The ODDness of Webpages

Marcin Furtak
Gianforte School of Computing
Montana State University
Bozeman, MT
marcin.furtak@interia.pl

Mike P. Wittie
Gianforte School of Computing
Montana State University
Bozeman, MT
mike.wittie@montana.edu

***Abstract*—Users and content providers want websites to load quickly. A widely used web performance metric that rewards both the early appearance of content and the timely completion of page load is the Speed Index (SI). Lower SI values correspond to higher user satisfaction, which makes reducing page SI an important goal.**

In this paper, we observe that all images on a webpage are not created equal and indeed vary considerably along a metric we dub *image density*, or the ratio of byte weight to pixel size. Variation in image density creates opportunities to prioritize lower density images to reduce page SI by displaying more pixels sooner for every loaded byte. We define Object Density Distribution (ODD) – a new webpage characterization metric. To understand the potential for image prioritization, we characterize ODD of existing webpages, their *ODDness* if you will, and show that ODD skewness and kurtosis indicate meaningful prioritization opportunities. To understand the effectiveness of image prioritization, we propose a URL-based prioritization mechanism and measure its performance across 20 test pages loaded from the Apache, NGINX, and Caddy servers. Our results show SI improvement over 40% in some cases and mean improvement of 5.7%. These SI improvements and the simplicity of our prioritization method create a compelling case for the adoption of our method by content distribution networks (CDNs) and future browser implementations.

***Index Terms*—web performance, speed index, content delivery**

I. Introduction

Users and content providers want their websites to load and display quickly. Static resources, such as HTML, JavaScript (JS), Cascading Style Sheet (CSS), and image files, needed to render website views load from content delivery network (CDN) servers. As a result, user experience, quantified by web performance metrics such as page load times (PLT) and Speed Index (SI), depends on the fast acquisition of hosted resources.

Although research on web performance has helped to make a more efficient use of available network resources, for example by prioritizing CSS and JS objects and eliminating Head of Line (HoL) blocking [1], network delay remains difficult to mask. One persistent challenge is the delay of loading images, which depends on transport layer throughput, and so both bandwidth and latency. The introduction of web metrics such as SI showed that the order of object appearance during a page load matters to user experience, which led to prioritization of above-the-fold (ATF) objects to accelerate their display, even though page load time (PLT) remains unchanged [2].

Currently, browsers request images in order of their embedding in the base HTML, or in the order of their placement on the page. This top to bottom order makes sense in that it leads the browser to request ATF images first. With the deployment of HTTP/2 implementations in browsers and servers there is potential to prioritize images with respect to each other to force a strict priority of ATF images. Chrome does just that and initially loads images with the `Net:Lowest` priority, but upgrades to `Net:Medium` priority those images that are discovered to be in the viewport, or ATF [3].

In this paper, we conjecture that browsers may leverage HTTP/2 prioritization to improve page SI scores, not just ATF time. We make the observation that all images are not created equal. For example, two images of same display dimensions (number of pixels) may take up different numbers of bytes on disk. As a result, the image with the lower byte to pixel ratio, a metric we dub *image density*, transfer more quickly and display earlier to improve page SI and user experience. That approach would eliminate head of line (HoL) blocking in the transport layer send buffer among images of different densities. The questions we address in this paper is whether and how the web performance community may leverage differences in image density to improve user experience with existing websites. In other words, can browsers use HTTP/2 stream prioritization to remove HoL blocking among images of different densities to improve page SI.

We address the questions of the practical utility of image density variation on several fronts. First, we describe Object Density Distribution (ODD) as a new webpage characterization metric that approximates the SI improvement of image prioritization during a page load. Second, we characterize ODD of real webpages, their ODDness if you will. Our analysis of the 200 most popular Alexa pages (Alexa-200) shows considerable ODD skewness and kurtosis, which indicates there exist meaninful opportunities for image prioritization. Finally, third, we develop a URL-based method and a Go client proxy to prioritize HTTP/2 streams based on the density of images they carry. To evaluate our method we measure the SI of 20 test pages composed of images from representative Alexa-300 pages loaded from Apache, NGINX, and Caddy servers. Finally, we present a machine learning model to identify rare page/network condition scenarios where image prioritization should not be used.

In sum, this paper offers the following contributions:

- We show the benefit of prioritizing images based on their density. Specifically, our results show SI improvement over 40% in some cases and a mean improvement of 5.7% across a variety of web pages and network conditions. Our work dovetails earlier results from the web performance community on the importance of prioritization of CSS and JS objects and on the correlation of total image weight on SI [4]. Our results represent an upper bound of SI improvement – to fully realize them, developers should avoid hierarchical page structures so as to avoid blocking of image loads by JS execution and DNS lookups.
- We present a URL-based technique for image prioritization and describe how it may be safely and incrementally deployed by CDNs and browsers.
- We describe the implementation of a Go proxy with a forwarding delay of less than 4 ms, which demonstrates the effectiveness of image prioritization even before the method becomes adopted by browsers.

We organize the rest of this paper as follows. In Section II we introduce ODD and characterize the ODD of existing web pages. Section III details the implementation of our proxy and method for image load prioritization. We measure the benefits of image prioritization in Section IV. Section V outlines related work on web performance metrics and HoL blocking. Finally, we discuss options for proxy deployment and conclude in Section VI.

II. Object Density Distribution (ODD)

Our guiding idea is that not all images are created equal. We begin by explaining the concept of image density. We then show the degree to which image density varies among the images of existing web pages to understand the opportunities for image prioritization.

A. Image Density

We define image density as the ratio of image weight in bytes to image size in rendered pixels on a webpage. Thus, a 614 KB image might take up 290×162 pixels at density of 0.0131. Note, that the pixel size of an image is its display size defined in HTML, as absolute pixel dimensions, or image scale factor, rather than the dimensions of the image file itself. Browser's Document Object Model (DOM) provides that information as the *client* image size. Using this definition, we calculate image density for every image, visible above the fold and dub this distribution *client ODD*.

While client ODD represents the ground truth of rendered images sizes, there are several disadvantages to that metric. First, the calculation of client ODD requires the analysis of the HTML structure of a page. Full page HTML may not be available until all JS resources have been loaded and executed, which in many cases does not happen until after the browser starts requesting the embedded images. Second, the same image may be embedded at different size in different pages, which makes it difficult for the server to know its density on any given page – a property that we will show in Section III-A is helpful in implementing image prioritization.

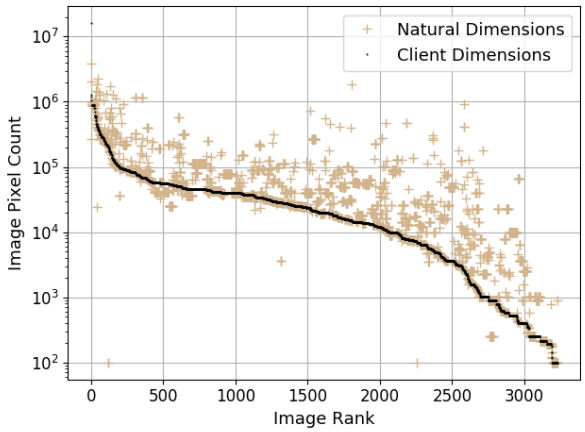

Fig. 1. Client and Natural pixel count for images in Alexa-200 pages.

To address these shortcomings of client ODD we make the observation that in well-optimized webpages the absolute image size does not significantly differ from the its size defined in the HTML [5], [6]. This correspondence makes sense, because embedding large images, but displaying them scaled down wastes network resources. Similarly, embedding small images, but scaling them up reduces their visual quality. The DOM provides the size of an image before scaling as the *natural* image size.

To quantify the relationship between client and natural image sizes we measure the images on the Alexa-200 pages. We use the Chromium browser to load each web page, extract the list of all objects with the '`img`' tag and obtain its client and natural dimensions. We ignore the pages that do not allow scraping.

Figure 1 shows the client and natural densities of the Alexa-200 images. The x-axis shows image rank sorted by size, while the y-axis shows image client size across all the pages. We observe that the client and natural size distribution are moderately close with a correlation of 0.45. For most of the images the natural image size is greater than the client size. We also observe that webpages scale down 7% of images by over 1000%. We call such images *black hole images* because their level of rescaling results in very high densities.

These results show that while natural density does to some extent approximate client density, the two cannot be used interchangeably. Thus an accurate ODD calculation does currently require HTML analysis for most pages. We believe it may be possible to eliminate that requirement by encoding images such that the natural and client densities are the same. In well-optimized pages natural and client image sizes should be made to match [5], [6]. Indeed we found that is already the case for about 30% of Alexa-200 images. We also observed that most of these well-optimized pages are in the more popular set of Alexa-100 pages.

B. ODD of Existing Webpages

When a page contains images of varying density, loading the less dense images first results in more pixels displayed after transferring same number of bytes. On the other hand,

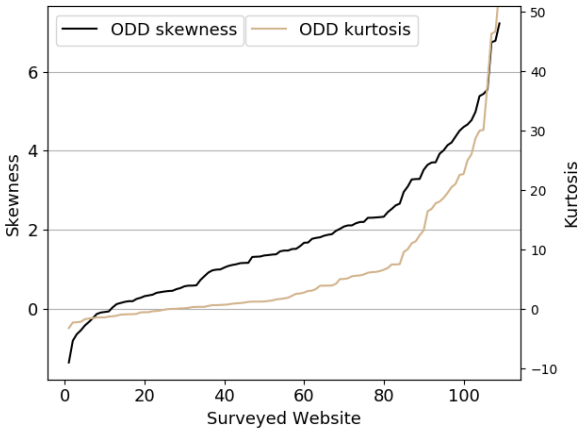

Fig. 2. ODD skewness and kurtosis of Alexa-200 pages.

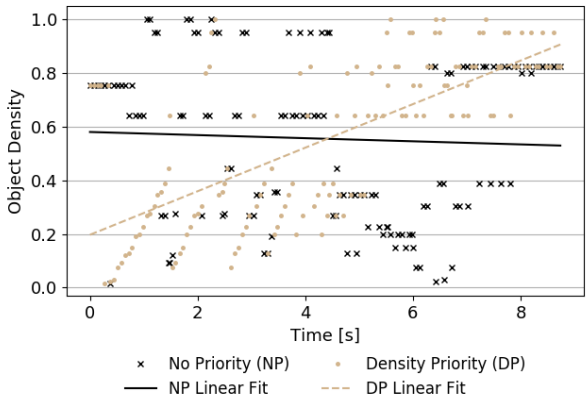

Fig. 3. Density of delivered bytes during page load.

on pages where the density of all images is the same there is no advantage to prioritization, because regardless of which image is loaded first the browser displays the same number of pixels per loaded byte.

To understand the opportunity for image prioritization we want to understand the variation of image density in the client ODD (hearafter just ODD) of existing webpages. To characterize ODD we compute its skewness (distribution asymmetry about its mean) and kurtosis (distribution "tailedness"). Both skewness and kurtosis are statistical measures of population differences. In our case, they signify the degree to which image density varies within a page and creates opportunities for image prioritization. In Section IV-C we find that indeed both ODD skewness and kurtosis predict SI speedup from image prioritization with a high accuracy.

Figure 2 shows the skewness and kurtosis of the Alexa-200 pages. The x-axis shows the pages sorted by the skewness/kurtosis metric, which is marked on the left and right y-axes respectively. For 50 of the pages the standard deviation of ODD is zero or very small, which causes divide by zero errors in floating point calculations of skewness and kurtosis – we remove those pages from the graph. Additional 39 pages did not permit crawling. We observe that most of the remaining pages have defined and non-zero values of ODD skewness and kurtosis, which indicate variation in image densities and opportunities for image prioritization.

C. Image Prioritization in Existing Browsers

Browsers already prioritize the loading of JS and CSS files [1]. We want to understand whether the browser does something similar under the hood for images of different density. To do so, we conduct the following experiment. We load a page containing 24 images of various densities and associate each transferred HTTP/2 frame with an image and record the density of the contained image, as well as the arrival time of the frame. In Figure 3 we plot the relative density (normalized to the densest image on the page) of delivered frames on the y-axis versus their arrival time on the x-axis. The black 'x' markers show the density of loaded bytes using the 71.0.3578.98 version of the Chromium browser from the Caddy server.

A page load that prioritizes objects by their density should show low density bytes transferred first. Figure 3 shows the opposite, where the density of loaded bytes is unordered over time. In fact the solid black trend line shows that the average density of loaded bytes actually decreases over time as the result of random load order with respect to image density. Our observations are consistent with a recent measurement study, which showed that browsers assign the same priority to images and load them in parallel [7]. A recent work by Wijnants *et al.* offers a more detailed treatment of browser implementation of HTTP/2 priorities [8].

For comparison we also plot (gold round markers and a dashed trend line) a download that prioritizes objects by their density, using the method described in the following section. Our approach tends to load low density images first although due to server behavior the prioritization is not strict. We come back to this result in Section IV-B after detailing the implementation of our method and quantifying the benefit of prioritization on page SI.

III. MITM Proxy

To ascertain the potential of image prioritization to improve SI at a large scale we design a collaborative mechanism between clients and content providers, or CDNs. Due to the onerous nature of browser modifications we choose to implement our mechanism on an HTTP/2 man-in-the-middle (MITM) proxy deployed on a client machine in parallel with the browser.

A. ODD and Stream Prioritization

To prioritize a website's images the proxy needs to know their densities. The requisite information, image byte weight and pixel size, is not available until page HTML and the images themselves finish loading. To avoid the delay on the browser, we propose that the content providers, or CDNs, analyze the HTML of hosted pages and embed image density in the object URL. Thus `pic.png` of weight of 614 KB and size 290×162 pixels embedded in the base HTML would

become `pic_0.0131.png`. However, as we explain later, the proxy needs to know density relative to other images on a page, such that the densest image receives normalized of 1 and other objects densities in the range $(0, 1]$. We assume that while analyzing page HTML the CDN could normalize image densities and convert absolute density of 0.0131 to, for example, 0.65 and save the image as `pic_0.65.png`. The MITM proxy forwarding `HEADERS` frames from the browser scans embedded image URLs for their densities and uses them to assign priorities to HTTP/2 streams originated by the browser to request embedded content. We note that for two images with the same density, even if they have different pixel sizes, there is no SI difference between loading one image first and loading them in parallel. Injecting image density metadata into image URLs manages to avoid the need for browser or server modifications thus making our method easy to deploy incrementally.

A potential disadvantage of this method is that the same image may embedded in two different pages and have different normalized densities with respect to other images on each page. As a result, it might have to be hosted twice as, for example, `pic_0.65.png` and `pic_0.55.png`. While hosting multiple versions of the image is relatively inexpensive, it is important to prevent the browser from loading the image twice and instead having it use a cached version. To allow browsers to use cached versions of an image the CDN could replace the name of the image with an ID unique within a page, for example, `0376_0.65.png`. The browser could then serve the image from the cache based on the `0376` prefix and the `.png` extension matching the existing cache mechanisms.

When a request for a new image passes through the proxy the proxy computes its stream priority as

```
strm_prio = (1 - normalized_density) *
              h2_max_prio
```

to assign less dense images higher priorities. Additionally, the proxy should make sure that the new priority is lower than the priority reserved for the CSS and JS objects (`h2_max_prio`) and stays between medium and low stream priorities [3]. In this way no image download blocks these high priority objects.

For the purposes of this paper we implement the priority mechanisms in the proxy, but the method could be implemented more efficiently in the browser itself. The browser already analyzes the page HTML and so it knows the client size for each image. If CDNs embedded the byte size in each image URL as, for example, `pic_614.png`, the browser would be able to calculate the client density for each image, normalize it, and use it to assign stream priorities. While the proxy could do so as well, the overhead of processing HTML on the proxy would add additional delay to its forwarding function. This mechanism would still enable browsers to leverage their cache by matching the `pic` prefix and the `.png` extension. If the same image were embedded at different sizes on two different pages, the browser could use the image from its cache as long as it was the larger of the two.

B. Proxy Implementation

We decided to implement our MITM proxy in Go after early experiments with a popular Python proxy added unacceptable forwarding delays. The Go proxy exposes an HTTP/2 server towards the browser and uses standard HTTP/2 client to connect with the web server. To achieve low latency and high throughput the synchronization and data exchange between the said proxy parts and its main logic loop is asynchronous using Go's channels and routines. Throughout its operation the proxy maintains two independent TCP connections and two independent HTTP/2 connections: one on its client side and one on its server side. The proxy operates as follows:

1) Connect into the lower levels of Go's HTTP/2 server to access HTTP/2 frames as they arrive from the browser.
2) Intercept `HEADERS` frames containing browser requests.
3) For each request, extract relative image density (normalized to the website's densest object) from request URI, calculate stream priority based on image density, and create a request for the server with the updated priority.
4) Forward object streams from server to client according to priorities from Step 3.

IV. Evaluation

To understand the effectiveness of image prioritization on reducing SI we conduct a measurement of 20 webpages loaded from three server implementations under a range of network conditions. Our results show that most pages in that set benefit from image prioritization performed by our proxy implementation. We also show a predictive model to decide when images on a page may be safely prioritized.

A. Experimental Setup

The benefit of image prioritization depends on how well servers support HTTP/2 stream prioritization. The question arises because RFC 7540 states that "expressing priority is (. . .) only a suggestion" for the server [9]. Indeed many CDN servers do not obey prioritization of ATF images [10]. To understand whether image prioritization is effective across server implementations we evaluate our method on the Apache, NGINX, and Caddy servers. The servers differ in how they implement HTTP/2. Apache uses `libnghttp2` library [11], NGINX uses its own implementation of the standard – the `ngx_http_v2_module` [12], and Caddy relies on Google's Go implementation of HTTP/2 (the same one we use in our proxy) [13].

To accurately measure the impact of image prioritization on sets of images contained in real pages our approach is to save offline the images embedded in a webpage and create a new page structure that displays them at as a grid. We include images embedded in the base page, but not banner ads, which in general load from servers not controlled by page developers. This method separates the impact of image load time from other factors, such as JS execution, DNS lookups, or blocking calls to third party servers. Indeed in some real world web pages we observe such blocking behavior, which nevertheless may be avoided with existing page optimization techniques

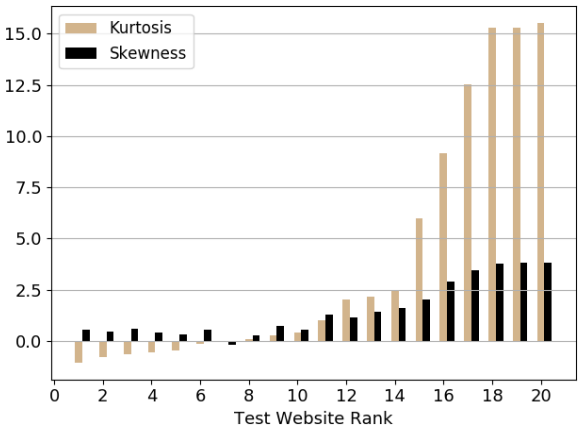

Fig. 4. ODD skewness and kurtosis of 20 test pages.

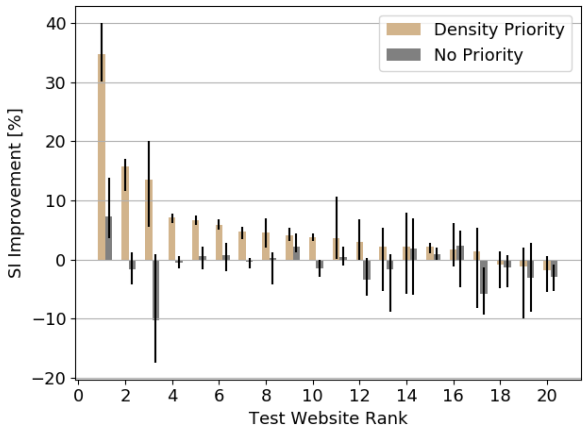

Fig. 5. Test page SI improvement for Caddy over 5 Mbps with 30ms RTT.

discussed further in Section V-A [1], [14]–[17]. As such, our measurement of page SI and improvement from prioritization represent an upper bound. At the same time, a recent analysis of HTTP Archive shows a high correlation between total image weight and SI, so our results indicate the impact of image prioritization on that aspect of web performance [4].

For our evaluation we create a set of 20 test pages with images originating from Alexa-300 websites to capture different representative page layouts. Based on pictures from real pages, we wanted to create a set of different pages that span possible page characteristics to understand their relationship with prediction and benefit. While measurement with real web pages would have been in some ways preferable, our method does require object renaming, which would require us to serve these pages from our servers and estop 'in the wild' realism anyway. Our test pages contain images from news service, video streaming, e-commerce, image gallery websites. We obtain image sizes by querying the DOM. The number of images on these pages ranges from 19 to 125, their total byte size from 128 KB to 5 MB, and their image area coverage from 19.2% to 78.7%. We show the ODD skewness and kurtosis of the test pages in Figure 4, which in shape are representative of the skewness and kurtosis of Alexa-200 pages in Figure 2.

Our evaluation environment consists of the following. We run the Chromium browser (71.0.3578.98) and proxy (Go version 1.10.3) on commodity Ubuntu 16.04 Dell OptiPlex 7040 with Intel Core i7-6700T CPU and 16GB of memory. We orchestrate our browser tests by accessing Chromium remotely from a Node.js script via Chrome DevTools Protocol [18] and the chrome-remote-interface client library (0.26.1) [19]. We measure SI by saving each page load history into Chromium's trace file and use the Speedline library (1.4.2) to interpret it [20]. The server machine is a Lubuntu 17.04 ThinkPad R61E Intel Core 2 Duo T7100 CPU with 4GB of memory. We use default TCP settings for both machines. Client and server machines connect over 100 Mbps Ethernet.

We conducted our experiments using three types of `netem` emulated networks conditions of 2, 5 and 10 Mbps with 10, 30, and 50 ms RTT and three popular web servers Caddy (0.11.1), Apache (2.4.25) with `libnghttp2` library (1.2.1.1) and NGINX (1.10.3). For the purpose of evaluation we conduct each measurement 40 times.

B. Image Prioritization and SI Speedup

We aim to understand the effectiveness of image prioritization by density to reduce page SI. For each test page we measure its SI in three download scenarios: without a proxy, through a proxy but without prioritization, and through a proxy with image prioritization. SI measurement through a proxy without prioritization allows us to distinguish any negative, or positive effects of the proxy apart from prioritization.

Figure 5 shows the SI improvement of pages loaded through our proxy with and without prioritization with respect to the SI of the no proxy scenario. We load the pages from the Caddy server with 5 Mbps bandwidth and 30 ms round trip time (RTT) controlled by `netem`. The x-axis shows pages in descending order of percentage SI improvement, while the y-axis shows the percentage of SI improvement. Each point represents the mean of the SI improvement over 40 runs with the error bars showing 95% confidence intervals.

In general we observe that the SI of most of the test pages improves under prioritization. In Figure 5 the maximum SI improvement is almost 40% with the mean improvement across all pages of 5.7%. In contrast using a proxy without prioritization does show a slight degradation in performance (SI increase) of 0.85%.

Although the prevailing wisdom is that extra bandwidth does not matter much [21], there is a correlation between image weight and SI that can only be accounted for by bandwidth [4]. To gain a fuller picture of the effect of image prioritization we conduct the experiments under other network conditions. Table I shows the mean SI improvement for Apache, NGINX, and Caddy under combinations of 10, 30, and 50 ms latency and bandwidth of 2, 5, and 10 Mbps. We observe that image prioritization helps in all cases except for Apache and Caddy servers under very performant networks with 10 ms latency and bandwidth of 10 Mbps. This adverse effect is due to the forwarding overhead of the proxy, around 4 ms, and may be eliminated if prioritization were to be implemented in the browser itself. We also observe large

RTT (ms)	Bandwidth (Mbps)	Mean SI Improvement (%)		
		Apache	NGNIX	Caddy
10	2	2.55	5.35	6.00
	5	1.55	2.85	3.35
	10	-1.2	0.95	-1.15
30	2	4.55	5.8	8.35
	5	3.0	4.6	5.7
	10	1.95	3.35	2.9
50	2	5.75	7.95	10.1
	5	5.65	8.3	9.0
	10	4.9	6.95	6.75

TABLE I
MEAN SI IMPROVEMENT ACROSS TEST SCENARIOS

benefits of image prioritization for experiments with 50 ms latency. While a portion of the improvement is due to the proxy splitting the TCP connect (the proxy without prioritization by itself reduces SI by 1-4%) the bulk of the benefit comes from prioritizing images.

C. Prediction of SI Speedup

Although image prioritization leads to SI improvement on average, we want to understand if it is possible to predict whether prioritization would reduce SI a given page. An accurate prediction model would enable our proxy to safely decide when to apply image prioritization. For example in Figure 5, prioritization should be applied to pages 1-17, but not 18-20.

We used our measurement data to predict the ratio of SI speedup from image prioritization (prioritization SI to no proxy SI) for each server across the different network conditions. SI ratio less than 1 indicates that prioritization helps, while ratio greater than 1 indicates that prioritization should not be used. We trained a linear regression model (`sklearn.linear_model.LinearRegression`) based on webpage characteristics (factors) that govern the benefit from image prioritization. These were: 1. the number of images on a webpage (num_imgs) – the more images, the more opportunities for prioritization; 2. the percent of a page covered by images (img_area_prcnt) – the greater the image coverage, the greater the impact of images on SI; 3. the total image size (total_img_bytes) – the more image bytes, the more time spent in image transmission; 4. webpage ODD characterized through its standard deviation (ODD_std_dev), variance (ODD_var), kurtosis (ODD_kurt), and skewness (ODD_skew). For each factor set we trained a model for all $|n-1|$ website subsets and predicted the SI ratio for the remaining site. We then computed the percent error of predicted to actual SI ratio for each server and set of network conditions.

Table II shows the results of the prediction model. For each factor set and server we show the means of percent error, SI ratio, and factor coefficients. We found that ODD_var was not a predictive factor (error greater than 100%) and so we omit the results from factor sets including the metric. We also found that total_img_bytes, while helpful in predicting SI, is not a significant factor in predicting speedup ratio (less than 1.2×10^{-8}) and so for simplicity we present results from factor sets that do not include it. We also considered other factor combinations, for example num_imgs, img_area_prcnt, ODD_kurt, ODD_skew, which did not show more accurate predictions in general.

The results show that linear regression predicts SI ratio with a high accuracy across websites and network conditions. While the results in Table II are means across network conditions, we observe that the prediction error for individual server, factor set, network condition scenarios is small enough to accurately predict whether the speedup ratio is positive, or negative, and so whether prioritization would be helpful.

We observe that ODD kurtosis is the most predictive ODD characterization. The predictive power of kurtosis makes sense, because kurtosis indicates whether there is a spread of densities among page images which benefits prioritization. Similarly ODD skewness is predictive, because high skewness indicates a long tail of low density images on a page, which likewise benefits prioritization. These results show a proxy with a trained model, knowledge of server type from the HTTP reply header, and ODD distribution computed from image URLs embedded in the HTML base page, could safely decide whether or not to apply image prioritization to reduce page SI. These results also justify the use of ODD kurtosis and skewness in Figure 2 to estimate the potential of image prioritization in existing webpages.

D. Server Behavior

Finally, we aim to understand server behavior when prioritizing HTTP/2 streams. Figure 6 shows the timeline of webpage objects loaded from Caddy with and without prioritization. In each scenario the browser requests all images as it parses the base HTML. The y-axis shows the image density (and relative priority) while the x-axis shows object load time.

Factors	Server	Error %	SI Ratio	Coefficients
num_imgs, img_area_prcnt, ODD_kurt	Caddy	6.68	0.9335	0.0006, 0.0009, 0.0020
	Nginx	4.70	0.9512	0.0006, 0.0007, 0.0039
	Apache	3.63	0.9653	0.0002, 0.0003, 0.0009
num_imgs, img_area_prcnt, ODD_skew	Caddy	6.79	0.9325	0.0007, 0.0010, 0.0078
	Nginx	4.81	0.9510	0.0006, 0.0008, 0.0161
	Apache	3.68	0.9647	0.0002, 0.0003, 0.0030
num_imgs, img_area_prcnt, ODD_std_dev	Caddy	7.13	0.9299	0.0007, 0.0013, 0.0503
	Nginx	5.62	0.9522	0.0006, 0.0009, 0.0234
	Apache	3.83	0.9635	0.0002, 0.0004, 0.0162

TABLE II
LINEAR REGRESSION PREDICTION OF SI SPEEDUP RATIO.

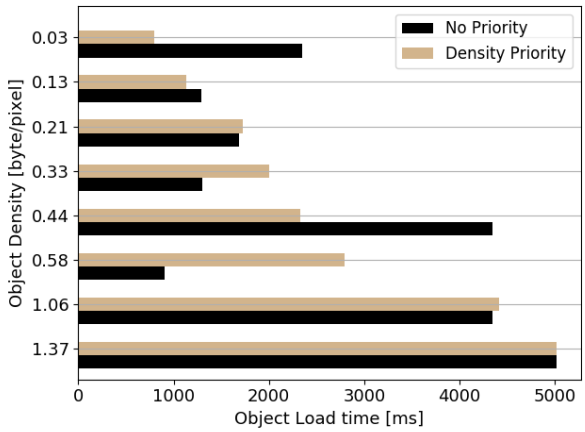

Fig. 6. Time line of a page load with and without density prioritization.

We observe that prioritizing lower density images allows them to finish loading earlier. In contrast, when the same images load at the same priority their load time depends on their size. This effect of prioritization holds regardless of the actual priorities given to streams, as long as the relative stream priority (priority order between streams) remains the same. We have seen this effect on Apache and NGINX servers as well. In other words, servers seem to implement strict priority and adjusting stream priorities in a way that does not affect their order has no effect on performance. Interestingly, strict priority seems to be in contrast with RFC 7540, which states that "Each dependency is assigned a relative weight, a number that is used to determine the relative proportion of available resources that are assigned to streams dependent on the same stream" [9].

We also observed that Apache and NGINX servers deviate from requested priorities by sending a few bytes of a lower priority (higher density) object first. Figure 3 shows this behavior where the first few bytes transferred under prioritization (gold round markers) have density around 0.75. Then after around 0.2 seconds the server starts sending frames containing a set of low density images in a round robin fashion. We suspect that the initial transmission of high density bytes might be due to races between threads that fetch objects from disk. As object requests arrive from the server, they trigger server threads that load images from disk and pass them to the HTTP/2 send buffer. It is possible that low priority stream bytes enter the HTTP/2 send buffer and pass to the send buffer of the TCP connection before bytes of a high priority stream displace them in the HTTP/2 send buffer. To avoid this HoL blocking in the TCP send buffer the server administrator may set the `TCP_NOTSENT_LOWAT` socket option to restrict the TCP send buffer to the minimum size required to fully utilize connection bandwidth [7]. Even if low priority bytes get in initially, once the buffer empties, higher priority objects may be transmitted by the server more quickly. Another option in Apache is to use `mod_file_cache` module to keep high priority (low density) objects in memory [22].

V. Related Work

To put our work in context we briefly discuss related work on web page optimization developers might use to reduce SI or other web performance metrics [2], [23].

A. Web page optimization

There are a number of well-understood methods for web page optimization [24], [25]. Browsers automatically prioritize the load of CSS files, such that they do not block the display of images when these load. Similarly, the load of JS files receives high priority, because they might point to additional HTML, CSS, or object files. Beyond that developers may optimize image byte size to its display size, use lossless compression to encode them, and leverage browser caching such that images embedded in multiple pages do not need to be reloaded. Further, developers may optimize a page to defer loading of objects not visible ATF and to defer parsing of JS scripts not on the critical path to first paint. After a page design is finalized a developer may also minimize HTML and JS to reduce script sizes. There are several tools on the market to identify these and other candidate optimization for a given page [5], [6], [26].

Recent browsers also provide facilities to order the loading and display of image objects. For example the `preload` HTML tag directs a browser to load certain images first [1]. This mechanism may be augmented by asking the browser to defer the load of other images until they come into view, for example when a user scrolls down the page [27]. Combined, these mechanisms allow a user to specify three priority levels even in HTTP/1.1. For more control, Chrome allows developers to order image load and display using the recently released image decoding API [14]. The intent of the API is to allow images to load asynchronously then display when loaded, thus masking the loading time from the main thread. The API can block the loading of certain images, such as advertisements or high density images in our case, until the hero images or low density images have been displayed. Finally, there is recent work on ordering the preloading of media content such as video, audio, and track [15].

The research community has also proposed several solutions for automatic web page optimization. Shandian pre-processes page structure on a proxy and then orders network transfers and computation to reduce PLT [16]. KLOTSKI discovers object dependencies and uses greedy scheduling to reduce PLT without violating them [17]. Instartlogic CDN recodes images to enable progressive load [28]. These methods, however, do not prioritize images by their density, and so our method is orthogonal.

B. Head of line blocking

Frame-based protocols, such as HTTP/2, suffer a "latency tax" as they traverse TCP connections [29]. The problem was first described by Clark and Tennenhouse as a disconnect between in-order delivery of bytes in TCP streams and frame-based application protocols, which only require in-order bytes within each frame [30]. As a result, the delivery of frame $i+1$

to the application layer may be unduly held up by frame i in the TCP receive buffer waiting for the retransmission of a lost TCP segment. Qian *et al.* define this type of HoL blocking as Type-L (loss) blocking [31]. They also define Type-S (sender) blocking, which occurs when delay sensitive HTTP/2 frames queue in the TCP send buffer behind frames of a large, delay insensitive object.

In this paper, we described that Type-S HoL blocking also occurs when a low density object is blocked in the TCP send buffer behind a high density object. The research community has proposed several solutions to eliminate Type-S HoL blocking specifically. Kernel-Informed Socket Transport (KIST) prevents Type-S blocking in the Tor network by buffering data at the application layer and passing to the transport layer data only at the socket serialization rate [32]. μTCP builds in multi-queue support into TCP [29]. TM^3 modifies kernel packet scheduling to assign data to packets, in priority order, right before their transmission [31]. SMig enables migration of HTTP/2 streams between connections, even over different network interfaces [33]. Finally, Goel *et al.* propose a multiple connection approach that does not require any modifications to the client [34]. The proxy solution we present in this paper mitigates Type-S HoL blocking among images of different densities.

VI. Conclusions

We presented a new method to improve web page Speed Index by prioritizing the images of low density, or low byte weight to pixel size ratio. We also introduced a new web page characterization metric, Object Density Distribution (ODD), and showed that most pages in the 200 most popular Alexa pages have non-zero ODD kurtosis and skewness, which creates meaningful opportunities for image prioritization. We implemented a Go HTTP/2 proxy to prioritize images based on image density embedded in object URLs. Density based image prioritization reduced SI in our experiments by around 5.7% on average and is an incrementally deployable, orthogonal technique to existing web performance optimization methods.

References

[1] S. Gomes, "Resource prioritization getting the browser to help you." https://developers.google.com/web/fundamentals/performance/resource-prioritization, 8 2018.

[2] D. N. da Hora, A. S. Asrese, V. Christophides, R. Teixeira, and D. Rossi, "Narrowing the Gap Between QoS Metrics and Web QoE Using Above-the-fold Metrics," in *Passive and Active Measurement*, 3 2018.

[3] A. Osmani, "Preload, Prefetch And Priorities in Chrome." https://medium.com/reloading/preload-prefetch-and-priorities-in-chrome-776165961bbf, 3 2017.

[4] P. Calvano, "Correlating performance metrics to page characteristics." https://discuss.httparchive.org/t/correlating-performance-metrics-to-page-characteristics/1548, 1 2019.

[5] "GTMetrix." https://gtmetrix.com, 9 2018.

[9] M. Belshe and R. Peon, "Hypertext Transfer Protocol Version 2 (HTTP/2)." https://tools.ietf.org/html/rfc7540, 5 2015.

[6] "Website Grader." https://website.grader.com, 9 2018.

[7] P. Meenan., "Optimizing HTTP/2 prioritization with BBR and tcp_notsent_lowat." https://blog.cloudflare.com/http-2-prioritization-with-nginx, 10 2018.

[8] M. Wijnants, R. Marx, P. Quax, and W. Lamotte, "HTTP/2 Prioritization and Its Impact on Web Performance," in *WWW*, 4 2018.

[10] A. Davies, "Tracking HTTP/2 Prioritization Issues." https://github.com/andydavies/http2-prioritization-issues/blob/master/README.md, 1 2019.

[11] "Apache Module mod_http2." https://httpd.apache.org/docs/2.4/mod/mod_http2.html, 2 2019.

[12] NGINX, "Module ngx_http_v2_module." http://nginx.org/en/docs/http/ngx_http_v2_module.html, 2 2019.

[13] Light Code Labs, "Caddy: server.go." https://github.com/mholt/caddy/blob/620f9687c8a97885e8fa03e6959ad80e99e1e0ba/caddyhttp/httpserver/server.go, 12 2018.

[14] Chrome Platform Status, "Image Decode API: img.decode()." https://www.chromestatus.com/feature/5637156160667648, 7 2018.

[15] Y. Weiss, "PSA: Splitting preload's as value "media" to "video", "audio" and "track"." https://groups.google.com/a/chromium.org/forum/#!topic/blink-dev/BN6tqGLBmuI, 3 2017.

[16] X. S. Wang, A. Krishnamurthy, and D. Wetherall, "Speeding up web page loads with shandian," in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, 3 2016.

[17] M. Butkiewicz, D. Wang, Z. Wu, H. V. Madhyastha, and V. Sekar, "Klotski: Reprioritizing web content to improve user experience on mobile devices," in *USENIX NSDI*, 2015.

[18] "Chrome DevTools Protocol Viewer." https://chromedevtools.github.io/devtools-protocol/, 2 2019.

[19] A. Cardaci, "Chrome Debugging Protocol interface for Node.js." https://github.com/cyrus-and/chrome-remote-interface, 2 2019.

[20] P. Irish, "Speedline." https://github.com/paulirish/speedline, 9 2019.

[21] M. Belsche, "More bandwidth doesnt matter (much)." http://www.belshe.com/2010/05/24/more-bandwidth-doesnt-matter-much/, 5 2010.

[22] "Apache Module mod_file_cache." https://httpd.apache.org/docs/2.4/mod/mod_file_cache.html, 2 2019.

[23] E. Bocchi, L. De Cicco, and D. Rossi, "Measuring the Quality of Experience of Web Users," in *Workshop on QoE-based Analysis and Management of Data Communication Networks (Internet-QoE)*, 2016.

[24] A. B. King, *Website Optimization: Speed, Search Engine & Conversion Rate Secrets*. O'Reilly, July 2008.

[25] S. Souders, *Even Faster Web Sites: Performance Best Practices for Web Developers*. O'Reilly, June 2009.

[26] Google, "SpeedTest." https://developers.google.com/speed/pagespeed, 9 2018.

[27] Surma, "IntersectionObservers Coming into View." https://developers.google.com/web/updates/2016/04/intersectionobserver, 7 2018.

[28] "Web Application Streaming: A Radical New Approach." http://go.instartlogic.com/rs/growthfusion/images/Updated_White_Paper_Instart_Logic.pdf, 2013.

[29] M. F. Nowlan, N. Tiwari, J. Iyengar, S. O. Amin, and B. Ford, "Fitting Square Pegs Through Round Pipes: Unordered Delivery Wire-Compatible with TCP and TLS," in *USENIX NSDI*, 4 2012.

[30] D. D. Clark and D. L. Tennenhouse, "Architectural considerations for a new generation of protocols," in *ACM SIGCOMM*, 9 1990.

[31] F. Qian, V. Gopalakrishnan, E. Halepovic, S. Sen, and O. Spatscheck, "TM3: Flexible transport-layer multi-pipe multiplexing middlebox without head-of-line blocking," in *ACM CoNEXT*, 12 2015.

[32] R. Jansen, J. Geddes, C. Wacek, M. Sherr, and P. Syverson, "Never Been KIST: Tors Congestion Management Blossoms with Kernel-Informed Socket Transport," in *USENIX Security Symposium*, 8 2014.

[33] X. Mi, F. Qian, and X. Wang, "SMig: Stream Migration Extension for HTTP/2," in *ACM CoNEXT*, 12 2016.

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