FlexiWeb: Network-Aware Compaction for Accelerating Mobile Web Transfers

Shailendra Singh
University of California, Riverside
singhs@cs.ucr.edu

Harsha V. Madhyastha
University of Michigan
harshavm@umich.edu

Srikanth V. Krishnamurthy
University of California, Riverside
krish@cs.ucr.edu

Ramesh Govindan
University of Southern California
ramesh@usc.edu

***Abstract*—**To reduce page load times and bandwidth usage for mobile web browsing, middleboxes that compress page content are commonly used today. Unfortunately, this can hurt performance in many cases; via an extensive measurement study, we show that using middleboxes to facilitate compression results in up to 28% degradation in page load times when the client enjoys excellent wireless link conditions. We find that benefits from compression are primarily realized under bad network conditions. Guided by our study, we design and implement FlexiWeb, a framework that determines both *when* to use a middlebox and *how* to use it, based on the client's network conditions. First, FlexiWeb selectively fetches objects on a web page either directly from the source or via a middlebox, rather than fetching all objects via the middlebox. Second, instead of simply performing lossless compression of all content, FlexiWeb performs network-aware compression of images by selecting from among a range of content transformations. We implement and evaluate a prototype of FlexiWeb using Google's open source Chromium mobile browser and our implementation of a modified version of Google's open source compression proxy. Our extensive experiments show that, across a range of scenarios, FlexiWeb reduces page load times for mobile clients by 35–42% compared to the status quo.

Categories and Subject Descriptors

C.4 [**PERFORMANCE OF SYSTEMS**]: Performance attributes; Measurement techniques; C.2.2 [**Network Protocols**]: Applications

Keywords

Cellular Network, Mobile Web Browsing, Compression Proxy, Middle Box

MobiCom'15, September 7–11, 2015, Paris, France.

DOI: http://dx.doi.org/10.1145/2789168.2790128.

1 Introduction

The ill-effects of slow websites, especially when browsing using a cellular network, are well-documented. Recent surveys suggest that two-thirds of users encounter slow websites every week [1], and that 49% of such users abandon a site or switch to a competitor upon experiencing large delays [1]. To address this, several systems augment mobile web browsing with support from middleboxes or proxies in the cloud (e.g., [2], [3], [4], [5], [6], [7]); many of these systems are widely used today, such as Opera Mini [5], Amazon Silk [6] and Chrome beta [7]. These cloud-supported mobile browsing options can potentially reduce download times, device energy consumption, and data usage costs.

Network conditions should dictate whether or not a middlebox should be used for performing content compression: The aforementioned middleboxes primarily compress content to be delivered to wireless clients based on the common belief that compression reduces the volume of bytes downloaded, and thus, improves performance. Unfortunately, this is not always the case. As our first contribution, we conduct an extensive measurement study which shows that, when network conditions are good, there could in fact be an *increase* in page load times (by as much as 28%) due to the use of middleboxes (we refer to this as `Proxy Assisted` browsing). The problem especially occurs if the web page to be downloaded only contains small objects (no high quality images or large scripts). The primary reason for this degradation is that diverting content to a middlebox causes additional delays due to longer routes and/or processing at the middlebox; these delays offset the gains achieved due to compression in some cases, and thus, worsen page load times. On the other hand, when network conditions are bad, `Proxy Assisted` browsing decreases page load times compared to `Conventional` browsing by 32%. This suggests that the decision on whether or not to use middlebox support for mobile web transfers must be made based on (i) network conditions and (ii) object sizes.

Content transformations ought to be network-aware: Today, most middleboxes for the mobile web apply the same compression to all content. Insufficient compaction could potentially lead to large page load times when network conditions are poor. If users and web providers are willing to compromise on content quality for better page load times, one could enable middleboxes to perform different degrees of content compaction based on network conditions. Specifically, this is possible in the case of images if the network conditions are bad. For example, when the quality of the client's network connection is very poor, the middlebox could

even take extreme steps, such as transform a color image to a gray scale version, to ensure reasonable load times.

Challenges: Realizing the above vision towards network-aware, dynamic usage of a middlebox for accelerating mobile web transfers is associated with two primary challenges: (1) For every object, the decision as to whether to fetch it directly or via the proxy depends on the object's size, but that is typically known only after fetching the object. Issuing a HTTP HEAD request just to query the webserver hosting the object for its size will add significant overhead. (2) The extent to which any particular object on a page should be compressed depends on the sizes of other objects on the page. However, given how the process of loading a web page works, the objects on a web page are only revealed iteratively as the page load proceeds.

Design and implementation of FlexiWeb: As our primary contribution, we design and prototype the FlexiWeb framework, which addresses the above challenges, towards providing dynamic, network-aware middlebox support for the mobile web. FlexiWeb has a novel design that encompasses three key elements. First, it uses an empirically derived model that determines, given an object's size and the network conditions, whether the object should be fetched via a proxy[1] or not. Second, FlexiWeb incorporates a classifier that predicts an object's size, based primarily on features derived from the object's URL. Third, FlexiWeb performs network-aware data compaction at the proxy, wherein appropriate transformations are applied to images based on the bandwidth available between the client and the proxy; other objects (e.g., Javascripts) are compressed as in conventional middleboxes. For this, FlexiWeb uses an online algorithm that selects near-optimal compaction levels while attempting to keep page load times within a latency target dictated by user tolerance (shown to be 2–5 seconds by user studies [8]). We cast this algorithm as a utility maximization subject to latency constraints.

Flexibility: While performing content transformation, it is important to ensure that web browsing sessions do ensure a good Quality-of-Experience (QoE). While transformations reduce delays, they must still cater to a user's or a web provider's requirements in terms of quality (e.g., a user may not want gray scale transformations). Hence, we allow users or web providers to set a minimum quality limit either via browser settings or via metadata embedded in a web page's HTML; this sets the maximum allowable compression allowed on each object in a web page.

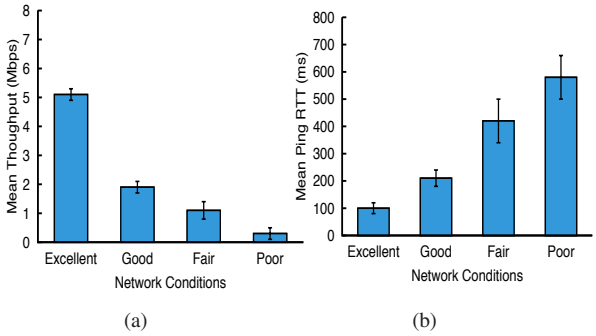
Evaluations: Finally, we perform extensive evaluations of FlexiWeb both via in-house emulations and real-world experiments using AT&T and T-Mobile's 4G networks in both static and mobile settings. Our experiments demonstrate that FlexiWeb reduces page load times by up to 35–42% as compared to both `Proxy Assisted` and `Conventional` mobile web browsing.

2 To proxy or not to proxy

We begin with a measurement study to understand the implications of using a middlebox for compressing mobile web content. We first describe our setup for the measurements and then discuss the results.

Client side setup: We set up multiple rooted mobile devices (HTC One phones with Android 4.3) running the open source web browser, Chromium, for Android. A Python program controls the browser via Chromedriver using RemoteWebDriver. All experiments are performed with a cold browser cache. We use "page load times" as the primary metric to capture performance. The page load time is defined as the time it takes for the browser to download and process all the objects associated with a web page. Most browsers fire a Javascript event (onLoad()) when the page is loaded. Chrome's remote debugging interface provides us the time taken to download each object in a web page.

[1]We use the terms proxy and middlebox interchangeably.



(a) (b)

Figure 1 (a) Average values of (a) Throughput (b) RTT between the client and the proxy server in different cellular network conditions

Network Conditions	RTT(ms)	Throughput(Mbps)	Loss Rate(%)
Excellent	100	5	0.006
Good	200	2	0.006
Fair	400	1	0.04
Poor	600	0.3	0.1

Table 1 Network Conditions (Note that MAC layer retransmissions limit error rates even under poor conditions [9])

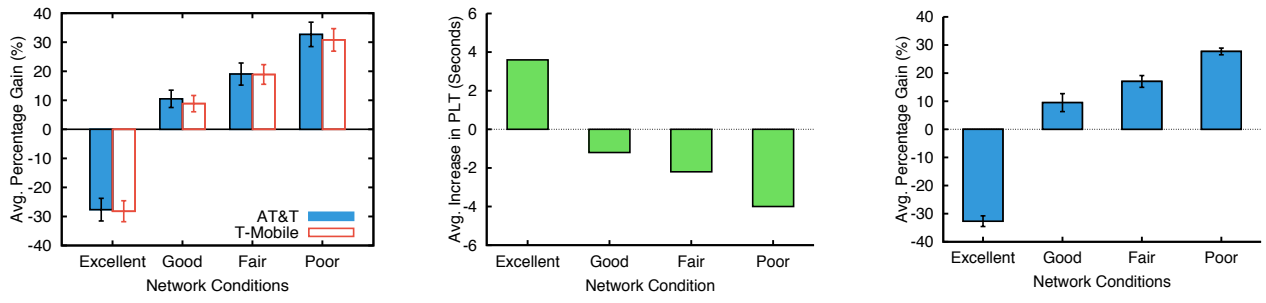
Server side setup: To ensure that a phone retrieved the same content each time it fetched a particular URL (in repeated experiments), we captured Alexa's top 500 websites (we believe these represent typically downloaded web pages) and replayed the content using the Web Page Replay tool [10]. All content was cached on our Web Page Replay server. Our server has 16 CPU cores and 64 GB of memory and is hosted on our campus network.

Network conditions: We operate over the 4G networks of two major US cellular providers, AT&T and T-Mobile. To examine the influence of different network conditions, for each provider, we carefully choose four locations where the RTT and throughput values were similar to those listed in Table 1. We choose these locations after monitoring the network conditions by running a long term (2 day) experiment to ensure the stability of the conditions (they stayed more or less the same) as shown in Figures 1a and 1b.

Data collection: For each page downloaded, we record the browser-reported page load time, as well as the load time for each individual resource on the page. In addition, we run `tcpdump` on the phone to capture a trace of all network traffic during the page load. In each run, we first load a page by downloading the resources on a page directly from the webservers hosting them, and then via a compression proxy, back-to-back. We perform 10 trials for each experiment.

Today's compression proxies can increase mobile web page load times: First we perform experiments over both AT&T and T-Mobile's networks, with currently deployed commercial compression proxies. Google's Chrome mobile browser comes with the `Proxy assist` option. Upon enabling this, the browser redirects all the requests to Google's compression proxy. Using this feature, we perform experiments with a *live version* (clients download the webpages directly from the source web sites as opposed to downloading a replay from our web server) of Alexa's top 500 websites.

Figure 2a shows the percentage gain in performance in different network conditions, when using Google's proxy in comparison to



(a) (b) (c)

Figure 2 (a) Gains from using a commercial compression proxy (Google compression proxy) in downloading Alexa's top 500 web pages under different client network conditions. Gains are measured in comparison to a `Conventional` browser. (b) Average increase in Page Load Time (PLT) in seconds from using a commercial compression proxy (Google compression proxy) in comparison to a `Conventional` browser.(c) Gains from using a `Proxy Assisted` browser (using our own proxy) in downloading Alexa's top 500 web pages under different network conditions. Gains are measured in comparison to a `Conventional` browser.

a `Conventional` browser (where all content is retrieved from the source). We see that the use of the proxy provides an average gain of about 32% in bad network conditions. In excellent network conditions, the `Conventional` browser (no proxy) outperforms the `Proxy assisted` browser by about 28%. Figure 2b shows the average gain in page load times in seconds. We see that the proxy can increase the page load time by ≈ 4 seconds. According to prior studies [11] a 1 second delay could potentially result in a nett loss of 2.5 million dollars in sales for an e-commence website. It is also seen that a 4 second delay can cause up to 25% increase in page abandonment. These results clearly demonstrate that the use of a proxy can significantly hurt performance when network conditions are excellent; however, as conditions degrade, compression at a proxy can provide significant benefits.

An in-depth study: To get a further understanding of the implications of using a compression proxy, we set up our own proxy (to emulate the behavior of Google's proxy) and conduct more in depth studies.

Compression proxy setup: We set up the compression proxy on Amazon EC2 to have a controlled environment. The proxy is located in northern California, relatively close to the geographical region of the client. We use Google's open-source compression proxy module called PageSpeed, which we setup as a forward proxy [12]. We use the optimization strategies [13] recommended for reducing page load times. These include combining and minifying JavaScript (JS) and Cascading Style Sheets (CSS) files, inlining small resources, and others. We configure PageSpeed to dynamically optimize images by removing unused metadata, resizing images to specified dimensions, and re-encoding images to the WebP format (which requires fewer bytes than other popular formats such as JPEG and PNG).

Figure 2c presents the average gain in page load time with the compression proxy as compared to `Conventional` browsing (or direct downloads). We see that the results are very consistent with what was observed with the real-world compression proxies.

The question that we then seek to answer is: *"why does performance degrade in excellent network conditions due to the use of a proxy?"* Via a careful study, we find that this is primarily due to penalties associated with loading content via a compression proxy: (i) circuitous routing between the client and web servers via the middlebox, and/or (ii) processing delays at the middlebox.

Penalties due to route stretch: Zarifis et al [9] provide a detailed analysis of path inflation in mobile networks and suggest that factors that primarily contribute to the path inflation are diversions due to (i) ingress points and (ii) peering points. With regards to the former factor, the device's carrier network may not have ingress points to the Internet in the device's area. With regards to the latter, the carrier network may connect with a web service provider at a peering point or Internet exchange point (IXP), which may be at a location distant from both the mobile device and the web provider.

An analysis of our traceroute data indicates that, in the presence of a proxy, traversal of multiple peering points (because of three different networks viz., the carrier network, the network hosting the proxy and the network hosting the web server, instead of two) is likely to lead to longer paths. Indeed, we observed that end-to-end routes in some cases were inflated by up to 8 hops due to the use of the proxy, which resulted in increases in RTTs by up to 45ms.

For example, when we fetched cnn.com without using the proxy, from a location in Southern California, the request for object `http://i.cdn.turner.com/cnn/.e/img/4.0/logos/city_of_tomorrow_bw.png` traversed through AT&T's core mobile network to a CDN (content distribution network) server which was geographically proximate to the client (near Los Angeles). We also found that ingress points for AT&T's mobile core network were in the same geographical location. When fetching the same object via the compression proxy, the request went to the proxy server located in northern California and the request for the object was forwarded to the CDN located near the proxy server (northern California). We observe a similar effect when fetching the same object via Google's compression proxy; the request was sent to Google's compression proxy located in northern California while the closest CDN server was in region Southern California (Los Angeles).

We wish to point out here that the cellular providers also deploy (transparent) proxies in their core network. According to [14] however, only Sprint performs any form of content rewriting. Other providers (T-Mobile, ATT and Verizon) primarily perform delayed handshaking, connection persistence and redirection using such a transparent proxy.

Penalties due to processing overhead: We find that the processing overhead to convert an image to WebP format (the image transformation performed by Google's compression proxy) ranges from 10ms to 30ms per image, depending on the image's size; if a web page contains a large number of objects, the total processing delay can be significant. In general, when compared to JPEG, the encoding speed for WebP is $\approx 10X$ slower (and the decoding is $\approx 1.4X$ slower) but it does provide $\approx$ a 30% gain in terms of a size re-

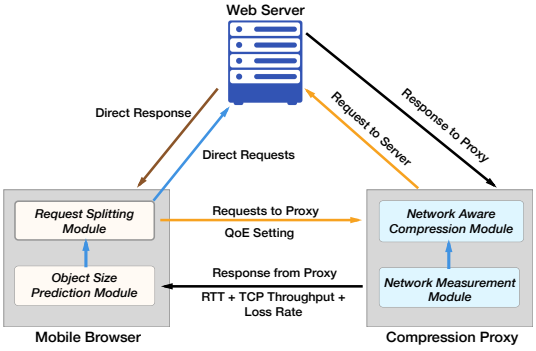


Figure 4 Overview of FlexiWeb's architecture

duction, on average [15]. A commercial proxy can reduce the processing times by using advanced caching techniques (to amortize processing costs across users) but websites are increasingly personalizing their content, e.g., amazon.com shows different items to different users based on their browsing patterns. This, in turn, results in the fetching of different objects for different users.

Impact of object size: To summarize, our evaluations indicate that neither the `Conventional` nor `Proxy Assisted` approach is the winner in all scenarios (in terms of network conditions). To further understand the implications of using `Proxy assisted` browsers, we next analyze the data collected for both types of browsing to understand the impact of network conditions on load times for objects of different sizes. We partition objects into 7 bins: 0–1 KB, 1–3 KB, 3–6 KB, 6–10 KB, 10–20 KB, 20–40 KB, and $\geq$ 40 KB. In Figure 3, we compare the average load times computed over all objects in each category, with both browsing strategies in different network conditions. We see that when the conditions are good, using the proxy is only beneficial for objects larger than 30 KB. This suggests that the delays due to indirection via the proxy outweigh the benefits of data compaction for small objects. As network conditions degrade, fetching compressed objects via the proxy improves object download times as compared to fetching them uncompressed directly from the source. In the extreme case, we see that the proxy's compression enables better download times for objects of all sizes in poor network conditions.

Takeaways: In summary, our measurement studies suggest the following. In very good/excellent network conditions, it is best if only larger sized objects ($>$ 30 KB) are diverted to a compaction middlebox during a web page download. Under fair to poor network conditions, one should retrieve all content indirectly via the middlebox. While FlexiWeb addresses other key challenges as well, these takeaways form the basis for middlebox usage in our framework.

3 Design of FlexiWeb

In this section, we describe the design of the FlexiWeb framework. As discussed earlier, FlexiWeb seeks to reduce page load times while browsing the web on mobile devices. The key property of FlexiWeb is that it is adaptive and achieves its goal under all network conditions.

Overview: To improve the performance vs. experience tradeoff on the mobile web, our over-arching goal is to enable mobile clients to dynamically select which resources on a web page to download via a proxy and to enable the proxy to determine how it should transform every resource that it relays. This goal imposes three logical challenges:

- How can a client account for the characteristics of the objects on a page and its network conditions to determine what to fetch directly from the source web servers and which objects to fetch via the proxy?
- Since the determination of whether to fetch an object via the proxy or not depends on the object's size, how can we predict an object's size before fetching it?
- To ensure that all objects on a page are delivered to a client within a target load time, the manner in which the proxy transforms any particular object on a page must depend on the sizes of other objects on the page. How can the proxy do so, given that all the objects on a page are only revealed iteratively during the page load process?

Our architecture of FlexiWeb towards tackling these challenges is shown in Figure 4. The *Network Measurement component* at the proxy measures the network's characteristics via a series of measurements. The measurements made by FlexiWeb do not consume excessive resources on the client side in terms of bandwidth or energy. These measurements are then fed back to the mobile browser. The mobile browser contains an *Object size prediction module* which estimates object sizes in a web page that it seeks to download. This module essentially uses (i) measured distributions to determine what objects (in terms of sizes) to fetch via the proxy after transformation, and what objects to fetch directly from the source and (ii) an incremental learning algorithm to predict object sizes. The *Request splitting module* then fetches each object, either directly from a web server or via the proxy; large objects are always fetched via the proxy, while the decision on "from where to fetch a small object?" is made based on network conditions (this part of FlexiWeb uses the inferences from our measurements in Section 2).

The *Network Aware Compression module* at the proxy fetches the objects from their sources. Note here that different objects could be potentially fetched from different web servers (e.g., ads may be fetched from a different domain than the one in which the web page being fetched is hosted). Since not all objects are fetched simultaneously, the proxy has to decide on the level of compression (transformation) to be performed on each object as it is fetched, with the objective of trying to keep the total page load time within a target. The proxy meets this goal by using a novel algorithm to dynamically choose the compression level for each object (images specifically) prior to sending it to the mobile client. Specifically, it performs an online optimization to determine object transformations that allow the download to fit within a target budget page load time; it reverts to a minimal transformation (maximum compression) when the optimization formulation is infeasible.

Handling Videos: Videos in a webpage are downloaded as an image (snapshot from the video) when a page is loaded by the browser. If a user clicks on the play button of the video it starts downloading/streaming the video from a server. Video streaming is not considered to be part of the webpage download [16] and hence, we do not address videos explicitly in this work.

In the rest of this section, we elaborate on the different functionalities of FlexiWeb.

3.1 Splitting requests

Current proxy-assisted browsers either fetch all or none of the objects in a web page, via a middlebox. However, as seen earlier, this model can hurt page load times when network conditions are excellent. To minimize the page load times given the conditions, FlexiWeb seeks to identify which objects should be diverted to the proxy and which ones should not. Based on our measurements in Section 2, we create a mapping between how objects should be re-

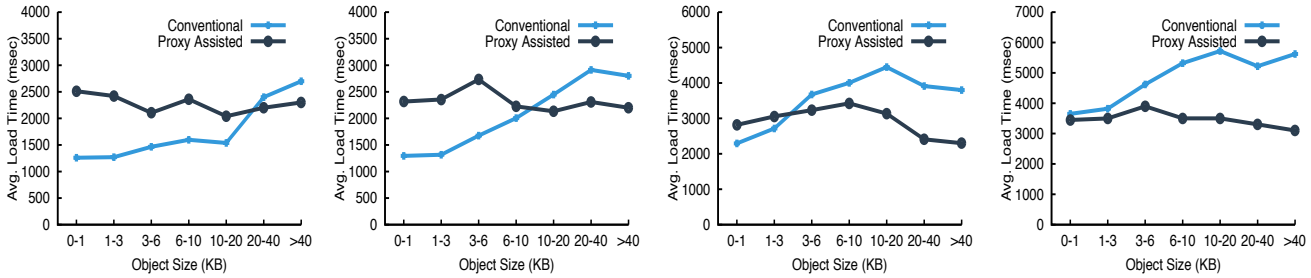


Figure 3 Download times of objects under various network conditions (a) Network: Excellent (b) Network: Good (c) Network: Fair (d) Network: Poor

Network Condition	0–1 KB	1–3 KB	3–6 KB	6–10 KB	10–20 KB	20–40 KB	$\geq$ 40 KB
Excellent	Direct	Direct	Direct	Direct	Direct	Proxy	Proxy
Good	Direct	Direct	Direct	Direct	Proxy	Proxy	Proxy
Fair	Direct	Direct	Proxy	Proxy	Proxy	Proxy	Proxy
Poor	Proxy	Proxy	Proxy	Proxy	Proxy	Proxy	Proxy

Table 2 Mapping that dictates when to fetch an object directly from the source server and when to fetch it via the proxy

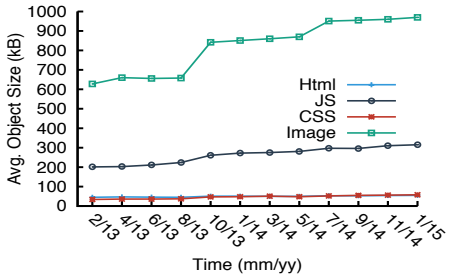
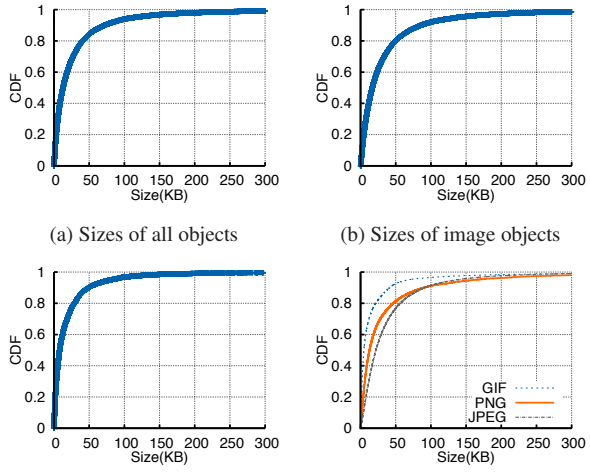


Figure 5 Average object size of top 4000 Alexa web pages over last 2 years



(a) Sizes of all objects

(b) Sizes of image objects

(c) Sizes of CSS, JS, and HTML objects

(d) Sizes of different types of images

Figure 6 Characteristics of top 4000 Alexa Web pages.

trieved based on their sizes and current network conditions; Table 2 depicts this mapping. The *Request Splitting Module* uses this mapping to dynamically send the request for an object either directly to the web server or to the proxy.

Dynamically selecting how to fetch an object based on its size is challenging because any object's size is not readily available to the client. After fetching the main HTML file (e.g., index.html) of a web page, current browsers parse the file to determine the next object that is to be fetched in order to render the web page. At this point, the browser only knows the object's URL; the object size information is yet unknown. While the browser could try to determine an object's size by issuing a HTTP HEAD request [17] to the web server hosting the object, this would add significant overhead since it imposes an additional round trip of wide-area communication for every object.

3.2 Predicting object sizes

To predict the size of an object given only its URL, we rely on learning and applying a predictive classifier. Based on the objects that it fetches over time the proxy continually builds this classifier, and periodically sends its prediction model to the client. The data from Alexa's top 4000 web pages gathered by http archive [16], shows that the average size of web page objects changes only once every few months (as shown in Figure 5). Thus, it suffices that the update frequency of the prediction model is set to (say) every few months.

Training the model: To construct a prediction model, a dataset to train the model is required. The middlebox records the URLs of the objects that it fetches and their corresponding sizes, and it feeds this information as input to a machine learning algorithm to predict the sizes of objects not seen thus far. Figure 6(a) depicts the distribution of object sizes in such a sample dataset, which comprises all objects seen on Alexa's top 4000 websites. We see that over 80% of objects are smaller than 50 KB. This is also true in the case of images (Figure 6(b)). Figure 6(c) shows that objects with text content (JavaScript, HTML, and CSS) are even smaller. We find that images are one of the biggest contributors to large web page sizes, and hence, we looked at the distribution of the sizes of different image types. Figure 6(d) shows that GIF images are much smaller in size than PNGs and JPEGs.

Rather than attempt to predict the precise size of every object, which is likely an impossible task, we leverage the fact that we only need to map every object to one of the bins in Table 2. Therefore, the problem at hand is to accurately predict the size range given an object's URL.

Extracting features: To perform this classification of object URLs to size ranges, we extract various features from every URL. First, we extract a "bag of words" [18] from the host name in the URL. For example, the host name in the URL `http://i.cdn.turner.com/cnn/.e/img/3.0/global/icons/gallery_icon2.png` yields the following bag of words: {i, cdn, turner, com}. In addition to features extracted from the host name, we include features from the URN and the type of content. Hence, the feature set for the above URL would be domain = {i, cdn, turner, com}, urn = {cnn, .e, img, 3.0, global, icons, gallery_icon2.png} and type = png.

Classifier: The problem of classifying a URL into one of the size ranges is a multi-class classification problem, i.e., it is a classification task with more than two classes. Many multi-class classification algorithms make the assumption that each sample is assigned to one and only one label; in our case, this means that an object's URL can only be "classified" into one size range.

The random forest learning algorithm [19], which is known to perform well for multi-class classification problems can be utilized at the proxy for our purposes. It uses multiple decision trees and the final decision is made based on an aggregation of different decision outcomes from the individual trees. The main parameters of the random forest algorithm are the maximum depth of each (individual) tree, the maximum number of trees, and the maximum number of features used to train the model; details can be found in [19].

Execution: In summary, the object size prediction component of FlexiWeb executes as follows. First, the proxy labels every URL in the training dataset by assigning each object to the appropriate class based on its size. Next, the proxy uses the random forest learning algorithm to generate a classifier model based on the extracted features from the labeled dataset. The client browser then uses this model to predict the size of the object associated with each request, and subsequently (in conjunction with network condition assessments), it makes a decision on whether to retrieve the object via the proxy or directly from the source.

3.3 Assessing Network Conditions

An assessment of network conditions by the *Network measurement module* at the proxy serves two purposes. First, it enables the *Request splitting module* at the client to determine whether to retrieve each object from the source directly or via the proxy. Second, it also enables the proxy to make an informed decision on the extent to which it should compress each object. To assess network conditions, the *Network measurement module* at the proxy tracks the RTT, the loss rate, and the TCP throughput to the client.

A simple way to accurately estimate network conditions is to perform measurements in the background; the client and the proxy would exchange packets periodically to infer network conditions. However, such an approach will be prohibitive in terms of bandwidth and energy overheads on the client side. In FlexiWeb, to enable accurate estimation of the average RTT and throughput, the client browser always downloads the main HTML file for every web page via the proxy. The main HTML file is typically sufficiently large for the purposes of FlexiWeb's estimation of network conditions, since the HTML's transfer requires the transmission of tens of packets. This also solves the issue of cold start, when the client browser is sending a page request for the first time.

From the HTML's transfer, the *Network measurement module* obtains a sequence of RTT samples, and uses the median sample value to estimate the RTT to the client; we borrow this approach from [20]. Since web page loads last only tens of seconds, these RTT measurements suffice in predicting subsequent data transfer delays to the specific client.

The proxy sends this "network condition report" back to the client in the object response headers of the requests; this enables the client to determine the network conditions (excellent, good, fair or poor as in Table 1) and how to fetch other objects on the web page (using Table 2).

3.4 Network Aware Compression

Current practices: Current proxy-assisted browsers transform content in a pre-determined manner. For example, scripts, CSS files, and other text-based content are minified and zipped before they are sent to the mobile client [21]. On the other hand, proxies convert images to a fixed image format with a fixed compression ratio [13]. For example, Google compression proxy's recommended setting is to transform images to Google's *WebP* image format with a pre-defined quality; WebP is known to have much better performance in comparison to other image formats (e.g., the average WebP file size is 25–34% smaller compared to the JPEG file size with equivalent image quality).

Problem formulation: In contrast to the status quo, we seek to adaptively transform any web page's content in a manner that enables us to deliver the page within the user's attention span, irrespective of the client's network conditions; a user's attention span has been shown to be in the 2 to 5 second range previously [8].

To deliver a web page's content within the user's attention span, in this work, we focus only on *adaptively* transforming images on the page. Since images make up around 65% of the bytes on the average web page [16], transformation of images (to either significantly compressed versions or lower qualities) can significantly reduce page load times. Text and Javascript are compressed as with traditional proxy assisted browsers (to static extents). The adaptive transformation of images is based on assessed network conditions. If network conditions are good very little compression is invoked; on the other hand, if conditions are poor, higher levels of compression (including a lowering of the image quality) are in play. Our focus here is primarily to achieve the right balance between page load time and user experience. Reducing the download time of an image by applying an appropriate transformation is important, but at the same time, quality or utility of the transformed image is also important. To achieve the right balance, we associate each transformed version of an image with a cost and a utility.

Cost: We define the cost of a transformation as the time taken to download the transformed image from the proxy to the client. Since TCP throughput can be modeled as $\frac{MSS}{RTT\times\sqrt{P}}$ [22] (where MSS is the maximum segment size, RTT is the round-trip time between the proxy and the client, and P is the packet loss rate), we compute the download time for an image of size S bytes as $\frac{S\times RTT\times\sqrt{P}}{MSS}$; the proxy measures the RTT and loss rate using the technique described earlier. Here, the size S corresponds to the *transformed size* of the image. Thus, the higher the degree of compression due to the transformation, the lower will be the cost.

Utility: We use the Peak signal-to-noise ratio (PSNR) as the utility resulting from a transformation. PSNR is the most widely used metric to quantify the quality or utility of images. It essentially captures the relative quality between two images and is defined as:

$$PSNR = 10log\left(\frac{MAX^2}{MSE}\right),$$

where MAX is the maximum possible pixel value (defined next) and MSE is the mean-squared error given by:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2.$$

In the above, the two images are of size $m \times n$ and $I(i,j)$ and $K(i,j)$ are the pixel values in the two images at position (i,j). The pixel values essentially reflect the intensity level in each pixel; thus, if the intensity is represented using 8 bits, then MAX will be 255. For color images (with three RGB values per pixel), note that the MSE is the average over all RGB values. With higher levels of compression or transformation, MSE increases and thus, PSNR decreases.

The proxy calculates the PSNR (utility) with each transformation with respect to the original image. Clearly, the more aggressive the transformation, the lower will be the utility of the image.

Cost vs. utility trade-off: As evident from the above discussion, a higher degree of compression reduces both utility and cost; an attempt to either decrease cost or increase utility adversely affects the other. Thus, though FlexiWeb should ideally maximize utility and minimize cost, both objectives cannot be realized simultaneously. Instead, it seeks to do the best it can in reducing cost and increasing utility as follows.

For tractability, we consider a limited number of transformation choices, and the proxy chooses from one of these options depending on network conditions. The set of transformations range from high utility and high cost to low utility and low cost: WebP with 85% quality, WebP with 65% quality, WebP with 45% quality, WebP with 25% quality, WebP with 5% quality, and a gray scale version of the image. As one might expect, the lowest quality image (gray scale) has the least associated cost but also offers the least utility; the highest quality image provides the highest utility but also incurs the highest cost.

Processing times: In the above discussion, we characterize the cost associated with a transformation to be the time taken to download that specific transformed version of an image. In reality, there is an additional cost due to processing the image. We find that regardless of which transformation we perform on each specific object, the processing delay does not change by much. We perform all of the transformations on 1000 images from Alexa's top 500 webpages and we find that the processing times are almost identical; in the worst case, the difference in processing time is about 8 % for a very few objects (we omit results due to space constraints). Thus, the processing cost does not influence the choice of transformation and is hence ignored here.

Compressing objects based on network conditions: Utilizing the cost vs. utility characterization of web content as above, we cast the proxy's task of performing network aware compression as follows.

Suppose that the client needs to download a total of N images in sequence over the duration of a web page download, and that it has a total page load time budget of B seconds (within which the page should be ideally downloaded). For every image object i, let us assume that the proxy can apply M_i different transformations. The question then is "which transformation should the proxy apply on each image ?" The overall goal here is to maximize the sum of utilities (i.e., PSNR values) of all the selected versions for the N images, subject to the constraints that (i) exactly one version for each requested image must be selected, and (ii) the total cost of all the selected versions must be within budget B (within the desired page download time). Note that maximizing the sum of the PSNR (chosen as the utility) values corresponds to a minimization of the sum of expected distortion over the images in the web page; the lower this value, the better the quality of the downloaded content and thus, user experience.

The above optimization can be formulated as follows:

$$maximize_{x_{ij} \in \{0,1\}} \quad \sum_{i=1}^{N} \sum_{j=1}^{M_i} u_{ij} x_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{M_i} x_{ij} = 1, \forall i$$

$$\sum_{i=1}^{N} \sum_{j=1}^{M_i} c_{ij} x_{ij} \leq B,$$

Here, the indicator variable x_{ij} is 1 if version j of image i is selected, and 0 otherwise. u_{ij} and c_{ij} are the utility and cost, respectively, associated with version j of image i.

The transformation selection problem as formulated in Equation 1 maps to the Multi-Choice Knapsack Problem (MCKP). The time budget is the size of the knapsack, and the goal is to fill in objects (images) such that the total utility of the objects in the knapsack is maximized. However, in our case, a client's requests for objects are made online, i.e., the middlebox does not have a priori knowledge of future image requests since the objects are dynamically fetched. Thus, we need an online algorithm for solving the MCKP problem.

The online MCKP problem is well studied. Competitive online algorithms have been proposed [23] with the relaxed constraint $\sum_{j=1}^{M} x_{ij} \leq 1$ for all i, i.e., the algorithm need not to choose a version for each image (image can also be discarded). We modify the relaxed online MCKP algorithm designed by Zhou et al. [23] for solving our problem. Specifically, (i) we do not allow for the discarding of images and (ii) we modify the time budget dynamically to account for time spent either due to transfer of other content (e.g., compressed Javascripts or text) or inactivity due to waiting for objects. However, these uncertainties can cause us to exceed the original target time budget. Thus, our modified goal is to try and stay within the budget, but if we are to exceed it, to do so to the minimum extent possible.

Now, the online transformation selection problem 1 can be transformed into the following relaxed problem with the inequality constraints shown; if an online algorithm for the relaxed problem has competitive ratio C (the performance of the algorithm w.r.t the optimal), then, so does the original problem in Equation 1.

$$maximize_{x_{ij} \in \{0,1\}} \quad \sum_{i=1}^{N} \sum_{j=1}^{M_i} u'_{ij} x_{ij} \tag{2}$$

$$\text{subject to} \quad \sum_{j=1}^{M_i} x_{ij} = 1, \forall i$$

$$\sum_{i=1}^{N} \sum_{j=1}^{M_i} c'_{ij} x_{ij} \leq B,$$

In the above formulation, the key idea is to subtract the utility and cost of the cheapest transformation of each image from other transformations of the same image, and use these *modified* utilities u' and costs c' as the new representations associated with the corresponding transformations. If the relaxed problem does not choose a transformation, in our case we default to the cheapest transformation. The reader is referred to [24] for the proof of the above claim.

The algorithm provides a competitive ratio of $log(\frac{U'}{L'})+1$, where the utility to cost ratio of the transformations are upper and lower

Algorithm 1 Online selection of best image transformations

1: **Initialize:** $b = 0$, Q (QoE level)
2: **Event:** Image i requested at time t_i
3: **if** $Q \leftarrow auto$ **then**
4: $M_i \leftarrow M_i$
5: **else**
6: **if** $Q \in \{1, 2, 3, 4, 5\}$ **then**
7: $M_i \leftarrow M_i - Q$, by removing the lower Q transformations
8: **end if**
9: **end if**
10: **for** $j = 1 \rightarrow M_i$ **do**
11: $u'_{ij} \leftarrow u_{ij} - u_{i1}$
12: $c'_{ij} \leftarrow c_{ij} - ci1$
13: **end for**
14: $transformations \leftarrow \{j | \frac{u'_{ij}}{c'_{ij}} \geq \phi(b, B)\}$
15: **if** $transformations$ is not empty **then**
16: $J \leftarrow \arg\max_j \{u'_{ij} | j \in transformations\}$
17: **if** $b + c'_{ij} \leq B$ **then**
18: $selectedTransform \leftarrow J$
19: **else**
20: $selectedTransform \leftarrow 1$
21: **end if**
22: **else**
23: $selectedTransform \leftarrow 1$
24: **end if**
25: Deliver transformation $selectedTransform$
26: **if** time $t_i \geq b$ **then**
27: $b \leftarrow t_i + c'_{i,selectedTransform}$
28: **else**
29: $b \leftarrow b + c'_{i,selectedTransform}$
30: **end if**

bounded by U' and L', respectively. Zhou et al. also assume that all transformed versions are much smaller in size than the knapsack, which is true of objects on a web page (each image is transferred in a time $<< B$).

Algorithm 1 presents our approach to modifying and using Zhou et al.'s solution to our problem of selecting the appropriate transformations of images at the FlexiWeb proxy. The intuition behind the algorithm is as follows. We only want to pick image transformations that have a sufficiently high utility to cost ratio, i.e., ones that satisfy a certain efficiency threshold. As we spend more budget, this threshold should increase (in other words, the cost for transferring objects later in the page load should decrease). This efficiency threshold is captured by a function $\phi(b(t), B) = (\frac{U}{L})^{\frac{b(t)}{B}} \frac{L}{e}$, where $b(t)$ is the amount of the budget spent at time t.[2] $\phi(b(t), B)$ increases with $\frac{b(t)}{B}$, the fraction of the knapsack filled at time t. Therefore, as time progresses, fewer transformations satisfy the efficiency condition.

Referring to Algorithm 1, the proxy first initializes the current usage b to 0. Then, the transformations $u_{ij} \longrightarrow u'_{ij}$ and $c_{ij} \longrightarrow c'_{ij}$, are performed. The proxy then considers the set of transformations that have a utility-to-cost ratio that is higher than $\phi(b, B)$. It chooses the transformation that has the maximum utility and fits within the budget; otherwise (i.e., if none fit within the budget), it chooses the cheapest transformation. At the end of each step (i.e., after an image is transformed and sent to the client), the proxy updates b with the cost of the selected transformation. The current budget usage is updated after the selection of a transformation. In case of inactive time periods (because of waiting for objects) an increase in knapsack usage is applied (lines 19-22 in Algorithm 1). Whenever static compression is applied to objects such as JS, CSS etc., we again apply an increase in knapsack usage (as with inactivity periods). If the target budget is exceeded, then the middlebox simply performs the cheapest transformation on each image, prior to sending it to the client. Note here that we simply subtract an object's cost from the budget, despite the client fetching multiple objects on a page in parallel, because we consider bandwidth to be the bottleneck between the client and the proxy.

[2]We perform a one time estimation of U and L offline, using a training set of images.

QoE vs. page load latency: While transforming content, it is important to ensure that web browsing has a good, associated Quality-of-Experience (QoE). While transformations reduce delays, they must still cater to a user's or a web provider's requirements in terms of quality (e.g., a user may not want gray scale transformations). Hence, users or web providers can set a minimum quality limit either via browser settings or via metadata embedded in a web page's HTML; this sets the maximum allowable compression level (QoE level) allowed on each object in a web page. The QoE level is set to an auto mode by default. In this mode, all possible (M_i) transformations are considered (when applying Algorithm 1). Instead of the auto mode, the user or the provider can specify a QoE level from 1 to 5. If QoE level 1 is chosen, Algorithm 1 can use $M_i - 1$ transformations; the the lowest quality transformation is left out. If a QoE level of 5 is chosen, Algorithm 1 can only apply a single transformation, viz., the one that performs the minimum compression and retains the highest quality (WebP with 85% quality). Note that the higher the level of QoE chosen, the larger could be the deviation (increase) with respect to a desired page load time. At this time, FlexiWeb chooses the higher of the QoE levels specified by the user and the provider.

4 Implementation and Setup

In this section, we first describe our implementation; subsequently we elaborate on our experimental setup.

4.1 Implementation of FlexiWeb

Client side implementation: We modify Google's Chromium 38.0.2125.50, an open source Android browser to implement the client side modules of FlexiWeb. Specifically, we make changes to the $url\_request$ module in Chromium to 1) determine whether or not to go through the proxy, for each object, and 2) to predict the size of any object based on its URL. We also modify Chromium to store network measurements in the browser cache after retrieving them from the HTTP request response header sent by the proxy. For each web page, the request for the main HTML file always goes through the compression proxy; this allows the proxy to gather accurate network condition estimates (RTT and throughput). These estimates are returned to the client in a custom field in the HTTP response (sent by the proxy). The estimates stay valid for 15 seconds, and if no object is fetched via the proxy by then, the client sends the immediately following request via the proxy. Based on an object's size and the current network conditions (stored in the browser), our modified version of Chromium chooses between retrieving the object via a proxy or directly from the source.

Proxy side implementation: We configure the compression proxy with 4GB memory and 2.40 GHz Intel Core 2 CPU, running Ubuntu 12.04. It is implemented using Google's mod_pagespeed-1.8.31.4 module running on top of an Apache 2.2 web server. TCP CUBIC is used. The Apache web server runs in the forward proxy mode. All the requests pass through mod_pagespeed module before content is compressed. We implement the passive RTT, throughput and loss rate estimation scheme described in [22] as a kernel module; the estimated values are exposed using the Linux socket interface, to other programs and modules. Specifically, these values

are used by the network aware compression module to calculate the cost of a transformation. We implement the network-aware compression module by modifying the mod_pagespeed's image transformation module to invoke network-aware compression (using Algorithm 1) instead of static compression. To report the network conditions to the client, we implement an Apache module to control and modify the HTTP response headers. This Apache module reads the current RTT, loss rate and throughput values and adds them to every out going response to the client using the previously discussed custom header field.

4.2 Experimental Setup

Scenarios for evaluation: We have three different scenarios in which we evaluate FlexiWeb.

Controlled settings: To reduce variability and control network conditions, we tethered our client phone to a laptop using a USB connection and applied traffic shaping to the tethered connection using Dummynet [25]. We emulated a 4G network with an uplink bandwidth of 1 Mbps [26] and with varying downlink bandwidths shown in Table 1.

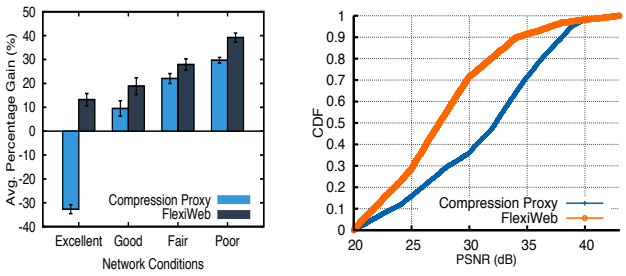
An automated script was used to change the values of the link bandwidth, the loss rate and RTT for every test case.

Static clients on cellular networks: To evaluate FlexiWeb on real cellular networks, we use AT&T and T-Mobile's 4G network connections. To create different network conditions we run experiments at the same locations that we used to gather our measurements in Section 2. Due to space constraints, we only present results from AT & T's network except in some sample cases.

Mobile scenarios: Finally, we also evaluate FlexiWeb in mobile scenarios. We choose around 20 paths in two US metropolitan areas in two states with a mix of local streets and Interstate highways. Our speeds on interstate highways were around 65 miles per hour while on city streets our speeds were 30-40 miles per hour. We omit the exact details of the paths to preserve anonymity.

Web pages requested: We use the top 500 web sites visited by mobile users to run our tests (the top Alexa sites). These have a good mix of news websites, online shopping and auction sites as well as professionally developed websites of large corporations. These websites contain anywhere from 5 to 323 objects. The objects in these sites were spread across 3 to 84 domains. Each web site had HTML pages, Javascript objects, CSS and images. Due to experimentation constraints (time), the experimental results reported for mobile scenarios are only using the top 50 web pages.

Execution: We setup 2 rooted mobile devices (HTC One phones with Android 4.3) running Chromium (with our modifications) for Android. We generated a random order in which to visit the web sites and used that same order across all experiments. The period between website requests was set to 60 seconds both to allow for websites to load completely, and to reflect a nominal think time that users typically indulge in between requests. If the web page took a much shorter time to load, the system was idle until the 60 second window elapsed. We used the "page load time" as the primary metric of performance. We alternated our test runs between "Direct", "Compression Proxy" and "FlexiWeb" to ensure that temporal factors did not affect our results. With Direct, requests are directly sent to the source web server. With Compression proxy, all the requests are sent to a proxy with static compression settings where images are transformed to WebP with a quality of 75% (this is what is done with commercial proxies today [27] to achieve some compression without compromising the quality significantly). We ran each experiment multiple times at different times during the day.



(a) (b)

Figure 7 : (a) Performance gains with FlexiWeb (b) PSNR of transformed images

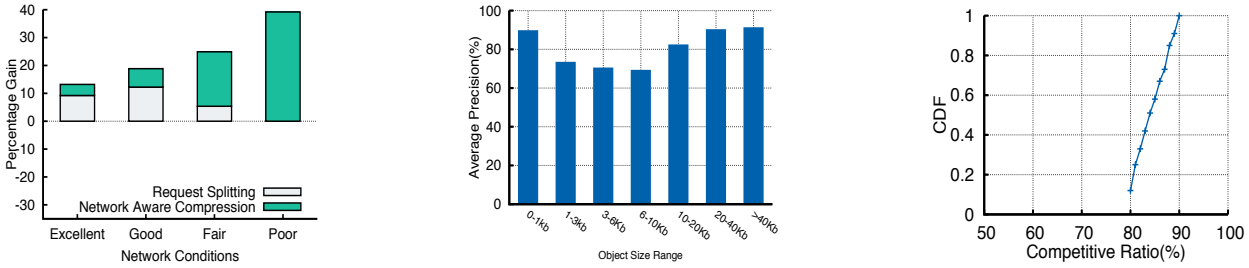
5 Evaluation of FlexiWeb

We evaluate FlexiWeb extensively, in each of the scenarios described in Section 4, and discuss the results below.

Performance of FlexiWeb in controlled settings: We first evaluate FlexiWeb in the controlled network settings listed in Table 1. We calculate the percentage gains relative to the direct scheme (no proxy) for both (i) FlexiWeb and (ii) compression proxy based browsing. Figure 7a shows the average percentage gains in page load times under different network conditions. The results, shown in Figure 7a, indicate that the compression proxy degrades the performance in terms of page load times, by up to 32% as compared to direct browsing in excellent network conditions (also seen earlier). As network conditions degrade, the compression proxy outperforms direct browsing; in all other settings the reduction in object sizes (from compression) outweigh the latency overheads of longer routes and processing. We see that FlexiWeb outperforms compression proxy based browsing by up to 45% in excellent network conditions. FlexiWeb significantly gains from sending requests for small objects directly to the web server; it thereby avoids the latency overheads incurred in retrieving such objects via the proxy. FlexiWeb still downloads the large objects via the proxy, but after performing network-aware compression (large objects benefit from compression). As network conditions deteriorate, the performance gains with FlexiWeb in comparison to compression proxy based browsing diminish (only about 10 %). This is because the gains due to compression now increase and almost all objects are now retrieved via the proxy; FlexiWeb outperforms the compression based proxy here, mainly due to network-aware compression.

Next, we present the PSNR values of the transformed images with FlexiWeb and the traditional compression proxy. Figure 7b shows that the average degradation of PSNR using FlexiWeb is only $\approx$ 4 dB in comparison to compression proxy; this is in spite of FlexiWeb performing more aggressive compression during poor network conditions.

How does FlexiWeb provide performance gains?: Next, we examine the gains from the two main functions performed by FlexiWeb i.e., request splitting and network-aware compression. We analyze the web page download traces and separate the objects into those retrieved directly from the source web server and those via the proxy. We measure the gain with respect to each object's load time due to each function, in comparison to the load time of the object when fetched directly from the server. We then calculate the average gain across all objects fetched, with respect to each function. From Figure 8a, we see that in excellent and good network conditions the gains are primarily due to objects being fetched directly from the source web server; in excellent conditions, almost 70% of the gains are due to this. As network conditions degrade



(a) (b) (c)

Figure 8 : (a) Impact of request splitting and network aware compression on FlexiWeb's gains (b) Precision in predicting object sizes (c) Competitive analysis of online MCKP

the number of objects fetched via the proxy increases and the gains due to network-aware compression dominate. Since in poor network conditions, FlexiWeb fetches all objects via the proxy, all the gains are due to network-aware compression.

Accuracy of object size predictions: Next, we evaluate the accuracy with which FlexiWeb predicts the size of objects. We use the data collected at the proxy as described in Section 3. We use the 10 fold cross validation method [28]. We divide our dataset into two parts: a training set that contains 30% of the dataset, and a test set that contains the remaining 70%. We train our model using the training set and later perform prediction on the remainder (70 %) of the data which was held out. For the purposes of evaluation, we use the *precision criterion* [29], which determines the fraction of records that are correctly classified from among those that that are grouped under a classification. Specifically,

$$Precision(\%) = \left(\frac{TruePositive}{TruePositive + FalsePositive}\right) \times 100.$$

Figure 8b shows that the accuracy of predicting the size of objects based on the URL text is quite high. We can predict objects of size 0 to 1 KB and objects larger than 20 KB with more than 90% accuracy. We can predict other object sizes with at least 70% accuracy. Note that an inaccurate prediction simply causes FlexiWeb to fetch the object sub-optimally (directly instead of via the proxy or vice versa). Thus, while it slightly causes a degradation in FlexiWeb's performance, it does not disrupt the web browsing process. Note here that our prediction has high accuracy because we only seek to predict "the" range into which an object's size falls, and not the object's precise size.

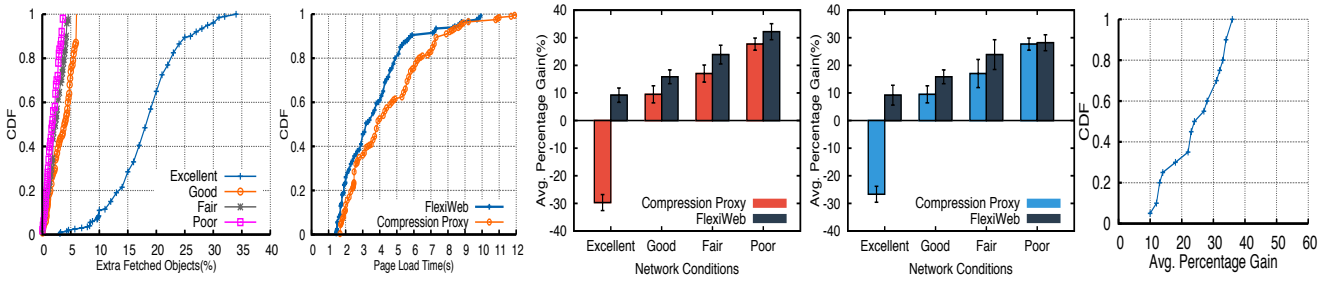
Quantifying the sub-optimality of network-aware compression: For network-aware compression, we use the modified online MCKP algorithm shown in Algorithm 1. To illustrate the difficulty of the online transformation selection problem, consider an end user with a time budget of 5 seconds to download a page. Let the download of version j of image i be represented by a utility-cost pair (u_{ij}, c_{ij}), where u_{ij} is the utility (e.g., PSNR or negative distortion) and c_{ij} is the cost. Suppose the first image has two possible versions with utility-cost pairs (1,1), (2,2), and the second image also has two versions with utility-cost pairs (2,1),(4,2). If the user requests for both the images sequentially, then the optimal (offline) solution is to choose the stream (1,1) followed by (4, 2). This gives a total utility of 5 with a total cost of 3. However, an online transformation selection algorithm must first choose a stream from $(1, 1), (2, 2)$, without knowledge of the subsequent request. Our online MCKP algorithm is likely to choose the transformation (2, 2) followed by (2, 1), resulting in sub-optimal utility.

Here, we seek to quantify the sub optimality of Algorithm 1. We select 100 web pages from the Alexa's top 500 web pages with varied number and sizes of the objects. We first download the pages using a modified version of MCKP, called OPT-MCKP. OPT-MCKP [30] is an offline version of MCKP, where the object requests are known in advance. With this information OPT-MCKP can always choose the right transformation. We repeat the same experiment using our ON-MCKP algorithm (Algorithm 1) . A popular way to evaluate online algorithms is using what is called the "Competitive Ratio" [31]. The competitive ratio of an online algorithm for an optimization problem is simply the ratio between the cost of the solution found by the algorithm and the cost of an optimal solution. Figure 8c shows the CDFs of Competitive ratio (converted to percentage) across the 100 pages; we see that 80% of the time, the page load times with ON-MCKP are within 10%-20% of that with OPT-MCKP.

Number of objects and web pages downloaded within a time budget: Next we evaluate FlexiWeb in terms of number of objects retrieved within a specified time budget of 5 seconds. We compare the number of requests satisfied by FlexiWeb in comparison with conventional compression proxy based browsing. Figure 9a depicts the CDF of the percentage of additional objects retrieved by FlexiWeb as compared to conventional compression proxy based browsing; we see that FlexiWeb fetches 17% more objects on average, within the same time budget. FlexiWeb achieves this by fetching small objects directly from the web server in excellent and good network conditions; the decrease in per-object download times allow the downloading of more objects within the page load time budget. Even in bad network conditions FlexiWeb fetches 1-2% more objects on an average than conventional compression proxy based browsing, but mainly due to network aware compression. An increase in number of additional fetched objects also translates to an increase in the number of web pages downloaded in the target budget. Figure 9b shows the CDF of page down load times of web pages using regular compression proxy and FlexiWeb. We see that FlexiWeb can download about 19% more web pages on average, within a target budget of 5 seconds.

Evaluations on AT &T and T-Mobile's networks: We evaluate FlexiWeb on real cellular networks to show that the gains seen in controlled settings also exist in real cases. From Figure 9c and 9d we see that FlexiWeb outperforms conventional compression proxy based browsing in excellent network conditions (by up to 38% with T-Mobile and 37 % with AT & T) In poor conditions it out performs the latter by up to 6% with T-Mobile and 2 % with AT & T. The gains are in between for other network conditions.

Scenarios with mobility: Next, we seek to evaluate FlexiWeb in scenarios where a passenger accesses web pages while in a mov-



(a) (b) (c) (d) (e)

Figure 9 : (a) Percentage of extra fetched objects in a time budget of 5 seconds (b) Gains in page load times (c) Performance of FlexiWeb on T-Mobile's network (d) Performance of FlexiWeb on AT&T's network (e) Gains with FlexiWeb in mobile settings

ing car. We choose 20 different paths to capture typical scenarios: (i) driving on a highway at high speeds and (ii) driving around in a city at moderate speeds. We used two mobile devices (both HTC One phones) connected to two laptops via USB, one downloading web pages via a conventional compression proxy while the other does so using FlexiWeb. We choose to download Alexa's top 50 web pages for this experiment. Both the devices are connected to AT & T's network. In addition to page load times we also keep track of signal strengths at both the devices. We start the experiments on both the phones at almost the same time to avoid variations in network conditions. We measure the average percentage gains with FlexiWeb while downloading all the web pages. In Figure 9e, we plot the CDF of this average percentage gain over all the webpages downloaded on the different paths; we see that 80% of the time, FlexiWeb provides an average percentage gain of 34 % over compression proxy based browsing, in terms of page load times. In 20% of the cases, it only provides 14% gains. Upon closer inspection, we found that some paths were experiencing excellent to good signal strength throughout while others were experiencing low signal strengths much more often than good signal strengths. It is well known that excellent/good signal strength is highly correlated with good network conditions (low RTT and loss rates, high throughputs) while the opposite is true with low/poor signal strength [32]. In excellent/good network conditions FlexiWeb significantly outperformed conventional compression proxy based browsing (since FlexiWeb retrieved most objects in the web pages directly and only few large objects via the proxy). On paths where network conditions were bad, both the conventional compression proxy based browser and FlexiWeb, fetched almost all objects via the proxy. FlexiWeb outperformed conventional compression proxy based browsing because of its network-aware compression module.

6 Related Work

Measurement studies: There is prior work on characterizing the properties of web pages, and protocols that impact mobile web browsing. The study in [33] shows that optimization of compute intensive tasks at proxies only offers marginal gains. In [34], the complexity of web sites is studied via browser-based active measurements. Other studies have evaluated the benefits of existing mechanisms for improving mobile web performance. Erman et al. [35] showed that SPDY [36], a recently proposed alternative to HTTP, does not clearly outperform HTTP over cellular networks. They identify the disharmony between TCP and cellular networks as the underlying cause. Similarly, Sivakumar et al. [37] showed that proxy-assisted thin client browsers, such as Amazon Silk [6], do not provide clear benefits in terms of page load time and energy. None of these efforts study the implications of using `Proxy Assisted` mobile web browsing in different network conditions, as we do here.

Client-based solutions: Client-side solutions have been proposed to improve mobile web performance. WebSieve [38] generates mobile-friendly websites from the original desktop versions. Zoomm [39] speeds up web page loads by parallelizing the execution of dynamic components of any web page. Adrenaline [40] parallelizes the fetching of web pages by decomposing existing web pages on the fly into loosely coupled mini pages, and loading mini pages in parallel via separate processes.

User studies: Lymberopoulos et al.'s user study [41] shows that mobile web browsing exhibits a strong spatio-temporal signature, different for every user. Based on this study, they propose a machine learning based model to accurately predict future web accesses for a user, and they use this prediction to prefetch content in a timely manner. Wang et al. [42] show that caching and prefetching provide very limited benefits for mobile web browsing, but speculative loading can decrease page load times by up to 20%.

Unlike FlexiWeb, none of the above approaches consider modifying web clients to dynamically select when to use proxies for data compaction.

Proxy-based solutions: Due to the computation and bandwidth limitations on mobile devices, researchers have proposed to offload various types of functionality from client devices to proxies. For example, Zhao et al. [3] offload execution of dynamic content to the proxy, while Cho et al. [43] propose the delegation of DNS lookups and TCP connection establishment to the proxy. Chava et al. [44] try to reduce the usage cost incurred by the end-user by computing a cost quota for each web request, and having a proxy adapt the web page accordingly. The cost quota for each web request is dynamically calculated based on the pricing plan of the user and her current data usage levels. Recently, Google [13] and Nokia [45] have incorporated data compression proxies in their mobile web browsers; these proxies (hosted in data centers) are expected to reduce cellular data usage and speed up mobile web browsing. Wang et al. [2] propose a framework that allows the execution of "any" portion of the page load process in the cloud (unlike browsers such as Opera Mini that only allow fixed parts to be executed in the cloud).

None of these proxy-based solutions are network-aware. While some of the proprietary solutions are not documented, to our knowledge, all existing systems always fetch content via the proxy, unlike FlexiWeb; further, they apply the same content compression irrespective of network conditions.

7 Conclusions

In this paper, we argue based on an in-depth measurement study that always using cloud-based middleboxes to assist mobile browsing can be detrimental to performance in terms of web page down-

load times. Our measurements reveal that the middlebox should be used only when network conditions are bad; otherwise, most objects in the web page should be directly fetched from the source web server. Based on this observation we build FlexiWeb, a framework that supports network-aware middlebox usage. In addition, FlexiWeb also performs dynamic network-aware compression to provide further performance gains. We demonstrate via extensive experiments that FlexiWeb outperforms conventional compression proxy based browsing by decreasing page load times by as much as 42 %, on average.

8 Acknowledgments

This work was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF- 09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. We would like to thank our anonymous shepherd, for her/his constructive comments and guidance through the camera ready submission process.

9 References

[1] [Online]. Available: http://bit.ly/1sgDfJ1

[2] X. S. Wang, H. Shen, and D. Wetherall, "Accelerating the mobile web with selective offloading," in *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing*, ser. MCC '13. New York, NY, USA: ACM, 2013, pp. 45–50. [Online]. Available: http://doi.acm.org/10.1145/2491266.2491275

[3] B. Zhao, B. C. Tak, and G. Cao, "Reducing the delay and power consumption of web browsing on smartphones in 3g networks," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, 2011.

[4] K. Matsudaira, "Making the mobile web faster," *Commun. ACM*, 2013.

[5] [Online]. Available: http://www.opera.com/turbo

[6] [Online]. Available: http://docs.aws.amazon.com/silk/latest/developerguide/split-arch.html

[7] [Online]. Available: https://developers.google.com/chrome/mobile/docs/data-compression

[8] J. Nielsen, *Usability Engineering*. Morgan Kaufmann, 1993.

[9] K. Zarifis, T. Flach, S. Nori, D. Choffnes, R. Govindan, E. Katz-Bassett, Z. M. Mao, and M. Welsh, "Diagnosing Path Inflation of Mobile Client Traffic," in *Passive and Active Measurement Conference (PAM '14)*, March 2014.

[10] [Online]. Available: http://info.iet.unipi.it/~luigi/dummynet/

[11] [Online]. Available: https://blog.kissmetrics.com/loading-time/?wide=1

[12] [Online]. Available: https://httpd.apache.org/docs/2.0/mod/mod_proxy.html#forwardreverse

[13] [Online]. Available: https://developers.google.com/speed/articles/spdy-for-mobile

[14] X. Xu, Y. Jiang, T. Flach, E. Katz-Bassett, D. Choffnes, and R. Govindan, "Investigating transparent web proxies in cellular networks," in *Technical report 14-944, University of Southern California*. USC, 2014.

[15] [Online]. Available: https://www.igvita.com/2013/03/07/faster-smaller-and-more-beautiful-web-with-webp/

[16] [Online]. Available: http://httparchive.org

[17] [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

[18] T. M. Mitchell, *Machine Learning*. McGraw-Hill, Inc., 1997.

[19] R. A. Berk, *Statistical Learning from a Regression Perspective*. Springer, 2008.

[20] J. MOGUL and L. BRAKMO, *Method for dynamically adjusting multimedia content of a web page by a server in accordance to network path characteristics between client and server*. U.S. Patent 6,243,761, 2001.

[21] [Online]. Available: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/optimize-encoding-and-transfer

[22] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the tcp congestion avoidance algorithm," *SIGCOMM Comput. Commun. Rev.*, 1997.

[23] Y. Zhou, D. Chakrabarty, and R. M. Lukose, "Budget constrained bidding in keyword auctions and online knapsack problems," in *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, 2008.

[24] A. G. Jiasi Chen and M. Chiang, "Qava: Quota aware video adaptation technical report," Department of Electrical Engineering Princeton University, Princeton NJ, USA, Tech. Rep., 2012.

[25] [Online]. Available: http://info.iet.unipi.it/~luigi/dummynet/

[26] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4g lte networks," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '12. ACM, 2012.

[27] [Online]. Available: https://developers.google.com/speed/pagespeed/module/install

[28] [Online]. Available: http://scikit-learn.org/stable/modules/cross_validation.html

[29] D. M. W. Powers, "Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation," Tech. Rep., 2007.

[30] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[31] S. M. LaValle, *Planning Algorithms*, 2006.

[32] Q. Xiao, K. Xu, D. Wang, L. Li, and Y. Zhong, "Concise paper: Tcp performance over mobile networks in high-speed mobility scenarios," *IEEE ICNP*, 2014.

[33] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie, "Why are web browsers slow on smartphones?" in *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '11. New York, NY, USA: ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/2184489.2184508

[34] M. Butkiewicz, H. V. Madhyastha, and V. Sekar, "Understanding website complexity: Measurements, metrics, and implications," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. New York, NY, USA: ACM, 2011, pp. 313–328. [Online]. Available: http://doi.acm.org/10.1145/2068816.2068846

[35] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan, "Towards a spdy'ier mobile web?" in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. New York, NY, USA: ACM, 2013, pp. 303–314. [Online]. Available: http://doi.acm.org/10.1145/2535372.2535399

[36] [Online]. Available: http://www.chromium.org/spdy/spdy-whitepaper

[37] A. Sivakumar, V. Gopalakrishnan, S. Lee, S. G. Rao, S. Sen, and O. Spatscheck, "Cloud is not a silver bullet: a case study of cloud-based mobile browsing," in *15th Workshop on Mobile Computing Systems and Applications, HotMobile '14, Santa Barbara, CA, USA, February 26-27, 2014*, 2014.

[38] M. Butkiewicz, Z. Wu, S. Li, P. Murali, V. Hristidis, H. V. Madhyastha, and V. Sekar, "Enabling the transition to the mobile web with websieve," in *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '13. ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2444776.2444795

[39] C. Cascaval, S. Fowler, P. Montesinos-Ortego, W. Piekarski, M. Reshadi, B. Robatmili, M. Weber, and V. Bhavsar, "Zoomm: A parallel web browser engine for multicore mobile devices," in *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '13. New York,

NY, USA: ACM, 2013, pp. 271–280. [Online]. Available: http://doi.acm.org/10.1145/2442516.2442543

[40] H. Mai, S. Tang, S. T. King, C. Cascaval, and P. Montesinos, "A case for parallelizing web pages," in *Proceedings of the 4th USENIX Conference on Hot Topics in Parallelism*, ser. HotPar'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342788.2342790

[41] D. Lymberopoulos, O. Riva, K. Strauss, A. Mittal, and A. Ntoulas, "Pocketweb: Instant web browsing for mobile devices," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVII. New York, NY, USA: ACM, 2012, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/2150976.2150978

[42] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie, "How far can client-only solutions go for mobile browser speed?" in *Proceedings of the 21st International Conference on World Wide Web*, ser. WWW '12. New York, NY, USA: ACM, 2012, pp. 31–40. [Online]. Available: http://doi.acm.org/10.1145/2187836.2187842

[43] J. Cho, J. Jeong, and E. Seo, "Twob: A two-tier web browser architecture optimized for mobile network," in *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia*, ser. MoMM '12. New York, NY, USA: ACM, 2012, pp. 267–270. [Online]. Available: http://doi.acm.org/10.1145/2428955.2429006

[44] S. Chava, R. Ennaji, J. Chen, and L. Subramanian, "Cost-aware mobile web browsing," *Pervasive Computing, IEEE*, vol. 11, no. 3, pp. 34–42, 2012.

[45] [Online]. Available: http://developer.nokia.com/Develop/Series_40/Nokia_Browser_for_Series_40/