WebPro: A Proxy-Based Approach for Low Latency Web Browsing on Mobile Devices

Ali Sehati and Majid Ghaderi
Department of Computer Science, University of Calgary
{asehati, mghaderi}@ucalgary.ca

***Abstract*—To load a webpage, a web browser first downloads the base HTML file of the page in order to discover the list of objects referenced in the page. This process takes roughly one round-trip time and constitutes a significant portion of the web browsing delay on mobile devices as wireless networks suffer from longer transmission and access delays compared to wired networks. In this work, we propose a solution for eliminating this initial delay, which is transparent to end systems, does not require modifying HTTP, and is well suited for web browsing on mobile devices. Our solution, called WebPro, relies on a network proxy that builds an up-to-date database of resource lists for the websites visited frequently by network users. The proxy resides in the wired part of the network, and hence can afford to pro-actively build and refresh the resource list database periodically. When a request for a webpage comes to the proxy, it simultaneously fetches the base HTML and all referenced objects required to render the webpage using the corresponding resource list stored in the local database. We have built a working prototype of WebPro and have conducted live experiments over WiFi and LTE networks. Our results show an average of 26% reduction in page load time for a mix of popular web sites chosen from categories such as news, sports and shopping. Moreover, in comparison to another best known proxy-based solution, WebPro provides delay reductions ranging from 5% to 51% for a variety of web sites.**

***Keywords*—Web browsing, Mobile devices, Browsing delay.**

I. Introduction

A. Motivation

Recent advances in cellular technology have given rise to the widespread adoption of mobile devices such as smartphones and tablets. Among numerous mobile apps, web browsing is still one of the most popular applications on mobile devices. Due to limited bandwidth and longer access delays in wireless networks (more specifically, cellular networks), however, web browsing is generally slower on mobile devices, which could frustrate users and lead to lost online business opportunities. For example, it is estimated that a 2 second increase in the load time of Bing's home page can reduce revenue per user by 4.3% [1].

Prior work [2], [3] has shown that different from desktop computers, there is a new set of factors causing the slow browsing experience on smartphones, which calls for solutions tailored to mobile web browsing. Some of these factors are: (1) Compared to the enterprise Ethernet typically used by desktop computers, wireless hop has longer round-trip times (RTTs) which dominate the end-to-end RTT. The long network RTT makes resource loading the bottleneck of web browsing on smartphones. On the contrary, compute intensive operations such as scripting, style formatting and layout are the bottleneck in desktop browsers. (2) Limited processing power of smartphones affects the resource loading process as it is associated with network stack and OS services. (3) Many webpages are not designed specifically for web browsing on mobile devices. For example, analysis of the traces of 25 iPhone users in [3] shows that over half of the webpages visited by smartphone users are not optimized for mobile devices or are non-mobile webpages.

978-1-4673-7113-1/15/$31.00 © 2015 IEEE

Recently, there has been a significant amount of work on reducing the latency of mobile web browsing [4]–[10]. Some of these efforts rely on modifying the web access protocol. For example, SPDY [9], a new protocol designed by Google, aims to minimize the latency of web browsing by adding request multiplexing, support for prioritization and a number of other advanced features. However, this solution requires changing the client and server side software which limits its widespread adoption. There are also prior attempts that rely on client side optimizations. This category includes solutions based on client side caching [4] and prefetching [5], [6] along with a recently proposed technique called speculative loading [7]. The short expiration times of most web objects limit the efficiency of caching techniques, while prefetching solutions suffer from wasted wireless bandwidth and battery resources that result from incorrect predictions (not a problem on wired desktop browsing). On the other hand, speculative loading technique relies on extensive changes to the mobile browser which is a hurdle to its adoption.

Other noticeable solutions are those based on network proxies. These solutions mostly try to reduce the computation time or energy consumption of web browsing by delegating some tasks involved in opening a page to a powerful entity in the network such as a cloud-based proxy [8], [11]. One of the major advantages of employing a network-based proxy solution is that a proxy can offer a better improvement by learning and exploiting the aggregate browsing behaviour of a diverse mix of mobile users which is not possible in client-only solutions.

Specifically, some network-based solutions such as VMP [8] and Opera Mini [11] aim at offloading compute-intensive operations of the page loading process to a proxy. However, it has been shown that optimizing compute-intensive operations leads to only marginal improvements in the overall page load performance [3]. Thus, other solutions such as EEP [12], [13] and PARCEL [14] try to offload *resource loading* operations to an infrastructure-based proxy in order to improve page load performance. Specifically, in these so-

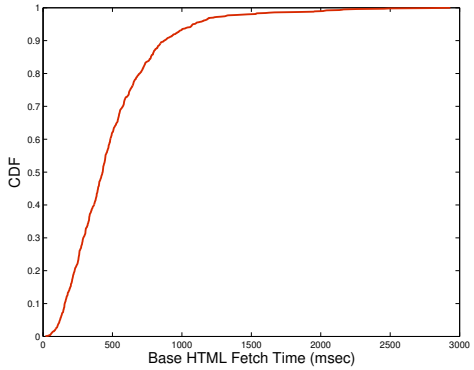


Fig. 1: CDF of the time to fetch the base HTML file for Canada's top 100 websites. In the median case, it takes 430 ms to download the base HTML file. However, this time can go beyond 1 second in some cases.

lutions, proxy retrieves the base HTML file of the page and parses it to discover referenced objects, which could be then fetched and transmitted to the client in a bundle (in order to reduce energy consumption of the mobile device).

One essential aspect of such proxy-based solutions is that the proxy can build and transmit the bundle only after it has finished downloading all the embedded objects of a page. Considering the request-response nature of the HTTP protocol, discovering the list of the referenced objects requires at least one RTT in order to fetch and parse the base HTML file. Also, one or more redirections might be involved before arriving at the base HTML file which can further delay the realization of the web objects.

To gain a better insight, we measured the latency of downloading the base HTML file for the top 100 Canadian websites [15] from a desktop computer connected to campus Ethernet. Because of the redirections, this time might be different from the RTT between our device and the corresponding web server. Figure 1 shows the cumulative distribution function of the time to fetch the base HTML file of each site. In the median case, it takes 430 ms to fetch the base HTML file. However, over 6% of the cases experience latencies beyond 1 second. Also according to the measurement results in [16], the base HTML fetch time constitutes the largest fraction of the network time for loading a page. *This implies that there is a potential for optimizing mobile browser performance by eliminating the initial fetch time*.

B. Our Work

In an effort to reduce the latency of mobile web browsing, we propose the design and implementation of a system that aims at eliminating the initial round-trip time required to fetch the base HTML file of a page. Our solution, called WebPro, is built on two cooperating proxies, one of which resides in the mobile device and the other one, remote proxy, is deployed inside the network, preferably as close to the user as possible. When a user wants to visit a page, the remote proxy will fetch all the required objects on behalf of the mobile device. After downloading all the objects, the remote proxy packs them in a bundle and pushes it to the local proxy, which will serve all browser's requests locally. In this dual proxy architecture, not only we are able to significantly reduce page load time but

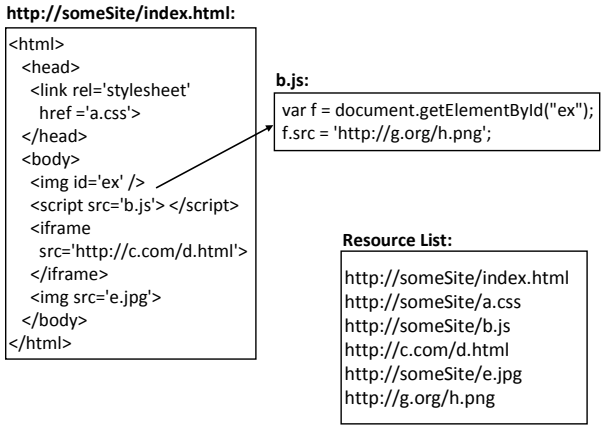


Fig. 2: Resource list for an example webpage. This webpage contains a CSS, a JavaScript, two images and an HTML iframe. Notice that the embedded JavaScript file itself refers to another image file which can be identified only after the JavaScript file is fetched and processed.

also reduce energy consumption by implementing bundling to eliminate unnecessary power state promotions and demotions in mobile's radio for each of the small objects [12], [17].

In order to fetch all the required objects of a page, the remote proxy employs the *speculative loading* technique [7]. The main idea behind this approach is to bypass the extra time for fetching and parsing the base HTML file, by using a previously recorded list of all the required objects for a webpage, hereafter called the webpage *"resource list"*. Figure 2 presents the resource list for an example webpage. We observed that the amount of change in the structure of the webpages within a few hours is relatively low and hence it should be feasible for a proxy to keep track of such changes and maintain an updated resource list of the popular pages (pages that are popular among its users). Note that maintaining the resource lists of the webpages is different from caching the actual web objects, the majority of which can not be cached or have a short expiration time [7]. Nevertheless, such legacy caching and prefetching techniques can be added to our system if desired.

Maintaining an updated set of resource lists is achieved by enhancing the remote proxy with a *profiler* that periodically visits popular websites and records their resource lists in a metadata repository. Considering that the proxy resides in the wired part of the network, it can afford to pro-actively fetch webpages and construct their resource lists for the most popular websites in the network. Such a profiling module can be easily integrated with the operational activities of high-performance dedicated middle-boxes that are already deployed by most mobile operators for caching, traffic monitoring and optimization purposes [18].

This way, the first step in loading a page at the remote proxy will become checking the metadata repository. In the case the repository contains the resource list of the page, multiple parallel connections will be used to fetch all the objects of the page from possibly different web servers. Otherwise, the remote proxy will employ a web engine to load the page by first fetching the base HTML file and then loading the discovered objects. WebPro's profiler employs a web engine to perform all the steps involved in loading a page except

rendering. This way, profiler will be able to record all the requests that result from parsing as well as script evaluations. We also implemented a filtering module to prevent profiler from recording changing URLs that result from third party advertisements and tracking systems.

We emphasize that in contrast to client-based approaches (e.g., [7]), WebPro is transparent to the end-points and does not require any changes to the clients browser. As a proxy, it exploits the common browsing activity across a diverse set of mobile users and hence provides a faster browsing experience. Moreover, in WebPro, the penalty of downloading wrong and unusable objects is negligible compared to that of client-based approaches as it resides in the wired part of the network. Thus, it can afford to pro-actively update the resource lists, which is very costly to implement on wireless clients.

We have implemented WebPro on Linux and have conducted an extensive set of measurement experiments. We believe that the common approach taken by proxy-based solutions EEP [12], [13] and PARCEL [14] is the state of the art and one of the most complete proxy-based solutions for improving web browsing performance on mobile devices[1]. We call this approach PBB (Proxy Based Browsing) and use it as benchmark to evaluate the performance of WebPro. In comparison to PBB, our scheme achieves lower page load times. Specifically in the case of a workload consisting of the 20 popular webpages from different categories, our approach loads 73% of the pages in less than two seconds while under PBB, only 28% of the pages load in that time. To the best of our knowledge, this paper is the first work to use the speculative loading approach in a network-based proxy server for improving mobile user experience.

C. Paper Organization

The rest of the paper is organized as follows. Section II introduces our proposed solution and discusses different aspects of it. Section III offers results on the performance evaluation of the system. Section IV presents a detailed review of the related work. Finally, the paper is concluded in Section V.

II. WebPro: Proxy-Based Speculative Loading

A. System Architecture

In order to eliminate the initial fetch time at the remote proxy, we take advantage of the speculative loading approach. The basic idea of speculative loading is to use the previously recorded knowledge about the structure of a website during the page load process. Our system, called *WebPro*, is depicted in Figure 3. WebPro equips the remote proxy with a profiling module that pro-actively and periodically loads webpages from a set of top visited websites and records their resource lists in a metadata repository. The list of top websites can be inferred from the web browsing behaviour of the users of the system. As will be discussed later, the memory footprint of keeping resource lists is very low, which means that the proxy can easily keep metadata for a large number (on the order of hundreds if not thousands) of websites. Consequently, the proxy does not need to employ any sophisticated mechanism

[1]The difference between EEP and PARCEL solutions is discussed in the related work section of this paper.

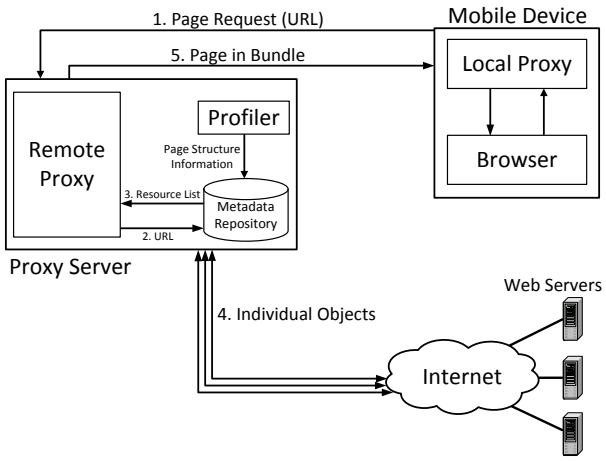


Fig. 3: High Level Architecture of WebPro.

for identifying top visited websites, which is an advantage when implementing the system.

After receiving a request to load a webpage at the remote proxy, if the resource list of that page already exists in the metadata repository, multiple parallel connections will be used to fetch the objects in the resource list. In case the remote proxy receives a request for the first time and notices the absence of the corresponding resource list, it will use the legacy approach of PBB by loading the page in a web engine. Once all the required objects of a webpage are fetched, remote proxy packs them in a bundle and sends it to the local proxy. Figure 4 shows the download pattern of this proxy-based system.

A defining feature of WebPro is that the profiler on the remote proxy can always keep a decent and fairly *recent* version of the resource lists for user requested webpages. However, the freshness of the maintained resource lists will depend on the frequency of change in the structure of the webpages. In Section III-D1, we will present measurement results indicating that on average the amount of such change within a few hours is relatively *small*. Therefore, given the abundance of the computation and communication resources at remote proxy, it should be feasible for the profiler to capture the temporal changes in resource structures by updating its metadata repository in a timely manner. Notice that doing so on the mobile device using a client-based approach is not feasible due to bandwidth and battery limitations. Also it is noteworthy that an optimized implementation of the proxy will not penalize the page load times in the case of websites with the rapidly changing structures (such as social media news feed sites), but it may not improve them either.

In order to learn and utilize the aggregate browsing activity of users in WebPro, whenever the remote proxy loads a page for the first time through the web engine, it also adds the corresponding resource list to the metadata repository. This way, the remote proxy will be able to exploit the common browsing activity across different users.

It is important to note the difference between WebPro and traditional proxy-based caching systems [19]. Those systems cache the actual content of web objects, which limits their efficiency as most web objects can not be cached or have a short expiration time [7]. However, with WebPro, the remote

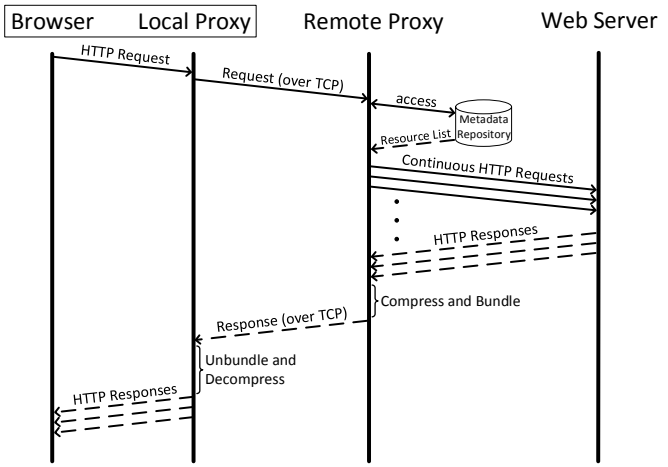


Fig. 4: Downloading a Webpage with WebPro.

proxy just keeps a list of the referenced URLs and fetches a fresh copy of the corresponding objects at each page request. Despite this difference, WebPro could be augmented with traditional caching as well in case some objects are usable because there is plenty of storage/processing capacity available at the remote proxy.

B. Circumventing Webpage Dependencies

In addition to eliminating the initial HTML fetch time, there are other reasons that lead to a reduced page load time in our approach. Those reasons are based on the fact that the activities involved in the process of loading a page are inter-dependent and can block each other [16]. For example, some of the objects may be referenced by a JavaScript or CSS file and loading those objects depends on evaluating the referencing scripts. Also, downloading and evaluating a synchronous JavaScript file blocks HTML parsing during the page load process.

The immediate implication of such dependencies is that a web engine's resource loading operations are not fully parallel and discovering web objects can be further delayed because of script evaluations and other dependencies. However, WebPro can use a previously recorded resource list and hence load all the required objects of a page without going through such dependent operations.

C. Practical Considerations

Webpage Customization: A growing number of websites provide a mobile version of their content which contains fewer and smaller images and short and concise text [2]. Also browser-dependent code in some webpages can download different set of objects for different browsers [14]. Therefore in order to comply with users' actual needs, the remote proxy needs to be aware of the client attributes such as *user-agent* and device's screen information. To this end, client provides this information to the proxy when it sends the initial request for the page. By using such information, the proxy will be able to imitate the client device when requesting objects from web servers. This way, proxy can also incorporate the resource list of the corresponding mobile website in its metadata repository.

Incremental Rendering: The bundling feature in WebPro enables the mobile device to stay in low power state during the entire time that remote proxy fetches the embedded objects of a page. While this can reduce energy consumption of mobile web browsing, it delays receiving the first set of objects by the browser which is required for the partial rendering of the page. To enable drawing intermediate displays in browser, we can envision WebPro without bundling in which the proxy forwards each object to the client as soon as it receives the object from a web server. Clearly, such a scheme has the potential to further reduce page load times with the cost of increased energy consumption (compared to WebPro with bundling).

Cost of Stale Records in Resource List: Notice that a webpage's structure can change since the last visit by the profiler which can lead to staleness of some of the records in its corresponding resource list. Considering the superior network connectivity and processing power of remote proxy we can ignore the overhead of fetching such stale objects on the proxy. On the other hand, a recent study of object sizes in the top 500 Alexa websites reveals that most of the web objects are typically small to moderate, with the median size being 18 KB [14]. Also because of selective compression component in WebPro, some of those small objects will be compressed before being included in the batch which is usually around a few megabytes for popular webpages. As a result, the overhead of stale objects for mobile device appears as a few extra kilobytes added to the size of a typically large batch file. However, the benefits of WebPro, and specifically elimination of base HTML fetch time, far outweighs such a negligible overhead. On the contrary, a client-only solution may incur significant costs in terms of energy and delay as fetching each of those stale objects can cause state promotion and demotion in the radio of the device.

Profiling Overhead: In WebPro, it is expected that usually the profiler's visit to a page will occur at an earlier time than serving a user request for that page. However in PBB, each page request triggers a new process of identifying page resources at proxy. Therefore in a setting that most webpages already have a corresponding resource list at proxy, the majority of user requests can be served without incurring any overhead due to profiling.

Handling Asynchronous JavaScript Requests: Most modern webpages use Asynchronous JavaScript requests (AJAX) to dynamically load contents such as advertisements even after the page is loaded (i.e., after the `onload` event). Usually such requests are for session dependent content and hence it would be better to fetch those objects directly from the web servers rather than the proxy. To accomplish this, the local proxy adopts a *selective forwarding* approach in which it forwards the initial page request to the remote proxy and after receiving the page batch from the remote proxy, forwards all subsequent requests to the corresponding web servers.

D. Prototype Implementation

Our current implementation of WebPro uses the Qt SDK version 5.3. Especially, QWebKit class which is a result of integration of WebKit into Qt enabled us to develop the

web engine component of the system. Also considering that for evaluating WebPro, we compare its performance with PBB, both approaches were implemented using the same Qt libraries. Here we briefly introduce the important parts of our implementation.

1) Resource Profiler: Profiler is responsible for constructing and updating webpage resource lists and storing the metadata information on the remote proxy. The Profiler is basically a WebKit-based web engine which loads webpages on demand. Note that loading a page in the profiler involves all the steps of opening a webpage except rendering. This way, we can obtain the list of all the objects whether they are resulted from parsing or from JavaScript/CSS evaluations. In particular, we intercept the network activity of this web engine and record the corresponding URLs of all the HTTP requests.

As mentioned in Section II, webpages from the set of popular websites should be loaded periodically in order to keep an up-to-date repository of resource lists on the remote proxy. This is achieved by a bash script that wakes up periodically and iteratively invokes profiler with a URL from a list of top visited websites.

A hash function of the URL determines the unique name and directory of the file that stores its resource list in the repository. In contrast to caching, storage overhead of this approach is negligible because instead of storing actual content of the objects, the proxy stores URLs of those objects. In our experiments, the total space required to store the resource lists of 20 popular websites was about 234 KB. As a result, the entire repository of resource lists can be loaded in the main memory during the operation of the proxy. Disk access is required only for backup purposes.

2) Object Bundling: We use *libtar* library to implement bundling in remote proxy and unbundling in the client proxy. In our experiments, the time spent in bundling and unbundling is negligible and has a minimal effect on page load times. For example, in the case of an experiment with `www.cnn.com` which contained 139 objects with a total size of 2.6 MB, the time spent in bundling was only 32 milliseconds.

3) Selective Compression: According to the results reported in [20], objects that have an image or video `content-type` and also most objects with binary data (e.g. app/octet-stream) already are in compressed form and there is very little room for saving. On the other hand, text files such as HTML, XML, JavaScript, and CSS can benefit greatly from compression. In line with this, the remote proxy has a selective compression component that uses the zlib [21] library to compress the body of HTTP responses with the text MIME type. We implemented bundling and selective compression in the same way for PBB as well.

4) Filtering Dynamic URLs: Many websites these days contain references to third party advertisement networks and web tracking systems. Tracking or targeted advertising is done by inclusion of a JavaScript code in a webpage that is executed when a user visits that page. Usually such JavaScript codes use random numbers or date information to create requests with dynamic URLs (i.e., different URLs over different visits). As a result, generated URL at the client's browser will be different from the recorded URL at the remote proxy. In other words, these URLs will change at every request and hence the Profiler should avoid recording them. To this end, we have implemented a module in our profiler that filters those changing URLs during the profiling period. In particular, this module detects changing URLs based on the prefixes in URLs and also URLs belonging to a blacklist [22]. To ensure a fair comparison with PBB, we also equipped PBB's web engine in the remote proxy with our filtering module.

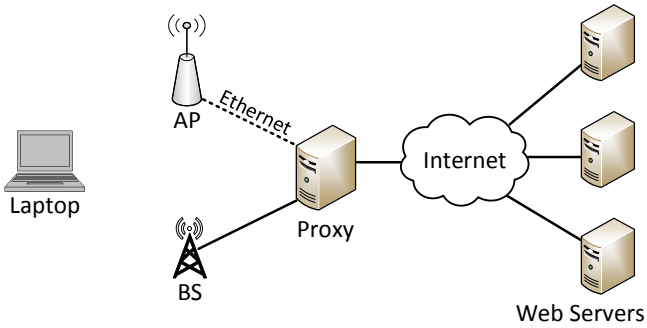


Fig. 5: Experimental Setup with the Remote Proxy.

Given that the advertisements fetched at different visits of a page can be of varying sizes and/or belong to different domains, we also incorporated the filtering module in our client side proxy to eliminate such variabilities in object load times.

III. Performance Evaluation

In this section, we use our prototype implementation to demonstrate the effectiveness of WebPro. Notice that we compare WebPro to benchmark system PBB as opposed to conventional web browsers, because the previous work [12], [14] has already shown the superior performance of PBB in comparison to traditional browsers.

A. Experimental Setup

Client Setup: Figure 5 depicts our experimental setup. We chose an ASUS UX31A laptop running Ubuntu 14.04 with built-in WiFi adapter as our mobile terminal. For cellular measurements, we equipped the laptop device with an LTE USB modem so that it can access the LTE network provided by a major Canadian cellular carrier. As mentioned in [10], the rationale for using laptops instead of smartphones is that slower processors of smartphones can influence our results on page load times. Also, by using laptops, we don't have to restrict our experiments to those websites that provide a mobile version of their site.

On the client side, we developed our own browser using QWebKit library. This way we can log detailed timing information and also clear browser's cache programmatically before each experiment. In practice, any browser can benefit from our proxy-based solution without any modifications. It only requires configuring the browser to use the local proxy.

Infrastructure Setup: We performed WLAN measurements using a *Cisco Linksys* EA2700 wireless router. The router was connected to the proxy server through the campus LAN (100 Mbps Ethernet). We also conducted cellular experiments over the LTE network at a location with good signal strength. In the cellular setting, proxy was configured with a public IP address. The average TCP throughput between the mobile

Webpage	Size (KB)	# of images	# of JS	# of CSS	# of other	Total # of objects
cnn.com	2712	90	36	1	12	139
espn.go.com	2404	76	13	3	5	97
mozilla.org	957	18	5	3	7	33
walmart.ca	3239	51	12	3	4	70
bbc.com	1599	43	24	3	3	73
ebay.ca	4078	132	4	3	9	148
shaw.ca/store	1944	26	20	2	10	58
go.com	3224	22	30	8	16	76
nytimes.com	2974	84	40	8	9	141
deviantart.com	2102	68	14	4	2	88
apple.com	1254	25	18	6	1	50
ikea.com/ca/en	2923	56	13	5	3	77
flickr.com	6736	24	4	2	8	38
ca.ign.com	2473	62	24	13	6	105
microsoft.com	1208	35	10	1	7	53
homedepot.ca	2180	32	12	5	11	60
Wikipedia Article	1932	80	9	2	1	92
cbssports.com	1535	37	26	2	5	70
tripadvisor.ca	3510	78	5	1	4	88
about.com	1437	43	5	2	3	53

TABLE I: Characteristics of the Websites Used in the Experiments.

device and the remote proxy, measured by `iperf` tool, was about 52.5 Mbps and 2.5 Mbps in WiFi and Cellular settings, respectively. Also the average ping RTT between the mobile device and the remote proxy was about 10 ms and 117 ms in WiFi and Cellular settings, respectively. The remote proxy was hosted on a fairly typical machine running Ubuntu 14.04 with no special server capability. This machine is connected to Internet using a 100 Mbps LAN connection. All experiments were conducted in a lab environment.

B. Workload Characterization

We selected 20 webpages from top Canadian websites listed on Alexa [15]. Similar to [10], we used desktop versions of these websites instead of their mobile versions because of widespread use of tablets and large screen smartphones. These webpages were chosen from different categories such as news, auction, sports, shopping, etc. Table I shows the detailed properties of our selected webpages. The average page size is 2521.05 KB and the total number of objects ranges from 33 to 148. Anything other than image, JavaScript and CSS is counted as *other*.

C. Performance Metrics Used

Page Load Time: We use page load time (PLT) as the primary indicator of user-perceived performance. In our measurements, page load time is the time elapsed between the initial page request and the time when all associated objects of a page have been downloaded and processed. This time is identified by the occurrence of the `onload` event at the browser and includes the time spent in executing CSS and synchronous JavaScript files. In the proxy-based systems discussed here, PLT consists of the following components:

1) Time to request the page from the remote proxy,
2) Time to download all the objects in the resource list (in WebPro) or the time it takes for the remote proxy's web engine to load the page (in PBB),
3) Time to receive the bundle from the remote proxy, and,
4) Time to download all the objects that are missing in the bundle until the entire webpage is loaded.

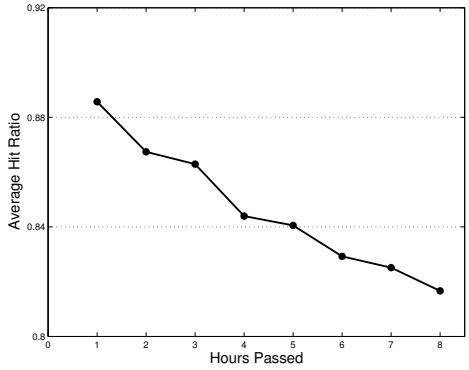


Fig. 6: Temporal Change in Webpage Structures. Drop in the value of the average hit ratio over time is an indication of the change in the structure of the webpages. However, the amount of such change is relatively low over an eight hour period.

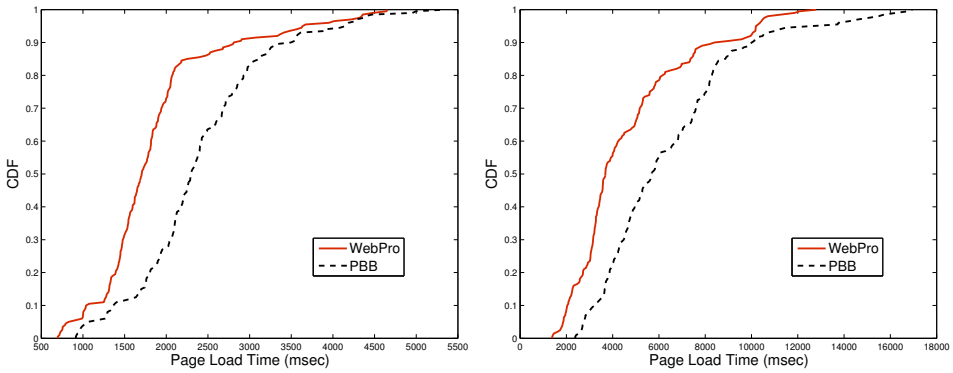
Hit Ratio: In order to capture the amount of change in webpage structures, we use the *hit ratio* metric. The hit ratio associated with a webpage's resource list is the number of objects from the resource list that are actually requested during the page load process, divided by the total number of objects in that resource list. It represents the fraction of the resource list that is still valid and accurate. A high hit ratio means that there has been little change in webpage's structure since the last time that the profiler visited the page.

D. Measurement Results

1) Change in Webpage Structures: The underlying hypothesis in WebPro is that the resource structure of a website changes less frequently than the actual content of the objects and webpages. We note that web publishers usually choose a short expiration time for web objects and also prevent web resources from being cached by using "`no-store`" in the `cache-control` HTTP header field.

In line with this, our first experiment studies the temporal changes in webpage structures. In particular, it monitors the average hit ratio of the resource lists of the websites presented in Section III-B. As mentioned in Section III-C, a decline in the value of the hit ratio associated with a resource list corresponds to change in that page's structure. Note that our selected webpages are a combination of fast changing pages such as news websites as well as stable homepages of large companies such as Apple.

We conducted five experiments over the span of five weeks, each separated by one week. In each experiment, we first constructed the resource list of the webpages and then used them to load the same pages every hour over an 8 hour period. Figure 6 plots the average hit ratio of the webpages among all the experiments as a function of the hours passed since loading the page for the first time. We see that the highest amount of hit ratio is achieved in the first hour, as expected. It can be observed that the amount of change in webpage structures over an eight hour period is relatively low. The difference between the average hit ratio in the first and eighth hours is less than 0.1 and the maximum amount of hour to hour change in the average hit ratio is about 0.02. As a result, it should be



(a) WLAN Measurements (b) Cellular Measurements

Fig. 7: Cumulative Distribution Function of Page Load Time. WebPro outperforms benchmark PBB. In the WLAN setting, under WebPro, 73% of the pages load in less than 2 seconds. However, in the PBB approach, 28% of the instances complete loading within 2 seconds. In the cellular environment, under WebPro, 78% of the page loads complete within 6 seconds while under PBB, only 55% of the pages complete loading in that time.

feasible for the remote proxy to capture the temporal changes in webpage structures by updating its resource list repository in a timely manner (every three hours in our experiments).

2) Comparison with Benchmark: Next we compare the performance of WebPro and the benchmark PBB, using the webpages presented in Section III-B. Because of the variability in load times between consecutive page visits, we performed ten back to back experiments with each page. Our experiments were conducted during quiet times and the browser's cache was cleared programmatically before each experiment. Speculative loading at the proxy involves using resource lists associated with user-requested webpages and in our experiments, the remote proxy used the resource lists that were constructed three hours before the actual measurements. Given the abundance of computation and communication resources at the proxy, it is feasible for the proxy to update its resource list repository of top visited webpages every three hours. Moreover, the results of our experiment in the previous section show that the amount of change in webpage structures within three hours is negligible.

Figure 7 represents the cumulative distribution function of page load time under these two approaches in WLAN and cellular settings. It can be seen that WebPro performs better in terms of page load time. Figure 7(a) shows that in the WLAN environment, WebPro helps up to 73% of the pages to load in less than 2 seconds, while with PBB only 28% of the instances complete loading in that time. Similarly Figure 7(b) shows that in the cellular environment, under WebPro, 78% of the pages finish loading within 6 seconds, but under PBB, only 55% of the instances finish loading in that time. In general, across all the experiments performed in the WLAN and cellular environments, our results indicate that an average of 26% reduction in page load times can be achieved by using WebPro. Figure 7(b) also confirms that in cellular networks, the same webpages experience longer load times underscoring the importance of page load time reduction in such networks.

Table II zooms into the details of these measurements by listing two of the webpages with the lowest amount of improvement and two of the pages with the highest reduction in load time. It shows that the improvements can range from 5% to 51%. Note that the variability in improvement across

Webpage	Page Load Time (ms)		Improvement
	PBB	WebPro	
www.tripadvisor.ca	3835	3623	5.5%
www.deviantart.com	10675	10070	5.7%
www.flickr.com	6717	3278	51.2%
www.about.com	4456	2166	51.4%

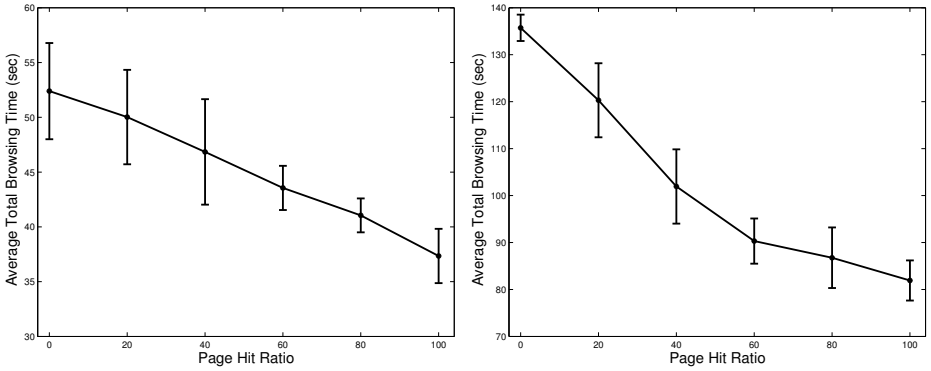
TABLE II: Improvement in Average Page Load Time.

websites results from several factors, of which we mention only a few:

- The number of domains that web objects are spread across which affects the number of unique connections required to fetch all the objects.
- The size of the website as indicated by the total number of bytes and also the number of objects.
- Website design which creates different set of dependencies between operations of page load process [16]. This can impose different orders for retrieving web objects.
- Topological proximity between the client and original web server or an edge server from content distribution networks (CDNs).

3) Effect of Page Hit Ratio: In a real deployment, it is possible that the remote proxy will not have the resource lists associated with all the user requests. In that case, it will load the page in a web engine and will send the whole page in a bundle to the client. That is, the remote proxy will employ a combination of the web engine-based and speculative loading approaches to satisfy user requests.

In light of this, our next experiment evaluates the improvements in page load time in a more realistic scenario. Here we gradually increase the hit ratio for the test webpages, that is we increase the fraction of user requests with a corresponding resource list at the proxy. To distinguish this fraction from the hit ratio metric introduced in Section III-C, we call it *page hit ratio*. Using the same webpages presented in Section III-B, we conducted five experiment runs associated with each page hit ratio. At each run, the remote proxy uses resource lists for a random set of pages that are determined based on the page hit ratio, and employs ordinary page loading for the rest of



(a) WLAN Measurements (b) Cellular Measurements

Fig. 8: Back to Back load time for 20 popular webpages as a function of page hit ratio. An increase in the page hit ratio reduces the total browsing time. In the case of WLAN and cellular measurements, there is a maximum reduction of 28% and 39%, respectively. The maximum improvements are achieved at 100% page hit ratio.

the pages. As a clarifying example, assume that the remote proxy is going to serve 20 distinct page requests. In the case of 40% page hit ratio, for each run, proxy randomly selects 8 out of the 20 pages to load using resource lists and employs web engine for loading the remaining 12 pages.

Figure 8 shows the average value for the total time to visit all 20 webpages back to back as a function of the page hit ratio. The results are represented with 95 percent confidence interval. It can be seen that a higher page hit ratio leads to a greater improvement in user's browsing experience. The upper bound of reduction in back to back page load time is 28% and 39% in the case of WLAN and cellular measurements, respectively. These upper bounds correspond to a 100% page hit ratio in both experiments.

4) Effect of Concurrent Connections: As mentioned in Section II, WebPro uses multiple concurrent connections to fetch all objects in the resource list associated with a webpage. Similarly, typical web engines use concurrent TCP connections to avoid the head-of-line blocking problem and reduce page load time [23]. However, in modern web engines there is a limit on the number of concurrent connections per domain. For example, the Chrome browser on Android mobile operating system limits the number of simultaneous connections per domain to 6. The WebKit-based web engine used in our implementation also caps the number of parallel connection per host/port combination to 6. This limitation is imposed by Qt's network access manager class and hence it is also applied to our implementation of WebPro, which uses the same class for network operations.

Our next experiment studies the effect of the number of concurrent connections on the performance of WebPro and PBB. Figure 9 shows the average page load time for a Wikipedia article page under a varying number of maximum concurrent connections. The results are averaged over 10 runs and error bars represent 95% confidence intervals. We observe a significant performance improvement in both approaches by increasing the number of concurrent connections. Specifically, increasing the concurrency limit from 2 to 8 results in 48% and 23% faster page load time in the case of WebPro and PBB, respectively. The justification for better performance of WebPro is that an increased number of concurrent connections allows more subresources to be fetched in parallel.

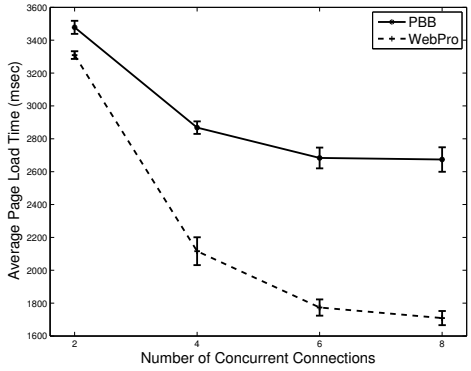


Fig. 9: Average Page Load Time for a Wikipedia article page as a function of the number of parallel connections. We see that increasing the concurrency reduces page load time. The benefits are greater for WebPro as it can fetch more subresources concurrently.

We also found that increasing the concurrency limit beyond 6 leads to marginal improvements in page load times. This can be due to several factors creating a bottleneck for browsing performance. For example, by increasing the concurrency beyond a limit, each connection obtains less bandwidth, which results in longer delays when downloading objects. On the other hand, high concurrency requires more TCP connection states and buffers to be maintained at the remote proxy and hence increases the processing overhead on the proxy.

Figure 9 also shows that WebPro benefits more from increased concurrency, compared to the PBB approach. In particular, a 4% difference in page load time between two approaches reaches 36%, by increasing the concurrency restriction from 2 to 8. This is due to the fact that processing tasks such as JavaScript evaluation can serialize the page load process in PBB's web engine. However, by using the resource list of a webpage, WebPro can utilize the full potential of concurrent connections.

5) Effect of Network Delay: As mentioned in Section II, WebPro improves the performance of mobile web browsing by eliminating the initial RTT required to fetch the base HTML file of a webpage. The length of this time can vary depending on the distance between the remote proxy and web servers, and the type of networks involved. Other factors such as queuing delays or congested links can also contribute to the variability

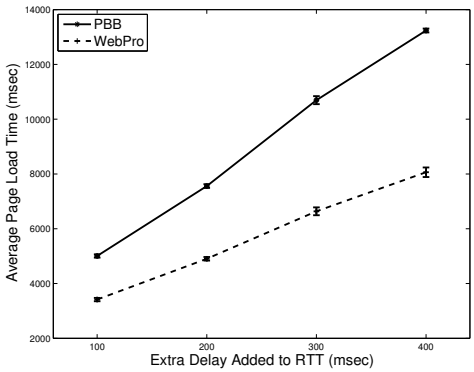


Fig. 10: Average Page Load Time for a Wikipedia article page as a function of network delay. We see that higher RTT values lead to higher page load times. By increasing RTT, PBB incurs higher latencies compared to WebPro.

in the end-to-end delay between the proxy and web servers. In order to study the impact of network delay on page load time, we conducted a set of experiments by artificially controlling the amount of packet delay in our tests.

We used the *dummynet* network emulator [24] to inject extra delay between the remote proxy and web servers. Specifically, we added 100, 200, 300 and 400 ms extra delay to the round trip time between our device and the servers hosting the objects referenced in a Wikipedia article page. Figure 10 shows the average page load time under WebPro and PBB as a function of the network delay. The results represent the average of ten runs along with the 95% confidence intervals. It is observed that increasing RTT (i.e., network delay) leads to a slower browsing experience in both approaches. In particular, raising the amount of injected delay from 100 ms to 400 ms increases the average page load time by 136% and 164% in WebPro and PBB, respectively.

Figure 10 also shows that in larger RTTs the amount of savings achieved by WebPro increases. This can be explained by the notions of *dependency graph* and *critical path*, introduced in [16]. The dependency graph of a webpage is a directed acyclic graph with load process activities as nodes. The edges of this graph represent the dependencies between those activities. Given that each node is associated with the duration of completing its corresponding activity, the simplest form of critical path is defined as the longest path in the dependency graph. Since in PBB, the extra delay impacts all the resource loading nodes of a critical path, the overall page load time will be affected by the aggregate of those extra delays. However, WebPro avoids traversing the critical path by downloading the objects in the resource list of a page.

We note that the last two experiments study the behaviour of WebPro and the PBB under different system conditions, i.e. concurrency limit and network delay. Given that these conditions only affect the wired part of the network between the remote proxy and web servers, we only presented the experimental results under WLAN setting. Similar behaviour is expected in the cellular environment.

IV. Related Work

There is a large body of work on improving the performance, energy usage and wireless data consumption of web browsing on mobile devices. Here, we classify the work that is most relevant to ours.

Client-based Solutions. Traditional solutions based on client-side caching and prefetching fall in this category. As an example, the authors of [6] used a machine learning approach to model the web browsing signature for each individual user. This model can predict the future web access patterns, enabling a prefetching scheme to download web content before the actual user request. Wang *et al.* [7] used a web dataset to assess the effectiveness of client-side caching and prefetching in improving mobile browser speed. Their results indicate that there is a limited efficiency gain due to caching and prefetching when it comes to mobile web browsing. Consequently, they proposed a new technique called speculative loading which predicts and loads the required resources of a page in parallel with the base HTML file of the page. However, their approach requires changing the mobile browser extensively, which limits its practical feasibility.

One major drawback of the client-only solutions is that any incorrect prediction can lead to downloading data that the user may never use. While not a significant problem in wired networks, this can waste the scarce resources of mobile battery and wireless bandwidth and hence harm user's experience rather than improving it in wireless networks. In order to accurately balance costs and benefits of prefetching, authors of [5] proposed a system level solution that provides explicit prefetching support to mobile applications. However, their solution requires extensive modifications of the existing applications. Another drawback of client-only solutions is that they cannot observe the aggregate behaviour of users and benefit from their common browsing activities which is at the heart of traditional caching techniques.

Protocol-based Solutions. SPDY by Google [9] is a new application layer protocol primarily designed for reducing latency of web browsing. SPDY multiplexes multiple data streams over a single TCP connection. It also enables unsolicited push of embedded objects by web servers which can speed up the resource loading process in the browser. Combined with other advanced features, SPDY can be very effective in reducing the web browsing delay [9]. However this protocol relies on web server support and given that only 3.4% of all websites support SPDY [25], its impact so far has been rather limited. Also the next generation HTTP protocol, HTTP/2, evolved from SPDY and currently is in the process of being standardized by IETF [26].

Infrastructure-based Solutions. Some of the previous work in this category has tried to improve the energy efficiency of mobile web browsing. Aggrawal *et. al.* [27] proposed a cloud-based proxy system to reduce the energy consumption of the smartphone's data communication by employing aggregation, redundancy elimination and opportunistic scheduling when downloading web objects from the network. Wang *et. al.* [12], [13] presented a dual-proxy architecture called EEP that utilizes bundling and compression to reduce the energy consumption of web browsing in 3G/WLAN networks.

There are also studies that try to reduce both power consumption and delay of mobile web browsing. For example, Zhao *et al.* [8] proposed a Virtual-Machine based architecture

in which a VM-hosted proxy performs all the computation expensive tasks of mobile browsing and sends a screen copy of the rendered page to the smartphone. However, as mentioned in [3], offloading compute-intensive operations when loading a webpage has negligible benefits compared to the improvements resulting from resource loading optimizations. Also Sivakumar *et. al.* [14] proposed PARCEL which uses the same architecture as in EEP while providing the proxy with the flexibility to optimize the bundle size in a cellular friendly manner.

Finally, this category includes studies with the goal of reducing latency of mobile web browsing. Some of them achieve this goal by reducing the amount of data transmitted because of web browsing [11], [28], while others employ solutions that directly deal with network access delay [29]. For example, Opera Mini [11] and Amazon Silk [28] are cloud-based browsers that aim to reduce both latency and data usage of mobile web browsing by offloading portions of the page load process to the cloud-based proxies. However, a recent study by Sivakumar *et. al.* [30] shows that cloud-based browsers are not always superior in terms of responsiveness and energy consumption, especially in dealing with client interactions.

The closest research to ours is EEP by Wang *et. al.* [12], [13] and PARCEL by Sivakumar *et. al.* [14]. While their focus is on reducing the energy consumption by batching and compression [12]–[14], our main goal is latency reduction using the speculative loading technique. These solutions are orthogonal to each other and can be used in combination to create a solution that is both energy efficient and low latency.

V. Conclusion

In this paper, we proposed a system called WebPro for reducing the latency of mobile web browsing. WebPro is designed to eliminate the initial round-trip time required to discover the list of objects referenced in a webpage by using a previously recorded resource list of the webpage. Using measurements involving real world websites, we showed that within a few hours, the amount of change in the structure of webpages is relatively low, and hence it is feasible for WebPro to maintain an updated resource list of the popular websites. We performed a detailed set of experiments to assess the efficiency of a prototype implementation of the system. Our results indicate that WebPro outperforms state-of-the-art in terms of the page load time, though the amount of improvement varies between webpages. This work is a step toward optimizing the existing wireless infrastructure and mobile applications for an improved quality of experience. In the future, we plan to incorporate opportunistic scheduling in WebPro to further reduce the transmission energy consumption on the mobile device during web browsing.

References

[1] *S. Souders. Velocity and the bottom line*, (accessed February 14, 2015), http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html.

[2] J. Huang *et al.*, "Anatomizing application performance differences on smartphones," in *Proc. ACM MobiSys*, 2010, pp. 165–178.

[3] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie, "Why are web browsers slow on smartphones?" in *Proc. ACM HotMobile*, 2011, pp. 91–96.

[4] F. Qian *et al.*, "Web caching on smartphones: ideal vs. reality," in *Proc. ACM MobiSys*, 2012, pp. 127–140.

[5] B. D. Higgins, J. Flinn, T. J. Giuli, B. Noble, C. Peplin, and D. Watson, "Informed mobile prefetching," in *Proc. ACM MobiSys*, 2012, pp. 155–168.

[6] D. Lymberopoulos, O. Riva, K. Strauss, A. Mittal, and A. Ntoulas, "Pocketweb: Instant web browsing for mobile devices," in *Proc. ACM ASPLOS*, 2012, pp. 1–12.

[7] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie, "How far can client-only solutions go for mobile browser speed?" in *Proc. ACM WWW*, 2012, pp. 31–40.

[8] B. Zhao, B. C. Tak, and G. Cao, "Reducing the delay and power consumption of web browsing on smartphones in 3G networks," in *Proc. IEEE ICDCS*, 2011, pp. 413–422.

[9] *SPDY: An experimental protocol for a faster web*, (accessed November 6, 2014), http://www.chromium.org/spdy/spdy-whitepaper.

[10] J. Erman, V. Gopalakrishnan, R. Jana, and K. Ramakrishnan, "Towards a SPDY'ier mobile web," in *Proc. ACM CoNEXT*, 2013, pp. 303–314.

[11] *Opera mini browser*, (accessed November 8, 2014), http://www.opera.com/mobile.

[12] L. Wang and J. Manner, "Energy-efficient mobile web in a bundle," *Computer Networks*, vol. 57, no. 17, pp. 3581–3600, 2013.

[13] L. Wang, B. Yu, and J. Manner, "Proxies for energy-efficient web access revisited," in *Proc. ACM e-Energy*, 2011, pp. 55–58.

[14] A. Sivakumar, S. Puzhavakath Narayanan, V. Gopalakrishnan, S. Lee, S. Rao, and S. Sen, "PARCEL: Proxy assisted browsing in cellular networks for energy and latency reduction," in *Proc. ACM CoNEXT*, 2014, pp. 325–336.

[15] *Alexa Internet Inc. "Top Sites in Canada"*, (accessed November 8, 2014), http://www.alexa.com/topsites/countries/CA.

[16] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "Demystifying page load performance with wprof," in *Proc. USENIX NSDI*, 2013.

[17] A. Gerber, S. Sen, and O. Spatscheck, "A call for more energy-efficient apps," *AT&T Labs Research*, 2011.

[18] R. Mahindra, H. Viswanathan, K. Sundaresan, M. Y. Arslan, and S. Rangarajan, "A practical traffic management system for integrated LTE-WiFi networks," in *Proc. ACM MobiCom*, 2014, pp. 189–200.

[19] G. Barish and K. Obraczka, "World Wide Web Caching: Trends and Techniques," *IEEE Communications Magazine*, vol. 38, pp. 178–184, 2000.

[20] F. Qian, J. Huang, J. Erman, Z. M. Mao, S. Sen, and O. Spatscheck, "How to reduce smartphone traffic volume by 30%?" in *Proc. PAM*, 2013, pp. 42–52.

[21] L. Deutsch and J. Gailly, "Rfc 1950: Zlib compressed data format specification version 3.3," *IETF*, May 1996.

[22] J. van den Brande and A. Pras, "The costs of web advertisements while mobile browsing," in *Information and Communication Technologies*. Springer, 2012, pp. 412–422.

[23] B. Thomas, R. Jurdak, and I. Atkinson, "SPDYing up the web," *Communications of the ACM*, vol. 55, no. 12, pp. 64–73, 2012.

[24] M. Carbone and L. Rizzo, "Dummynet revisited," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 12–20, 2010.

[25] *Usage Statistics of SPDY for Websites*, (accessed February 3, 2015), http://w3techs.com/technologies/details/ce-spdy/all/all.

[26] D. Stenberg, "HTTP2 explained," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 120–128, 2014.

[27] B. Aggarwal, P. Chitnis, A. Dey, K. Jain, V. Navda, V. N. Padmanabhan, R. Ramjee, A. Schulman, and N. Spring, "Stratus: energy-efficient mobile communication using cloud support," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 477–478, 2010.

[28] *Amazon silk browser*, (accessed November 8, 2014), http://amazonsilk.wordpress.com/.

[29] R. Chakravorty, A. Clark, and I. Pratt, "Optimizing web delivery over wireless links: design, implementation, and experiences," *Selected Areas in Communications, IEEE Journal on*, vol. 23, no. 2, pp. 402–416, 2005.

[30] A. Sivakumar, V. Gopalakrishnan, S. Lee, S. Rao, S. Sen, and O. Spatscheck, "Cloud is not a silver bullet: A case study of cloud-based mobile browsing," in *Proc. ACM HotMobile*, 2014.