnLoad and Speed Index metrics. However, for pages with a lot of dynamic Javascript, the median difference between uPLT and OnLoad is 9.3 seconds. Similarly, for image-heavy pages, the difference between uPLT and OnLoad is high. This is largely because, as the number of images and dynamic content increases, the order in which the objects are rendered becomes important. As we show in the next section, users typically only focus on certain regions of the page and find other regions unimportant, making it critical that the important objects are loaded first.

3.5 Varying Network Conditions

Finally, to verify the robustness of our results, we analyze the differences between uPLT OnLoad, and Speed Index under varying network conditions.

Under the slower 3G-like network conditions across 30 lab users and 23 Web pages, median uPLT poorly correlates with OnLoad and Speed Index with a correlation coefficient of 0.55 and 0.51 respectively. The median uPLT was greater than Onload 46% of times, with a median difference of 4.7 seconds. The uPLT was less than Speed Index 72% of the time with the median difference of 1.86 seconds. When we evaluate under WiFi-Like conditions we find the correlation between between

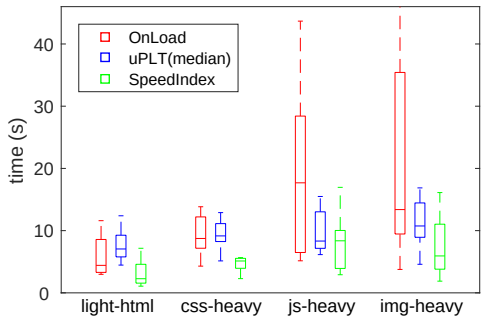

Figure 4: OnLoad, SpeedIndex and uPLT for different categories of Web pages

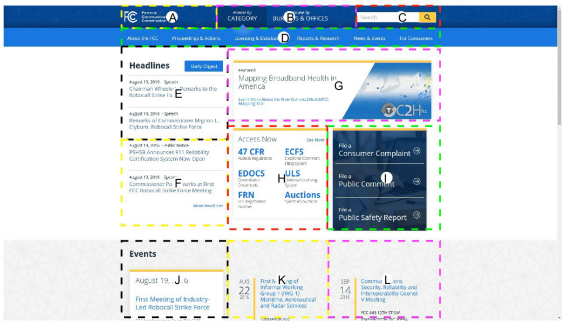

Figure 5: Segmentation of the Web page of fcc.gov into visual regions. The visual regions are named "A", "B", "C", etc.

OnLoad and uPLT is much higher at .74. This result is likely because in faster networks, more pages load instantaneously causing the user perceived latency to not differ much from the OnLoad.

4 Gaze Tracking

Existing Web page optimizations focus on improving traditional PLT metrics. However, our analysis shows that traditional PLT metrics do not correlate well with uPLT, rendering the effect of existing optimizations on user experience unclear. Instead, we propose to leverage users' eye gaze to explicitly improve uPLT.

4.1 Inferring User Interest Using Gaze

Gaze tracking has been widely used in many disciplines such as cognitive science and computer vision to understand visual attention [23, 40]. Recently, advances in computer vision and machine learning have also enabled low cost gaze tracking [35,37,41,51]. The low cost trackers do not require custom hardware and take into account facial features, user movements, a user's distance from the screen, and other user differences.

WebGaze leverages the low cost gaze trackers to capture visual attention of users. As a first step, we conduct a user study to collect eye gaze from a large number of users across Web pages. Using gaze data collected using both a low cost gaze tracker and an expensive custom gaze tracker, we show that the tracking accuracy of the low cost tracker is sufficient for our task.

Next, we analyze the collected gaze data to infer user patterns when viewing the same Web page. Specifically, we identify the *collective fixation* of a region on the Web page, which presents a measure to represent how much a broad group of users attention is fixated on the specific region. WebGaze uses collective fixation as a proxy for user interest, and leverages it to improve uPLT.

4.2 Gaze User Study Set Up

The gaze user study set up is similar to the lab user study described in §3.1. Recall that in our lab user study, we collect uPLT feedback from 50 users as they browse 45 Web pages. In addition to obtaining the uPLT feedback, we also also capture the user's eye gaze.

The gaze tracking is done using an off-the-shelf webcam-based software gaze tracker called GazePointer [27]. GazePointer tracks gaze at 30 frames/sec and does not require significant computational load because it uses simple linear regression and filtering techniques [37, 56] unlike gaze trackers that require more complicated machine learning [25]. We use a 1920 x 1080 resolution 24 inch monitor with a webcam mounted on top. The screen is placed at a reading distance ($\approx$ 50cm) from the participant. We perform a random point test, where we ask the users to look at 100 predetermined points on the screen. We find the error of tracker to be less than 5° at the 95th percentile.

The user study requires gaze calibration for each user; we perform this calibration multiple times during the study to account for users shifting positions, shifting directions, and other changes. We can potentially replace this calibration requirement using recent advances in gaze tracking that utilize mouse clicks for calibration [41]. These recent calibration techniques are based on the assumption that the user's gaze will follow their mouse clicks which can then be used as ground truth for calibration.

We augment the gaze study with an auxiliary study using a custom gaze tracker with 23 users. The study set up is similar to above, except we use a state-of-the-art *Eye Tracking Glasses 2 Wireless* gaze tracker manufactured by SMI [48]. The gaze tracker uses a custom eyeglass, tracks gaze at 120 frames/sec, and has a very high accuracy ($\approx 0.5°$ is typical).

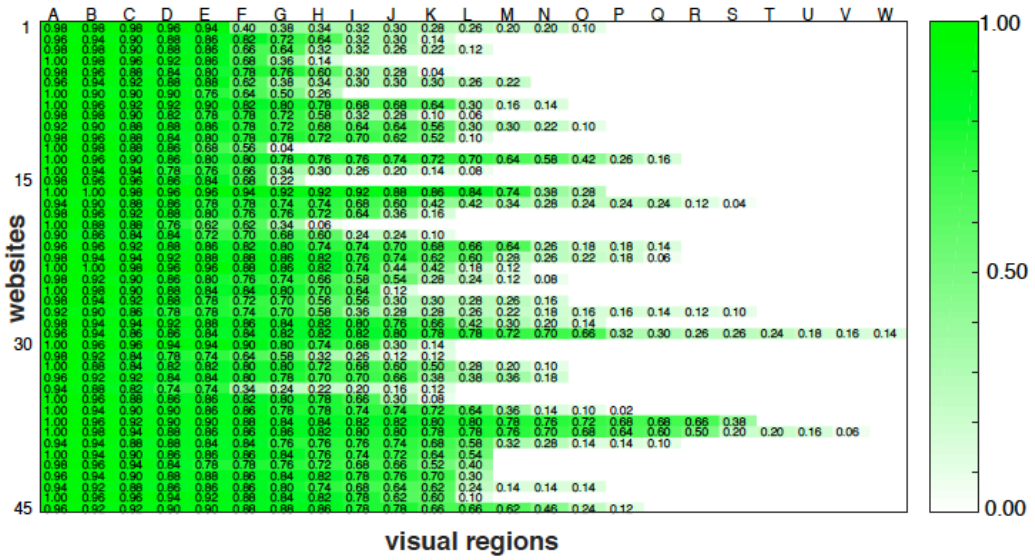

Figure 6: A heatmap of the collective fixation of Web page visual regions. Rows correspond to Web pages and the columns correspond to visual regions. For example, for Web site 1, visual region "A" has a collective fixation of 0.98 which means 98% of the users fixated on region "A" during gaze tracking.

4.3 Gaze Tracking Methodology

When a human views a visual medium, his/her eyes exhibit quick jerky movements known as *saccades* interspersed with relatively long ($\approx$ .5 second) stops known as *fixations* that define the his/her interest [33].

Web pages are designed for visual interaction, and thus contain many visually distinct elements, or *visual regions* [12], such as headers, footers, and main content sections, that help guide a user when viewing the page. Rather than track each fixation point, we segment a Web pages into its set of visual regions and track only the regions associated with the user's fixation points [24]. Figure 5 shows an example segmentation of `fcc.gov` into its visual regions. It is from this representation that we estimate the collective fixation of a visual region as the fraction of all users' gaze tracks that contain a fixation on the visual region. As part of future work, we will explore other signals of a user's gaze, including fixation duration and fixation order.

4.4 Collective Fixation Results

Figure 6 shows the collective fixation across each visual region of each Web page. The rows correspond to the Web page and the columns correspond to the visual regions in the Web page labeled 'A', 'B', etc (see example in Figure 5). Note that different Web pages may have different visual regions, since region creation depends on the overall page structure.

Figure 6 shows that for the first Web page, 5 regions have a collective fixation of over 0.9. In other words, 90% of the users fixated on these 5 regions in gaze tracking. But the remaining 75% of the regions have a collective fixation of less than 0.5.

In general, we find that across the Web pages, at least 20% of the regions have a collective fixation of 0.9 or more. We also find that on an average, 25% of the regions have a collective fixation of less than 0.3; i.e., 25% of the regions are viewed by less than 30% of the users.

Figure 7 shows the data in Figure 6 from a different perspective. Figure 7 is the median of the CCDF's of collective fixations for each site. Each point in the graph shows the percentage of regions with at least a certain collective fixation value. For example, the graph shows that 55% of the regions have a collective fixation of at least 0.7 in the median case. Our key takeaways are: *(i)* several regions have high collective fixation, and *(ii)* there is a significant number of regions that are relatively unimportant to the users. These points suggest that a subset of regions are relevant to the users' interests, an observation that can be exploited to improve uPLT (§6).

Figure 8 shows a visualization of the gaze tracks on `fcc.gov` across all users. The combined gaze fixations show a high degree of gaze overlap. The thicker lines show the regions on the Web page where the users' gaze exhibit a high degree of collective fixation. The thinner lines show the regions that only a few users look at.

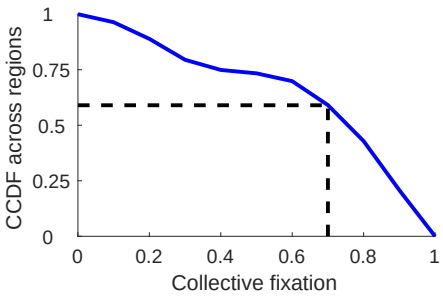

Figure 7: The median of the CCDF's of collective fixations across regions. Each point in the graph shows the fraction of regions with at least a certain collective fixation value in the median case.

Figure 8: A visualization of the gaze of all users when viewing `fcc.gov`. Certain regions on the page have more gaze fixations than others (as evidenced by the thicker lines).

4.5 Auxiliary Studies

In our auxiliary studies, we track gaze using a state-of-the-art gaze tracker as users viewed Web page loads under slow 3G and fast WiFi-like network conditions (network set up discussed in §3.1). The collective fixation results using the custom gaze tracker are quantitatively similar to the results when tracking gaze using the low cost tracker. For instance, 30% of the regions have a collective fixation of more than 0.8, and 30% of regions have a collective fixation of less than 0.1 under slow network conditions. The results under fast network conditions are similar.

We also conducted an additional set of experiments to study the effects personalized Web pages have on the user's gaze. Web pages such as `Facebook` customize their page to a given user, even though the overall structure of the page remains the same. This customization may result in different users focusing on different parts of the page. We choose five personalized Web pages where the users login to the site: Facebook, Amazon, YouTube, NYTimes, CNN. We conduct a user study with 20 users who gave us permission to track their gaze while they were browsing the logged-in Web pages. Despite customized content, we see similar patterns in collective fixation. All sites see a collective fixation of 0.8 or above for 30% of regions while still having at least 30% of regions with collective fixations below 0.1. In addition, on average these sites have 20% of their regions with a collective fixation above 0.9 and 33% below 0.3. Thus, even for pages where specific contents of the page vary across users, we observe there exist regions of high and low collective fixation.

5 WebGaze Design and Architecture

The previous section establishes that for each Web page, there exists several regions with high collective fixation. WebGaze is based on the intuition that prioritizing the loading of these regions can improve uPLT. This intuition is derived from existing systems and metrics, including Klotski [21] and the Speed Index. The goal of the Klotski system is to maximize the number of objected rendered within 3–5 seconds, with the intuition that loading more objects earlier improves user experience. Similarly, Speed Index uses the visual loading progress of a page as a proxy for the user's perception. The Speed Index value improves when more objects are rendered earlier on the screen. Similar to these works, our goal is also to render more objects earlier, but WebGaze chooses objects that are more important to the users as determined by their gaze feedback.

5.1 Architecture

Figure 9 shows the architecture of WebGaze. WebGaze is designed: *(i)* to have no expectations that all users will provide gaze feedback, *(ii)* to not require that pages be optimized based on real time gaze feedback. We note that existing gaze approaches for video optimization do rely on real time gaze feedback for prioritization [43]. However, Web page loads are transient; the short time scales makes it infeasible to optimize the Web page based on real time gaze feedback.

The WebGaze architecture collects gaze feedback from a subset of users as they perform the browsing task. WebGaze collects the gaze feedback at the granularity of visual regions. When sufficient gaze feedback is collected, the WebGaze server estimates the collective fixation across regions. The server periodically collects more gaze information and updates its fixation estimations as the nature of the Web page and users' interests change.

Based on the collective fixation values, WebGaze, (1) identifies the objects in regions of high collective fixation, (2) extracts the dependencies for the identified objects, (3) uses HTTP/2 Server Push to prioritize the identified objects along with the objects that depend on them. Below, we describe these steps in detail.

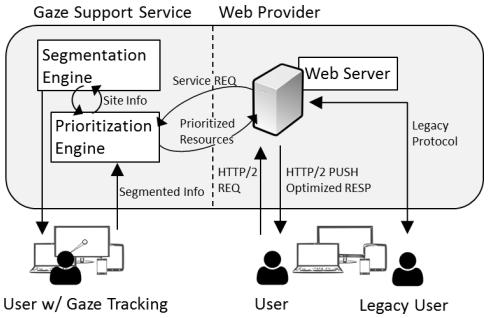

Figure 9: WebGaze architecture.

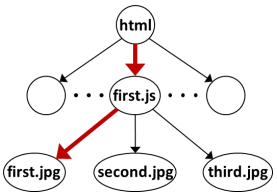

Figure 10: A dependency graph for an example page. If the `first.jpg` needs to be prioritized based on the collective fixation measure, then `first.js` also needs to be prioritized since `first.jpg` depends on it.

5.2 Identifying Objects to Prioritize

To identify which Web objects to prioritize, we use a simple heuristic: if a region has a collective fixation of over a *prioritization threshold*, then the objects in the region will be prioritized. In our evaluation, we set the prioritization threshold to be 0.7, thus any objects within a visual region that has a collective fixation of 0.7 or higher are prioritized. Recall from Figure 7 that this value identifies 55% of regions as candidates for prioritization in the median case.

Moving this threshold in either direction incurs different trade-offs. When the prioritization threshold is increased (moving right in Figure 7) we become more conservative in identifying objects to prioritize. However, in being more conservative we may miss prioritizing regions of which are important to some significant minority of users, which can in-turn negatively affect the aggregate uPLT. When the prioritization threshold is decreased, more regions are prioritized. The problem is that prioritizing too many objects leads to data contention for bandwidth that in turn affects uPLT [14] (in §6 we show the effect of prioritizing too many objects.) Empirically, we find that the prioritization threshold we chose works well in most cases (§6), through it can be further tuned.

Since each region may have multiple objects, WebGaze extracts the objects that are embedded within a region. To do this, we query the Document Object Model (DOM) [4], which is an intermediate representation of the Web page created by the browser. From the DOM we obtain the CSS bounding rectangles for all objects visible in the 1920x1080 viewport. An object is said to be in a given region if its bounding rectangle is within the region. If an object is said to belong to multiple regions, we assign the maximum of the collective fixation of the regions to the object.

5.3 Extracting Dependent Objects

Web page objects are often dependent on each other and these dependencies dictate the order in which the objects are processed. Figure 10 shows an example dependency graph. If `first.jpg` belongs to a region with high collective fixation and is considered for prioritization, then `first.js` also needs to be prioritized, because `first.jpg` depends on `first.js`. If not, then prioritizing `first.jpg` is not likely to be useful since the browser needs to fetch and process `first.js` before processing the image.

Our system identifies dependencies of each object to be prioritized, and considers these dependent objects for prioritization as well. Our current implementation uses WProf [60] to extract dependencies, but other dependency tools [21, 39] can also be used. While the contents of sites are dynamic, the dependency information has shown to be temporally stable [21, 39]. Thus, dependencies can be gathered offline.

5.4 Server Push and Object Sizes

WebGaze, like other prioritization strategies [21], uses HTTP/2's Server Push functionality to implement the prioritization. Server Push decouples the traditional browser architecture in which Web objects are fetched in the order in which the browser parses the page. Instead, Server Push allows the server to preemptively push objects to the browser, even when the browser did not explicitly request these objects. Server Push helps *(i)* by avoiding a round trip required to fetch an object, *(ii)* by breaking dependencies between client side parsing and network fetching [39], and *(iii)* by better leveraging HTTP/2's multiplexing [14].

Of course, Server Push is still an experimental technique and is not without problems. Google's studies find that using Server Push can, in some cases, result in a reordering of critical resources that leads to pathologically long delays [14]. To avoid such pathological cases, we check for a simple condition: if the FirstPaint of the page loaded with WebGaze takes longer than the LastVisualChange in the default case, we revert back to the default case without optimization (recall the definitions of FirstPaint and LastVisualChange from §3.1). In our evaluation, we found that for 2 out of the 45 pages, WebGaze's use of Server Push resulted in such delays.

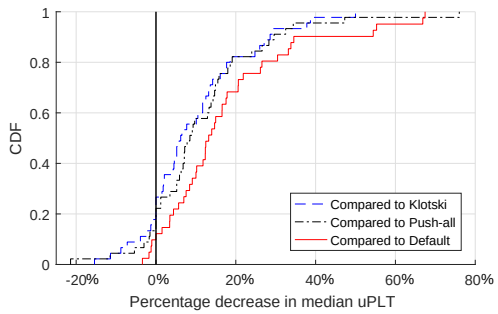

Figure 11: CDF of improvement in uPLT over Default, Push-All, and Klotski across the 100 users and 45 Web pages.

Another problem is that Server Push can incur performance penalties when used without restraint. Pushing too many objects splits the bandwidth among the objects, potentially delaying critical content, and in-turn, worsening performance. To address this, Klotski avoids prioritizing large objects or objects with large dependency chains [21]. Although we do not consider object sizes in our current implementation, we plan to do so as part of future work.

Finally, Server Push can use *exclusive* priorities [13] to further specify the order in which the prioritized objects are pushed as to respect page dependencies. However, existing HTTP/2 implementations do not support fully this feature. With a full implementation of HTTP/2's exclusive priorities, WebGaze's mechanism can potentially be tuned even further.

6 WebGaze Evaluation

We conduct user studies to evaluate WebGaze and compare its performance with three alternative techniques which are:

- *Default*: The page loads *as-is*, without prioritization
- *Push-All*: Push all the objects on the Web page using Server Push. This strategy helps us study the effect of Server Push at its extreme.
- *Klotski*: Uses Klotski's [21] algorithm to push objects. The algorithm pushes objects and dependencies with the objective of maximizing the amount of ATF content that can delivered within 5 seconds.

As before (§3.1), we record videos of the Web page as it is loaded using WebGaze and each alternate technique. The users provide feedback on when they perceive the page to be loaded as they watch the video. We conduct the user study across 100 users to compare WebGaze and each alternative technique. Videos of the Web page loads, under each technique, are available on our project Web page, `http://gaze.cs.stonybrook.edu`.

6.1 Methodology

Web pages: We evaluate over the same set of 45 Webpages as our uPLT and gaze studies (§3.1). Recall, from the WebGaze architecture, that the Web server corresponding to each Web page prioritizes content based on input from WebGaze. For evaluation purposes, we run our own Web server instead and download the contents of each site locally. We assume that all cross-origin content is available in one server. We note that HTTP/2 best practices suggest that sites should be designed such that as much content as possible is delivered from one server [14]. Nondeterministic dynamic content, such as ads, are still loaded from the remote servers.

Server and client: The Web pages are hosted on an Ubunbu 14.04 server running version 2.4.23 of Apache `httpd` which supports HTTP/2 protocol and Server Push functionality. The Web client is Chrome version 52.0.2743.116, which supports both the HTTP/2 protocol and Server Push, that is also run on an Ubuntu 14.04 machine. Traffic on the client machine is controlled using `tc` [18] to replicate standard DSL conditions (§3.1). When using push, we use default HTTP/2 priorities. Due to the standardized conditions of our network, the average variance in OnLoad is less than 1%. So we are able to compare the uPLT values across different page loads.

User study: We conduct pairwise comparisons of uPLT. To this end, we show the users randomized Web page loads that are loaded using WebGaze and using one of the three alternatives. The users provide feedback on uPLT. For each set of 45 comparisons, we recruit 100 users, for a total of 300 users. An alternative design would be to conduct a user study where a single user provides feedback for Web page loads under all four alternatives; but this requires users to give feedback on 180 Web pages which becomes tedious.

6.2 Comparing WebGaze with Alternatives

Figure 11 shows the CDF of the percentage improvement in uPLT compared to alternatives. On an average, WebGaze improves uPLT 17%, 12% and 9% over Default, Push-All, and Klotski respectively. At the 95th percentile, WebGaze improves uPLT by 64%, 44%, and 39% compared to Default, Push-All, and Klotski respectively. In terms of absolute improvement, when WebGaze improves uPLT the improvement is by an average of 2 seconds over Default and Push-All, and by an average of 1.5 seconds over Klotski. At the 90th percentile, WebGaze improves uPLT by over 4 seconds.

In about 10% of the cases WebGaze does worse than Default and Push All in terms of uPLT and in about 17% of the cases, WebGaze performs worse than Klotski. Of these cases where the competing strategies outperform

Alternative	# WebGaze better	# WebGaze same	# WebGaze worse
Default	37	4	4
Push-All	35	4	6
Klotski	33	4	8

Table 1: Number of Web pages for which WebGaze performs better, same, and worse, in terms of uPLT in the median case compared to the alternatives.

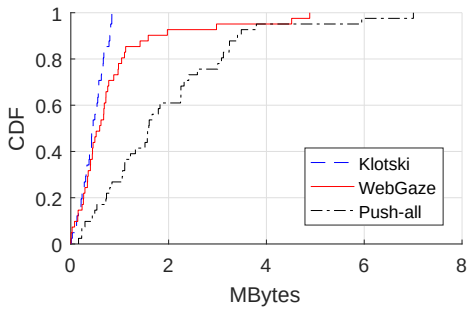

Figure 12: The total size of pushed objects under WebGaze, Klotski, and Push-All.

WebGaze, the average reduction in performance is 13%.

Table 1 shows the number of Web pages for which WebGaze performs better, the same, and worse in terms of uPLT for the median case, as compared to the alternatives. Next we analyze the reasons for the observed performance differences.

6.3 Case Study: When WebGaze Performs Better

It is not surprising that WebGaze improves uPLT over the default case. Recall our intuition based on prior work [21, 32] that prioritizing regions with high collective fixation can improve uPLT. In addition, pushing objects with adherence to their dependencies has been shown to improve page load performance [39, 60].

Push-All is an extreme strategy, but it lets us study the possible negative effects of pushing too many objects. We find that Push-All delays critical object loads and users see long delays for even the First Paint [31]. In our study, Push-All increases First Paint by an average of 14% compared to WebGaze. Push-All, in-turn, tends to increase uPLT. The problem with pushing too many objects is that each object only gets a fraction of the available bandwidth, in spite of techniques such as HTTP/2 priorities [14].

Different from uPLT, for OnLoad, it is more critical that all objects are loaded even if objects critical to the user are delayed. We see this tendency in our results: the Push-All strategy in fact improves OnLoad for 11 of the 45 pages, whilst hurting uPLT. This example shows that optimizations can help OnLoad, but hurt uPLT.

The uPLT improvement compared to Klotski comes from content-aware prioritization. In the case of Klotski, ATF objects are pushed based on whether they will be delivered within 5 seconds. This may not correlate with the objects that the user is interested in. For example, the Webpage `www.nysparks.com`, Klotski prioritizes the logo.jpg image which is in a region of low collective fixation. This essentially delays other more critical resources that are observed by a large number of users.

6.4 Case Study: When WebGaze Performs Worse

WebGaze performs worse than Klotski in 17% of the cases with a median performance difference of 5.5% and a maximum difference of 15.4%. In each of these cases, we find that Klotski sends less data compared to WebGaze and is more conservative. Figure 12 shows the relative size of objects pushed by WebGaze and Klotski across the Web pages. This suggests that we need to perform more analysis on determining the right amount of data that can be pushed without affecting performance. Similarly, when compared to Default, WebGaze performs worse for 4 of the 45 Webpages. In each of these cases, WebGaze pushed too much data causing bandwidth contention.

In all cases when WebGaze performs worse compared to Push-All, we find that the Web pages were smaller, less than 1.2 MB. We speculate that pushing all objects for pages of small sizes does not force as much contention for bandwidth.

6.5 Case Study: When WebGaze Performs the Same

For a fraction of less than 10% of the pages we find that WebGaze performs similar to the other alternatives. For two of the Web pages, the uPLT values are similar across the four alternatives. In other words, Server Push did not change performance. This could be because the default page itself is well optimized. For the other two pages, WebGaze's Server Push resulted in pathologically delays, and therefore the pages were not optimized (§5.4).

Although they are not the metrics WebGaze intends to optimize, for completeness we briefly discuss the performance of WebGaze in terms of the OnLoad, Speed Index, and First Paint. In terms of all three metrics, WebGaze and Klotski show comparable performance. In comparison to Default and Push-All, WebGaze shows only 1–3% improvement in the OnLoad. WebGaze improves the

Speed Index metric by an average of 18% compared to the Push-All strategy. However, there is no difference in the average Speed Index measure between WebGaze and Default. Lastly, as discussed earlier, WebGaze improves the average First Paint metric by 14% compared to Push-All. However WebGaze does increase the time to First Paint by 19% on average compared to Default, thus improving uPLT despite increasing the First Paint overall. This result loops back to our intuition (§5) that loading more objects important to the user sooner is critical to uPLT.

7 Related Work

We discuss three related lines of research that are relevant to our work: Web performance, page load measurements, and modeling user attention for Web pages.

7.1 Improving Web Performance

Given the importance of Web performance, significant research effort has gone into improving Web page loads. These efforts include designing new network protocols [10, 52], new Web architectures [39, 47, 62], best practices [1], and tools to help developers write better Web pages [29]. However, most of these efforts target the traditional *OnLoad* metric.

More recently, systems such as Klotski [21] are targeting the user quality of experience rather than optimizing traditional PLT metrics. As discussed earlier, Klotski uses HTTP/2's Server Push functionality to push *high utility and visible* objects to the browser. WebGaze uses a similar prioritization technique, but prioritizes objects based on user interest. Our evaluations show that WebGaze improves uPLT compared to Klotski across 100 users (§6).

7.2 Page Load Measurements

The research community has reported on a broad spectrum of Web page user studies. On the one end, there are controlled user study experiments [53], where the researchers create specific tasks for the subjects to complete. However, to create repeatable user studies and to control the Web page load times, the authors create *fake* Web pages. On the other end, there are large scale, less controlled studies [20] that measure performance of hundreds of real Web pages. But these studies only measure objective metrics such as the OnLoad metric.

Around the same time as the design and development of WebGaze, researchers have developed a similar testbed called `eyeorg` to crowd-source Web user experience [54]. The eyeorg study also uses a user-perceived PLT metric to measure user experience, and records the Web pages to obtain standardized feedback from the users as to when they feel the page is loaded. Their methodology in obtaining feedback is slightly different from our study in that they allow the users to transition frame by frame before providing their uPLT. The eyeorg study finds high correlation between the OnLoad and uPLT metrics, similar to our findings in the WiFi-like environment. Different from the eyeorg study, we vary the network conditions when loading the page and show that the correlation results depend on the underlying network (§3). On slow networks, OnLoad and uPLT are poorly correlated, while in faster networks, OnLoad and uPLT are better correlated; the later corroborating more with the results of eyeorg. Going beyond crowd-sourcing uPLT feedback, our work also shows how uPLT can be improved by leveraging eye gaze.

7.3 Web Saliency

The computer vision community has widely studied how eye gaze data can be used as ground truth to build saliency models [28,59]. Saliency defines the features on the screen that attract more visual attention than others. Saliency models predict the user's fixation on different regions of the screen and can be used to capture user attention without requiring gaze data (beyond building the model). While most of the research in this space focuses on images [44, 64], researchers have also built saliency models for Web pages.

Buscher et al. [19] map the user's visual attention to the DOM elements on the page. Still and Masciocchi [50] build a saliency model and evaluate for the first ten (10) fixations by the user. Shen et al. [46] build a computational model to predict Web saliency using a multi-scale feature map, facial maps, and positional bias. Ersalan *et al.* [24] study the scan path when the user is browsing the Web. Others have looked at saliency models for Web search [45] and text [26, 58].

However, existing Web saliency techniques have relatively poor accuracy [19, 46]. This is because predicting fixations on Web pages is inherently different and more challenging compared to images: Web pages, unlike images, are a mix of text and figures. Web page loading is an iterative process where all objects are not rendered on the screen at the same time, and there is a strong prior when viewing familiar Web pages.

Our work is orthogonal to the research on Web saliency. WebGaze can leverage better Web saliency models to predict user interest. This will considerably reduce the amount of gaze data that needs to be collected, since it will only be used to provide ground truth. We believe that our findings on how gaze data can improve user-perceived page load times can potentially spur research on Web saliency.

8 Discussions

There are several technical issues that will need a close look before a gaze feedback-based Web optimization can be widely useful.

Mobile Devices: It is expected that more and more Web content will be consumed from mobiles. Mobile devices bring in two concerns. First, errors in gaze tracking may be exaggerated in mobiles as the screen could be too small, or the performance of gaze tracking on mobile could be too poor. Significant advances are being made on camera-based gaze tracking for mobile smartphone class devices [7]. But, accuracy is also as not critical to our approach as we require the gaze to be tracked at the granularity of large visual regions.

A second concern is that gaze tracking on mobile devices may consume additional amounts of energy [43]. This is due to the energy consumed in the imaging system and on image analysis in the CPU/GPU. While this can be a concern, a number of new developments are pushing for continuous vision applications on mobiles and very low power imaging sensors are emerging (see, e.g., [36]). Also, lower resolution tracking may still provide sufficient accuracy for our application, while reducing energy burden. Therefore, we expect that gaze tracking can be leveraged to improve uPLT in mobile devices.

Exploiting Saliency Models: Saliency models have been discussed in the previous section. A powerful approach could be to decrease reliance on actual gaze tracking, but rely instead on saliency models. In other words, inspecting Web pages via suitable modeling techniques could discover potential regions of user attention that could be a good proxy for gaze tracks. This approach is more scalable and would even apply to pages where gaze tracking data is not available. The challenge is that research on saliency models for Web pages is not yet mature. Our initial results show promise in leveraging gaze for improving uPLT; exploiting Web saliency models can significantly increase the deployability of our approach.

Systems Issues: There are a number of systems issues that need to be addressed to build a useful Web optimization based on gaze feedback. For example, a standardized gaze interface needs to be developed that integrates with the browser. The gaze support service (Figure 9) needs to adapt to changing nature of the Web contents and user interests. For example, a major event may suddenly change users' gaze behaviors on a news Web site even when the structure of the page remains the same.

Security and Privacy: There are additional security and privacy related concerns if gaze feedback is collected by Web sites or third party services. For example, it is certainly possible that gaze tracking could provide a significant piece of information needed to uniquely identify the user, even across devices. The use of eye tracking on the end-user's device exposes the user to hacks that could misuse the tracking data. Note that course-grained tracking information is sufficient for our task, but guaranteeing that only course-grain information is collected requires a hardened system design.

Gaze Tracking Methodology: Web page loads are a dynamic process. Therefore, collecting gaze data when the user looks only at the loaded Web page is not representative of the Web viewing experience. Instead, in this work, we collect gaze data *as* the page is being loaded. However, one problem is that, the gaze fixation is influenced by the Web object ordering. For instance, if objects that are important to the user are rendered later, a user may direct her gaze towards unimportant, but rendered objects. Our methodology partially alleviates the problem by capturing gaze only after the First Paint (§6) and even after OnLoad. As part of future work, we propose to track user gaze when the Web objects are loaded in different orders. By analyzing gaze under different object orderings, we hope to alleviate the problem of the Web page loading order influencing gaze tracks.

9 Conclusions

There has been a recent interest in making user experience the central issue in Web optimizations. Currently, user experience is divorced from Web page performance metrics. We systematically study the user-perceived page load time metric, or uPLT, to characterize user experience with respect to traditional metrics. We then make a case for using users' eye gaze as feedback to improve the uPLT. The core idea revolves around the hypothesis that Web pages exhibit high and low regions of collective interest, where a user may be interested in certain parts of the page and not interested in certain other parts. We design a system called WebGaze that exploits the regions of collective interest to improve uPLT. Our user study across 100 users and 45 Web pages shows that WebGaze improves uPLT compared to three alternate strategies for 73% of the Web pages.

Acknowledgements

We thank our shepherd Srinivasan Seshan and the anonymous reviewers for their invaluable feedback that greatly improved the presentation of this paper. We thank graduate students Sowmiya Narayan and Sruti Mandadapu for their help in bootsrapping the project and for their help with setting up the Web page videos. This work was supported by the National Science Foundation, through grants CNS-1551909 and CNS-1566260.

References

[1] 14 Rules for Faster-Loading Web Sites. http://stevesouders.com/hpws/rules.php.

[2] Above-the-fold time (AFT): Useful, but not yet a substitute for user-centric analysis. http://bit.ly/29MBmip.

[3] Alexa: a commercial web traffic data and analytics provider. http://www.alexa.com/.

[4] Document Object Model(DOM). https://www.w3.org/DOM/.

[5] Going Beyond OnLoad: Measuring Performance that matters. http://oreil.ly/2cpaUhV.

[6] Hick's law: On the rate of gain of information. https://en.wikipedia.org/wiki/Hick%27s_law.

[7] How to use eye tracking on samsung galaxy s7 and galaxy s7 edge. http://bit.ly/29LDiqj.

[8] Measuring the critical rendering path with Navigation Timing. https://developers.google.com/web/fundamentals/performance/critical-rendering-path/measure-crp?hl=en.

[9] Microworkers: Crowdworking/Crowdsourcing Platform. https://microworkers.com/.

[10] SPDY: An experimental protocol for a faster web. https://www.chromium.org/spdy/spdy-whitepaper.

[11] The onload property of the GlobalEventHandlers. https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onload.

[12] AKPINAR, M. E., AND YESILADA, Y. *Vision Based Page Segmentation Algorithm: Extended and Perceived Success*. Springer International Publishing, 2013, pp. 238–252.

[13] BELSHE M., PEON R., AND THOMPSON M. ED. Hypertext Transfer Protocol Version 2 (HTTP/2). https://tools.ietf.org/html/rfc7540.

[14] BERGAN T., PELCHAT S., B. M. Rules of thumb for HTTP/2 server push. http://bit.ly/2d4O1RN.

[15] BIXBY, J. Case study: The impact of HTML delay on mobile business metrics. http://bit.ly/2cbN22l, November 2011.

[16] BIXBY, J. 4 awesome slides showing how page speed correlates to business metrics at walmart.com. http://bit.ly/1jfACl2, February 2012.

[17] BOJKO, A. Using Eye Tracking to Compare Web Page Designs: A Case Study. In *Journal of Usability Studies* (2006), vol. 1, pp. 112–120.

[18] BROWN, M. A. Traffic control how to, overview of the capabilities and implementation of traffic control under linux. http://www.tldp.org/HOWTO/html_single/Traffic-Control-HOWTO/, 2006.

[19] BUSCHER, G., CUTRELL, E., AND MORRIS, M. R. What do you see when you're surfing?: Using eye tracking to predict salient regions of web pages. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '09, ACM, pp. 21–30.

[20] BUTKIEWICZ, M., MADHYASTHA, H. V., AND SEKAR, V. Understanding website complexity: Measurements, metrics, and implications. In *Proceedings on Internet Measurement Conference*, IMC '11, pp. 313–328.

[21] BUTKIEWICZ, M., WANG, D., WU, Z., MADHYASTHA, H. V., AND SEKAR, V. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI '15, USENIX Association, pp. 439–453.

[22] CHROME DEBUG TEAM. Chrome remote debugging. https://developer.chrome.com/devtools/docs/debugger-protocol.

[23] CIOTTI, G. 7 marketing lessons from eye-tracking studies. https://blog.kissmetrics.com/eye-tracking-studies/.

[24] ERASLAN, S., YESILADA, Y., AND HARPER, S. Eye tracking scanpath analysis techniques on web pages: A survey, evaluation and comparison. *Journal of Eye Movement Research vol. 9*, No. 1 (2015), 1–19.

[25] FERHAT, O., AND VILARIÑO, F. Low cost eye tracking: The current panorama. *Computational Intelligence and Neuroscience vol. 3*, No. 2 (2016), 1–14.

[26] GAO, R., UCHIDA, S., SHAHAB, A., SHAFAIT, F., AND FRINKEN, V. Visual saliency models for text detection in real world. *PloS ONE vol. 9*, No. 12 (2014), 1–20.

[27] GAZEPOINTER. Control mouse cursor position with your eyes via webcam. http://gazepointer.sourceforge.net/.

[28] GOFERMAN, S., ZELNIK-MANOR, L., AND TAL, A. Context-aware saliency detection. In *Transactions on Pattern Analysis and Machine Intelligence*, vol. 34 of *PAMI '12*, IEEE, pp. 1915–1926.

[29] GOOGLE DEVELOPERS. PageSpeed Tools: The PageSpeed tools analyze and optimize your site. https://developers.google.com/speed/pagespeed/.

[30] GOVINDAN, R. Modeling HTTP/2 speed from HTTP/1 traces. In *Proceedings of 17th International Conference on Passive and Active Measurement*, vol. 9631 of *PAM '16*, Springer, p. 233.

[31] GRIGORIK, I. Analyzing critical rendering path performance. http://bit.ly/1ORhrNj.

[32] IMMS, D. Speed index: Measuring page load time a different way. http://bit.ly/2dbuUpr, September 2014.

[33] JOHN FINDLAY AND ROBIN WALKER. Human saccadic eye movements. http://www.scholarpedia.org/article/Human_saccadic_eye_movements.

[34] KIT EATON. How one second could cost amazon 1.6 billion in sales. http://bit.ly/1Beu9Ah.

[35] KRAFKA, K., KHOSLA, A., KELLNHOFER, P., KANNAN, H., BHANDARKAR, S., MATUSIK, W., AND TORRALBA, A. Eye tracking for everyone. In *Conference on Computer Vision and Pattern Recognition*, CVPR '16, IEEE.

[36] LIKAMWA, R., PRIYANTHA, B., PHILIPOSE, M., ZHONG, L., AND BAHL, P. Energy characterization and optimization of image sensing toward continuous mobile vision. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, ACM, pp. 69–82.

[37] LU, F., SUGANO, Y., OKABE, T., AND SATO, Y. Inferring human gaze from appearance via adaptive linear regression. In *Proceedings of the International Conference on Computer Vision*, ICCV '11, IEEE, pp. 153 –160.

[38] MOZILLA DEVELOPER NETWORK. Remote debugging. https://developer.mozilla.org/en-US/docs/Tools/Remote_Debugging.

[39] NETRAVALI, R., GOYAL, A., MICKENS, J., AND BALAKRISHNAN, H. Polaris: Faster page loads using fine-grained dependency tracking. In *Proceedings of the 13th USENIX Conference on Networked Systems Design and Implementation*, NSDI '16, USENIX Association, pp. 123–136.

[40] O'CONNELL, C. Eyetracking and web site design. https://www.usability.gov/get-involved/blog/2010/03/eyetracking.html, March 2010.

[41] PAPOUTSAKI, A., SANGKLOY, P., LASKEY, J., DASKALOVA, N., HUANG, J., AND HAYS, J. Webgazer: Scalable webcam eye tracking using user interactions. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, IJCAI '16, AAAI, pp. 3839–3845.

[42] PETROVICH, M. Going beyond onload: Measuring performance that matters. http://oreil.ly/2cpaUhV, October 2013.

[43] RYOO, J., YUN, K., SAMARAS, D., DAS, S. R., AND ZELINSKY, G. J. Design and evaluation of a foveated video streaming service for commodity client devices. In *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, ACM, pp. 6:1–6:11.

[44] SHARMA, G., JURIE, F., AND SCHMID, C. Discriminative spatial saliency for image classification. In *Conference on Computer Vision and Pattern Recognition*, CVPR '12, IEEE, pp. 3506–3513.

[45] SHEN, C., HUANG, X., AND ZHAO, Q. Predicting eye fixations on webpage with an ensemble of early features and high-level representations from deep network. *IEEE Transactions on Multimedia vol. 17*, No. 11 (2015), 2084–2093.

[46] SHEN, C., AND ZHAO, Q. *Webpage Saliency*. Springer International Publishing, 2014, pp. 33–46.

[47] SIVAKUMAR, A., PUZHAVAKATH NARAYANAN, S., GOPALAKRISHNAN, V., LEE, S., RAO, S., AND SEN, S. Parcel: Proxy assisted browsing in cellular networks for energy and latency reduction. In *Proceedings of the 10th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '14, ACM, pp. 325–336.

[48] SMI Eye Tracking Glasses 2 Wireless. http://bit.ly/29YWaDa.

[49] SOUDERS, S. Moving beyond window.onload(). http://bit.ly/1VzsdME, May 20.

[50] STILL, J. D., AND MASCIOCCHI, C. M. A saliency model predicts fixations in web interfaces. In *5th International Workshop on Model Driven Development of Advanced User Interfaces*, MDDAUI '10, pp. 25–28.

[51] SUGANO, Y., MATSUSHITA, Y., AND SATO, Y. Learning-by-synthesis for appearance-based 3d gaze estimation. In *Conference on Computer Vision and Pattern Recognition*, CVPR '14, IEEE, pp. 1821–1828.

[52] THE CHROMIUM PROJECTS. QUIC, a multiplexed stream transport over UDP. `http://bit.ly/2cDBKig`.

[53] VARELA, M., SKORIN-KAPOV, L., MÄKI, T., AND HOSSFELD, T. QoE in the Web: A dance of design and performance. In *7th International Workshop on Quality of Multimedia Experience*, QoMEX '15, pp. 1–7.

[54] VARVELLO, M., BLACKBURN, J., NAYLOR, D., AND PAPAGIANNAKI, K. Eyeorg: A platform for crowdsourcing web quality of experience measurements. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, ACM, pp. 399–412.

[55] VICK, R. M., AND IKEHARA, C. S. Methodological issues of real time data acquisition from multiple sources of physiological data. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, HICSS '03, pp. 1–7.

[56] VIOLA, P., AND JONES, M. J. Robust real-time face detection. *International Journal of Computer Vision vol. 57*, No. 2 (2004), 137–154.

[57] Peripheral vision. `https://en.wikipedia.org/wiki/Peripheral_vision`.

[58] WANG, H.-C., AND POMPLUN, M. The attraction of visual attention to texts in real-world scenes. *Journal of Vision vol. 12*, issue 6 (2012), 26–26.

[59] WANG, P., WANG, J., ZENG, G., FENG, J., ZHA, H., AND LI, S. Salient object detection for searched web images via global saliency. In *Conference on Computer Vision and Pattern Recognition*, CVPR '12, IEEE, pp. 3194–3201.

[60] WANG, X. S., BALASUBRAMANIAN, A., KRISHNAMURTHY, A., AND WETHERALL, D. Demystifying page load performance with WProf. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, NSDI '13, USENIX Association, pp. 473–486.

[61] WANG, X. S., BALASUBRAMANIAN, A., KRISHNAMURTHY, A., AND WETHERALL, D. How speedy is SPDY? In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI '14, USENIX Association, pp. 387–399.

[62] WANG, X. S., KRISHNAMURTHY, A., AND WETHERALL, D. Speeding up web page loads with shandian. In *Proceedings of the 13th USENIX Conference on Networked Systems Design and Implementation*, NSDI '16, USENIX Association, pp. 109–122.

[63] WebPagetest: website performance testing service. `http://www.webpagetest.org/`.

[64] YANG, J., AND YANG, M.-H. Top-down visual saliency via joint CRF and dictionary learning. In *Conference on Computer Vision and Pattern Recognition*, CVPR '12, IEEE, pp. 2296–2303.