Computer Networks 106 (2016) 134–150

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Network assisted latency reduction for mobile web browsing☆

Ali Sehati*, Majid Ghaderi

Department of Computer Science, University of Calgary, Calgary, AB, Canada T2N 1N4

ARTICLE INFO

Article history:
Received 15 January 2016
Revised 24 May 2016
Accepted 21 June 2016
Available online 23 June 2016

Keywords:
Web browsing
Mobile devices
Browsing delay

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 also show that the benefits of WebPro become more significant by increasing the complexity of webpages as it is able to circumvent the inter-object dependencies in a webpage. We have built a working prototype of WebPro and have used real-world traffic traces along with live experiments over Wi-Fi and LTE networks to assess its performance. 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.

1. Introduction

1.1. 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% [2].

Prior work [3,4] 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 access delays which dominate the end-to-end round trip time (RTT) and consequently result in longer RTTs. 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 [4] shows that over half of the webpages visited by smartphone users are not optimized for mobile devices or are non-mobile webpages.

Recently, there has been a significant amount of work on reducing the latency of mobile web browsing [5–12]. Some of these efforts rely on modifying the web access protocol. For example, SPDY [9], a new protocol designed by Google, aims to minimize

☆ A preliminary version of this work appeared in the IEEE/ACM International Symposium on Quality of Service (IWQoS) 2015 [1]

* Corresponding author.
E-mail addresses: asehati@ucalgary.ca, a.sehati@gmail.com (A. Sehati), mghaderi@ucalgary.ca (M. Ghaderi).

http://dx.doi.org/10.1016/j.comnet.2016.06.026
1389-1286/

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 [13] 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 [8,14,15]. 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 [15] 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 [4]. Thus, other solutions such as EEP [16,17] and PARCEL [18] try to offload *resource loading* operations to a network-based proxy in order to improve page load performance. Specifically, in these solutions, 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 [19] 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. Fig. 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 [20], 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.*

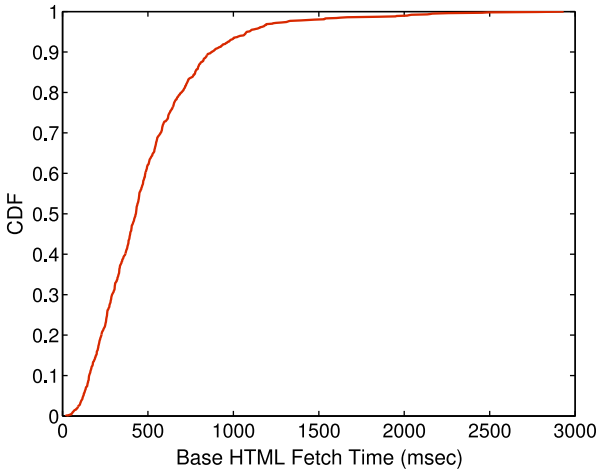


Fig. 1. CDF of the time to fetch the base HTML file for Canada's top 100 websites. In the median case, it takes 430 ms to download the base HTML file. However, this time can go beyond 1 second in some cases.

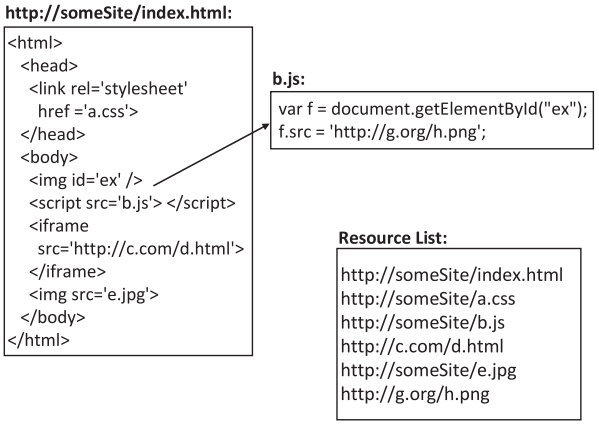


Fig. 2. Resource list for an example webpage. This webpage contains a CSS, a JavaScript, two images and an HTML iframe. Notice that the embedded JavaScript file itself refers to another image file which can be identified only after the JavaScript file is fetched and processed.

1.2. 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 (see Fig. 3). 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 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 [16,21].

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"*. Fig. 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.

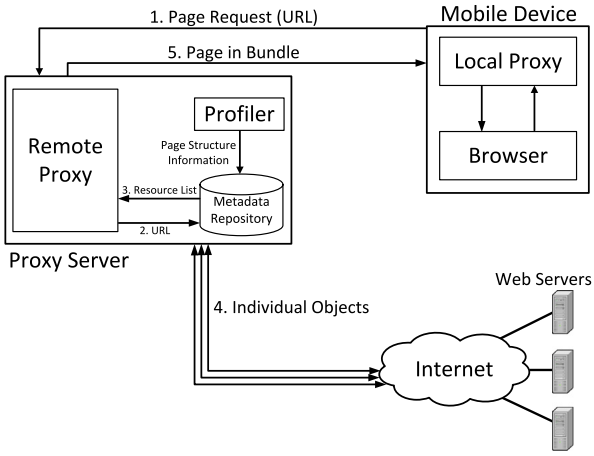


Fig. 3. High level architecture of WebPro.

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 [22].

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.

In order for the metadata repository to contain the resource lists of the majority of the requests, profiler should employ an effective mechanism to identify popular URLs in the network. Considering the flow of URLs into the proxy as a data stream, we exploit a well-known algorithm in data mining community, *space saving*, for identifying the popular URLs at the proxy. Our experiments using traffic traces collected from University of Calgary's Internet link shows that maintaining a metadata repository for the top-1000 URLs in the network, allows a timely update of their resource lists by the profiler while providing a high hit ratio to the user requests.

Using resource lists at the proxy also enables WebPro to avoid going through the iterative process of exploring objects in a webpage. In other words, it enables WebPro to break the inter-object dependencies in a webpage. The most common form of such dependencies happens when an embedded object itself refers to another object, similar to the example in Fig. 2. Accordingly, our experiments using carefully designed synthetic webpages reveals that the benefits of WebPro will extend as the number of dependencies in a webpage increases.

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 client's 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 (in terms of wireless bandwidth usage and mobile battery consumption) 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 [16,17] and PARCEL [18] 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 dual proxy architecture for improving mobile user experience.

1.3. Paper organization

The rest of the paper is organized as follows. Section 2 introduces our proposed solution and discusses different aspects of it. Section 3 offers results on the performance evaluation of the system. Section 5 presents a detailed review of the related work. Finally, the paper is concluded in Section 6.

2. WebPro: Proxy-based speculative loading

2.1. 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 Fig. 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 thousands) of websites.

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, the remote proxy packs them in a bundle and sends the bundle to the local proxy. Fig. 4 shows the download pattern of WebPro and PBB. Notice that both WebPro and PBB bundle objects when transferring them from proxy to the client. The flow chart in Fig. 5 shows the operations performed at WebPro's remote proxy for serving a user request.

A defining feature of WebPro is that the profiler on the remote proxy can always keep a fairly *recent* version of the resource lists

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

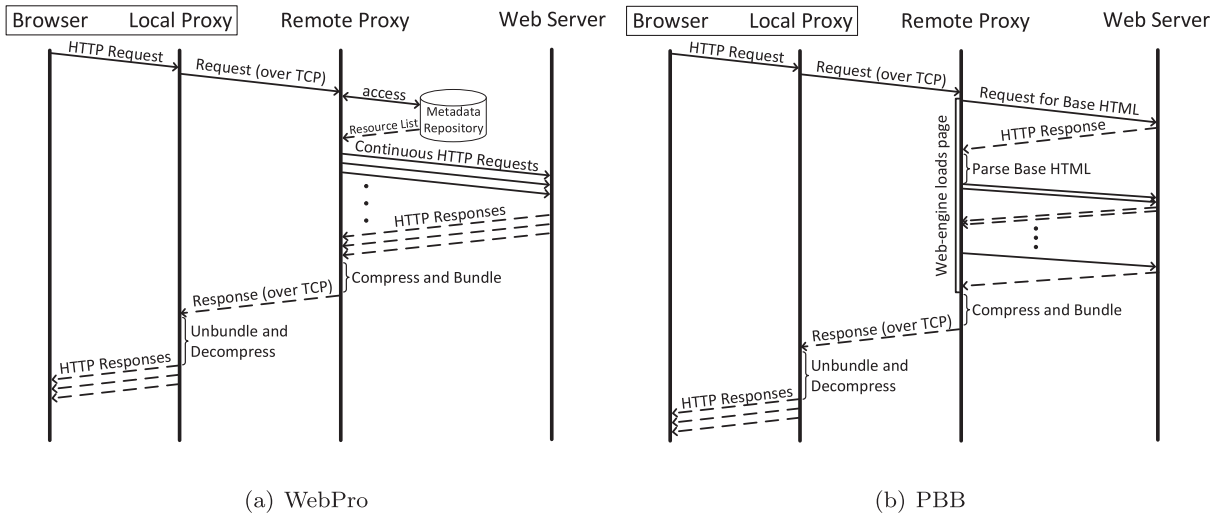


Fig. 4. Downloading a Webpage with WebPro (a) and PBB (b).

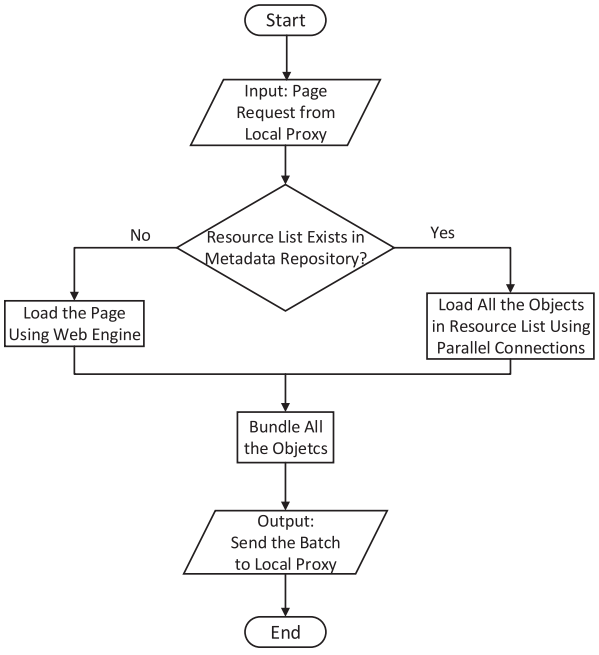


Fig. 5. Flow chart of operations performed at remote proxy.

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 3.4.1, 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 the 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 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 [23]. 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 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.

2.2. 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 [20]. 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.

2.3. Identifying popular websites

As implied in previous sections, WebPro's profiler keeps a list of popular URLs in the network and by periodically loading those URLs, updates its metadata repository. It is clear that the benefits of WebPro will increase if the profiler maintains a URL list that achieves higher hit ratios for page requests in the network. The reason is that the presence of a user requested URL in the list of top URLs (hit occurrence) means that the proxy already has the updated resource list of that page in its metadata repository and hence can load the page faster. A simple approach for constructing

Table 1
Hit ratio and top URL set size that result from running the simple algorithm over University of Calgary's HTTP traces collected between May 1, 2015 and May 6, 2015.

Day number	Hit ratio	# of Distinct URLs
1	93.4%	18,482
2	95%	24,725
3	97%	28,156
4	94.6%	38,235
5	95.3%	46,565
6	96%	54,040

such a list of URLs would be keeping a SET data structure and adding all the URLs to the SET upon their arrival at the proxy.

To gain a better insight, we applied this approach to network traffic traces that were collected from University of Calgary's Internet link. These traces contain summary information of all the HTTP transactions and are recorded by Bro [24], an open source Network Intrusion Detection System. We wrote an AWK script to extract the URLs of the landing pages from HTTP traces of six consecutive days (May 1, 2015 to May 6, 2015). We ran an offline analysis on the extracted traces and constructed the URL set by adding elements to it using the set union operation. Before adding a URL to the list, a dictionary lookup operation is performed to test whether it already exists in the list, and if so, a hit counter is incremented. In this experiment, we kept adding URLs to the same list for the entire six day period. For each day, we compute the hit ratio as the number of hits on that day divided by the total number of requests during that day. We also record the size of the SET at the end of each day which is the number of distinct URLs observed by that time. The results are presented in Table 1. It can be seen that in all days a high hit ratio of at least 93% is achieved. Also notice that the number of distinct URLs is $\sim 18K$ in the first day and reaches $\sim 54K$ by the end of the sixth day.

Notice that storing the resource lists of such a large number of URLs is feasible because of small size of resource lists (on average 11.7 KB for a page in our top 20 page selection) and abundance of storage space in the remote proxy. However, considering that on average a page can take about 6 seconds to load on a Desktop computer [25], it would take about 33 hours to visit 20,000 pages back to back and update their resource lists in metadata repository. Such a long update interval can compromise the freshness of resource lists for some of the fast changing websites.

Notice that the above problem stems from a large number of page requests constantly flowing into the proxy. As a result, we can cast it as an instance of identifying most popular k items in a data stream. Rather than storing all the distinct URLs in a set, in this setting we are interested in an online algorithm that accurately reports top-k elements of a data stream by taking only a single pass over data. This is one of the well-studied problems in data mining community. For instance, authors of [26] presented a survey of some of the most popular algorithms in this area and conducted experiments to compare the performance of these algorithms. Their findings indicate that for insert-only streams,[2] the *space saving* algorithm [27] performs better in terms of precision,[3] recall,[4] used space and update speed. As a result, we select this algorithm for identifying top-k URLs in the stream of URLs arriving at the proxy.

[2] As opposed to streams where elements can be both inserted and deleted
[3] Proportion of the items reported by the algorithm that are true frequent items
[4] Proportion of the true frequent items that are reported by the algorithm

2.3.1. Space saving algorithm

In order to identify top-k elements of a data stream, the space saving algorithm maintains k elements with their associated counters. Upon arrival of a new URL at the proxy, in case it is already monitored (exists in the list), we just increment its associated counter. Otherwise, if the URL list is not full, we insert the URL into the list and set its counter to 1. If the URL list is full and the URL does not match a monitored item, we find the URL with the least count, *min*, and replace it with the new URL. Finally, $min+1$ is assigned to the count of the new URL. The authors of [27] proposed a data structure called *Stream-Summary* that ensures constant time for finding the minimum element. Also incrementing counters in Stream–Summary can be performed using $O(1)$ pointer operations.

2.4. 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 [3]. Also browser-dependent code in some webpages can download different set of objects for different browsers [18]. 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, the 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 the 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 a 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 at the cost of increased energy consumption (compared to WebPro with bundling). We note that implementing WebPro without bundling can benefit from native Virtual Private Network (VPN) support in vast majority of modern mobile devices. Similar to *Meddle* proposed in [28], in this setting, a VPN tunnel can be used to direct all the Internet traffic of mobile device to the remote proxy. Such a VPN-based approach will eliminate the need to deploy a local proxy component on the mobile device.

Cost of stale records in resource list: 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 the 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 [18]. 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 pro-

motion and demotion in the radio of the device, which is a well-known cause of battery drainage on wireless devices.

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 (the solution proposed in [16–18]), each page request triggers a new process of identifying page resources at the proxy. Therefore, in a setting that most webpages already have a corresponding resource list at the 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 objects not present in its cache to the corresponding web servers.

2.5. Prototype implementation

Our current implementation of WebPro uses the Qt SDK version 5.3. Specifically, 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.

2.5.1. 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 2, 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.

2.5.2. Object bundling

We use *libtar* library to implement bundling in the 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.

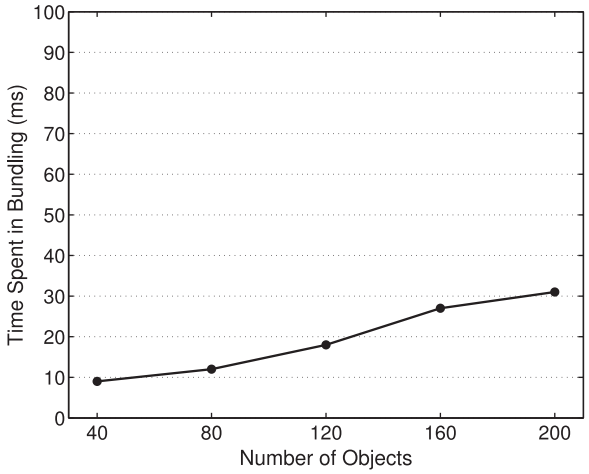


Fig. 6. Bundling performance.

To study the effect of the number of objects on the performance of bundling, we measured the time spent in bundling for different numbers of objects, all with size 20KB (the average object size in a modern webpage). Fig. 6 depicts the bundling performance as a function of object numbers. It can be observed that even for the case of 200 objects, the bundling time is negligible compared to the overall page load time (in the order of tens of seconds). It is also noteworthy that the timing values reported here are obtained using a typical machine in our lab, while it is expected that in a real-world deployment, the proxy will be hosted on a more powerful computer(s) with dedicated hardware. With the widespread adoption of cloud computing, we can also envision hosting remote proxy in a cloud platform which automatically scales up its processing power to handle an increase in workload.

2.5.3. Selective compression

According to the results reported in [29], 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 additional 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 [30] 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.

2.5.4. 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, the URL generated 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 [31]. To ensure a fair comparison with PBB, we also equipped PBB's web engine in the remote proxy with our filtering module.

Given that the advertisements fetched at different visits to a page can be of varying sizes and/or belong to different domains,

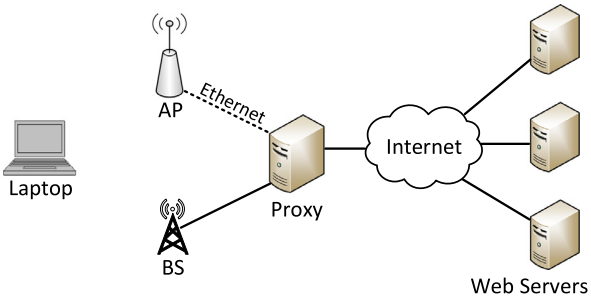


Fig. 7. Experimental setup with the remote proxy.

we also incorporated the filtering module in our client side proxy to eliminate such variabilities in object load times.

2.5.5. Local proxy

The local proxy is developed using QT's networking API (QTcpSocket and QTcpServer) and acts as a server to the mobile browser while acting as a client to the remote proxy. It also uses the same libraries discussed above for unbundling and decompression. Specifically, local proxy passes the first request of a page to the remote proxy and after receiving the page bundle, responds to the browser with the appropriate object while caching the rest of the objects in the bundle. For all the subsequent requests, local proxy will try to load the object from its cache if available, otherwise will forward the request to the corresponding web server.

3. 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 [16,18] has already shown the superior performance of PBB in comparison to traditional browsers.

3.1. Experimental setup

Client setup: Fig. 7 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, the proxy was configured with a public IP address. The average TCP throughput between the mobile 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.

3.2. Workload characterization

We selected 20 webpages from the top Canadian websites listed on Alexa [19]. 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 2 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*.

3.3. 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.

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.

3.4. Measurement results

3.4.1. 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 3.2. As mentioned in Section 3.3, 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

Table 2
Characteristics of the websites used in the experiments.

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

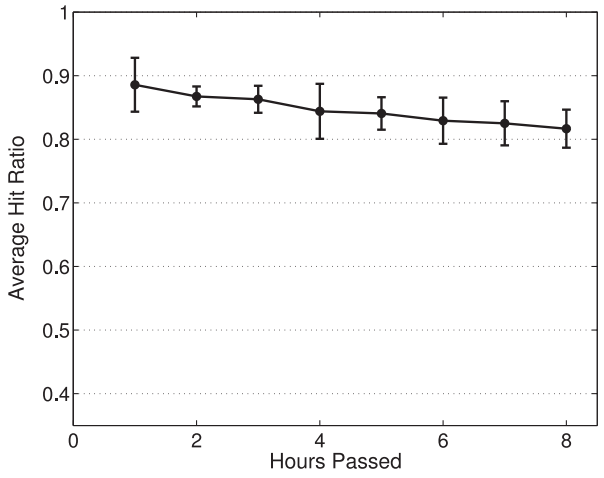


Fig. 8. Temporal change in webpage structures. Drop in the value of the average hit ratio over time is an indication of the change in the structure of the webpages. However, the amount of such change is relatively low over an eight hour period.

load the same pages every hour over an 8 hour period. Fig. 8 plots the average hit ratio and 95% confidence intervals 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 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). Notice that our selected three hour update interval is even less than the 4 hour update interval proposed by [32] for capturing the flux in dependency structure of webpages.

3.4.2. Comparison with benchmark

Next, we compare the performance of WebPro and the benchmark PBB, using the webpages presented in Section 3.2. 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.

Table 3
Improvement in average page load time.

Webpage	Page load time (ms)		Improvement
	PBB	WebPro	
www.tripadvisor.ca	3835	3623	5.5%
www.deviantart.com	10,675	10,070	5.7%
www.flickr.com	6717	3278	51.2%
www.about.com	4456	2166	51.4%

Fig. 9 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. Fig. 9(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 Fig. 9(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. Fig. 9(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 3 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 websites results from several factors, of which we mention only a few:

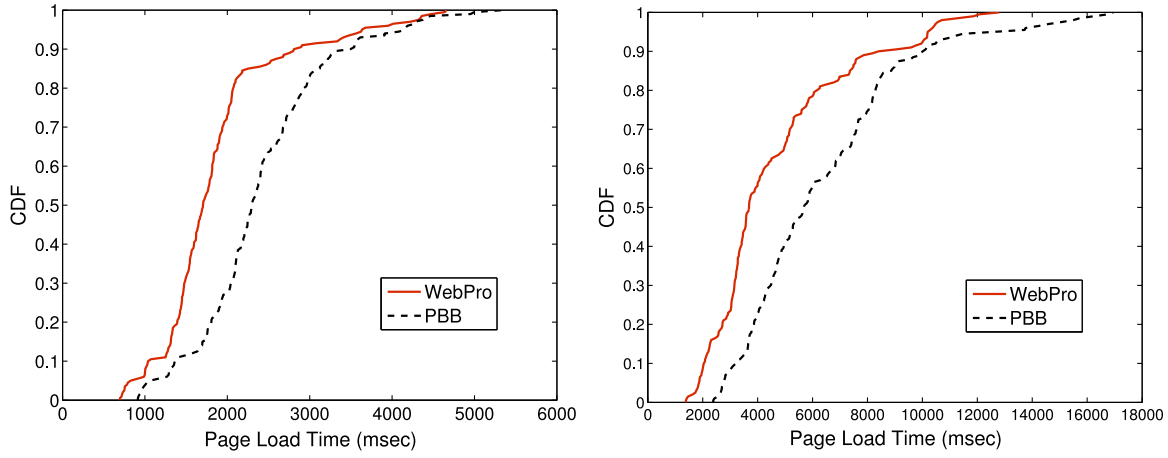


Fig. 9. Cumulative distribution function of page load time. WebPro outperforms benchmark PBB. In the WLAN setting, under WebPro, 73% of the pages load in less than 2 seconds. However, in the PBB approach, 28% of the instances complete loading within 2 seconds. In the cellular environment, under WebPro, 78% of the page loads complete within 6 seconds while under PBB, only 55% of the pages complete loading in that time.

- 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 the page load process [20]. 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).

3.4.3. 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.

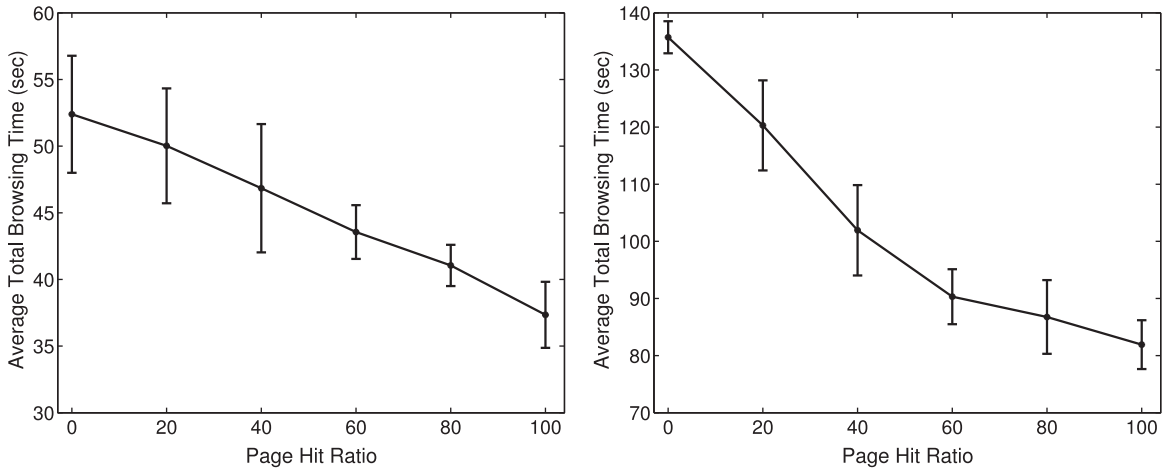
In light of this, our next experiment evaluates the improvements in page load time in a more realistic scenario. Here we gradually increase the hit ratio for the test webpages, that is we increase the fraction of user requests with a corresponding resource list at the proxy. To distinguish this fraction from the hit ratio metric introduced in Section 3.3, we call it *page hit ratio*. Using the same webpages presented in Section 3.2, 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 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.

Fig. 10 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 intervals. 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.

From Fig. 10 we can see that the amount of improvement gained from using resource lists depends on the page hit ratio. We observed in Section 2.3 that adding all the new URLs to profiler's URL list can result in high page hit ratios that are close to 100%. However, this may lead to a long update interval in profiler and endanger freshness of resource lists. The immediate alternative is to keep a relatively small summary of data stream (stream of URLs) rather than storing all of them. As discussed before, we propose using the space saving algorithm to identify top-k popular URLs of the stream.

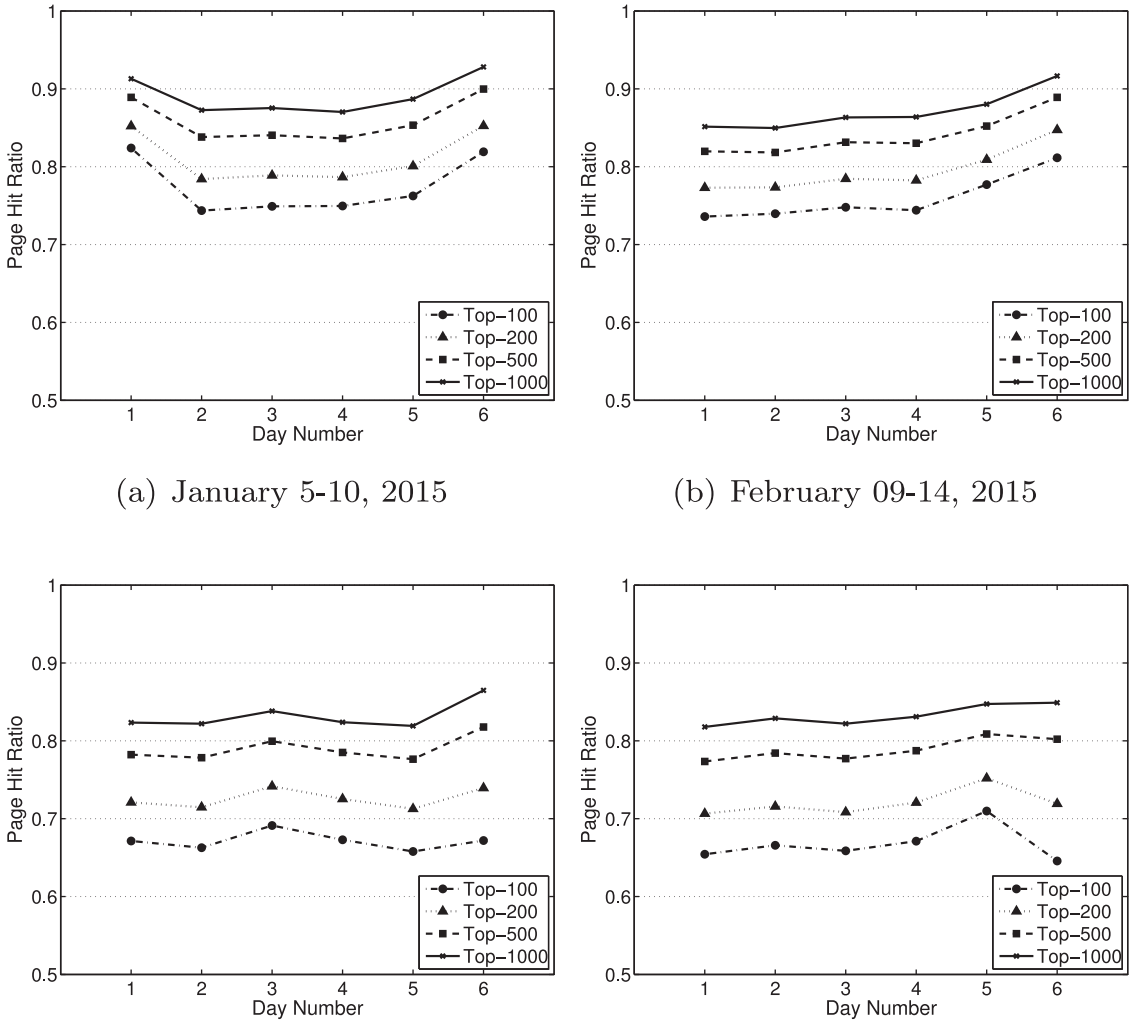
To study the effectiveness of this approach in a real setting, we ran the space saving algorithm over network traffic traces collected from the University of Calgary's Internet link. Similar to the experiment explained in Section 2.3, we used an `AWK` script to extract URLs of the landing pages from HTTP traces and fed them to the space saving algorithm. We performed four experiments over four different six-day intervals. At each experiment, we constructed the top-k list for the URL stream of six consecutive days while measuring page hit ratio separately for each day. Specifically, with each URL in the stream, we first check the top-k list to determine whether it is among the current popular elements (hit occurrence), and if so, we increment both its associated counter in the list and a hit counter. Otherwise, we add the URL and its associated counter to the top-k list in accordance with the space saving algorithm. Finally, page hit ratio of each day is calculated as the number of hits during that day divided by the total number of page requests received on that day.

Fig. 11 shows the page hit ratios achieved for different values of k. It can be seen that by increasing the size of the popular URLs list, k, the page hit ratio increases. For example, in May 25–30 period, by increasing k from 100 to 1000, the average page hit ratio increases from 67% to 83%. Fig. 11 also shows that in all four time periods, by keeping a popular URL list with only 1000 items, a page hit ratio of above 80% can be achieved. By assuming an average 6 seconds page load time on a Desktop computer [25], updating the resource lists of 1000 websites at the proxy will take only one hour and forty minutes. Comparing such a short update time to our results in Section 3.4.1 on the frequency of change in webpage structures implies that the profiler would be able to capture temporal changes in the structure of top-1000 popular webpages (a subset of all the requests in the network) and still provide a high page hit ratio to its users.



(a) WLAN Measurements (b) Cellular Measurements

Fig. 10. 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.



(a) January 5-10, 2015 (b) February 09-14, 2015

(c) May 25-30, 2015 (d) June 8-13, 2015

Fig. 11. Page hit ratios achieved by applying the space saving algorithm to the HTTP traces collected from the University of Calgary's Internet link over four different six day intervals. Increasing the size of the popular URLs list leads to higher page hit ratios. Also, on average, keeping just the top-1000 popular URLs in the stream of URLs that arrive at the proxy, results in over 80% page hit ratios.

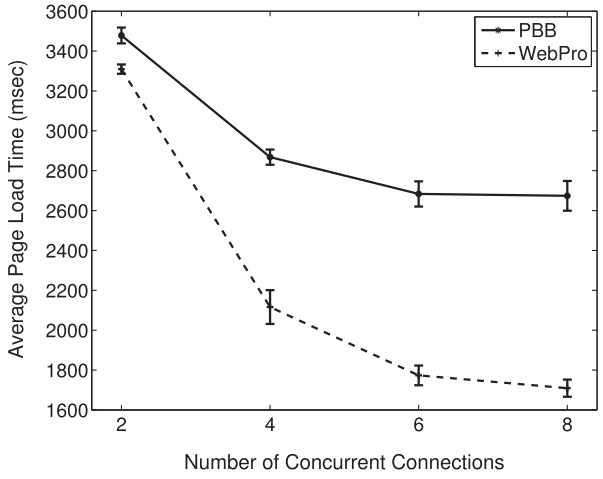


Fig. 12. Average page load time for a Wikipedia article page as a function of the number of parallel connections. We see that increasing the concurrency reduces the page load time. The benefits are greater for WebPro as it can fetch more subresources concurrently.

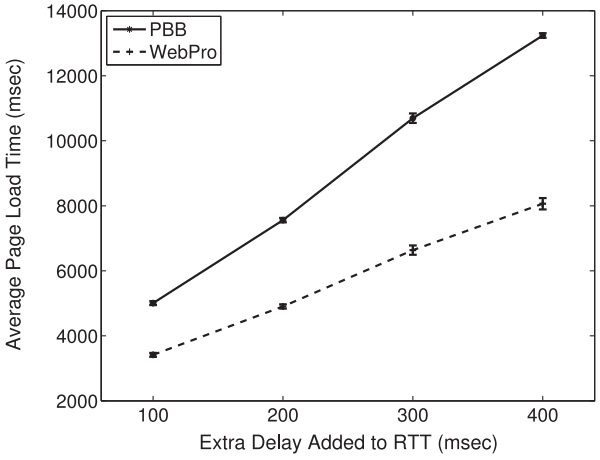


Fig. 13. Average page load time for a Wikipedia article page as a function of network delay. We see that higher RTT values lead to higher page load times. By increasing RTT, PBB incurs higher latencies compared to WebPro.

3.4.4. Effect of concurrent connections

As mentioned in Section 2, 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 [33]. 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 connections 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. Fig. 12 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.

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.

Fig. 12 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.

3.4.5. Effect of network delay

As mentioned in Section 2, 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 varies 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 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 [34] 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. Fig. 13 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.

Fig. 13 also shows that with 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 [20]. 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.

3.4.6. Effect of webpage complexity

As mentioned in Section 2.2, there are inter-object dependencies in today's webpages that lead to the serialization of network transfers required for loading a page. One of the common cases of

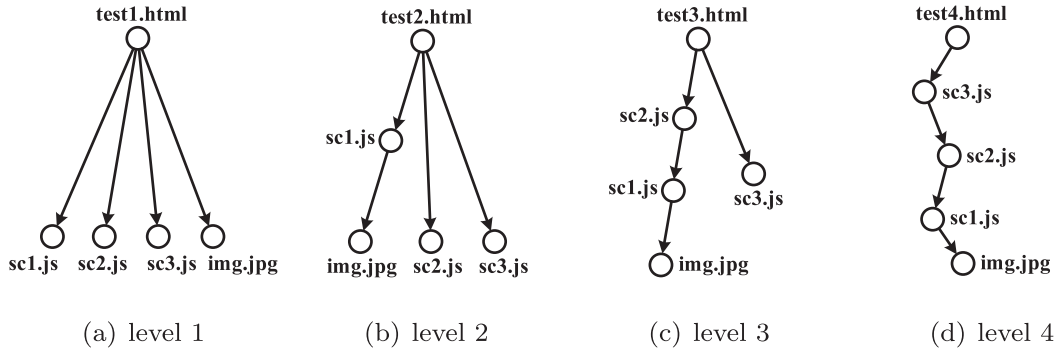


Fig. 14. Dependency graphs for four carefully designed test pages with the same set of embedded objects. In the first test page (a), all the objects can be discovered after fetching and parsing the base HTML file, giving it a critical path of length 1. The second page (b) has a critical path of length 2, because the image object can be revealed after fetching and evaluating the JavaScript object sc1.js. In the third page (c) with critical path length 3, fetching and evaluating JavaScript object sc2.js, reveals another JavaScript object, sc1.js, the fetching and evaluation of which reveals the image object. Finally, in the last page (d) with critical path length 4, evaluating sc3.js reveals sc2.js, evaluating sc2.js reveals sc1.js and evaluating sc1.js reveals the image object.

such dependencies is created when an embedded object itself embeds other objects. For example a JavaScript object can embed any kind of object and a CSS file can embed background images. In this case, discovering the object referenced in a script file, requires another RTT between the browser and the origin server. Because of such dependencies, a browser (or a web engine) cannot discover all the embedded objects of a page right after fetching and parsing the base HTML file. On the contrary, resource exploration becomes an iterative process in which local computations such as parsing and script executions are interleaved with network transfers [12].

Other than eliminating the base HTML fetch time, one other reason for WebPro's superior performance is that it breaks such object level dependencies by using a previously recorded resource list. To study the benefits that come from eliminating inter-object dependencies, we carefully designed 4 webpages, all with the same set of embedded objects. The base HTML files of these pages have slight differences but all are of the same sizes (174 Bytes).[5] Also in all the 4 pages, the final rendered page is the same which consists of just a pigeon image on the screen. The major difference between these pages is in the amount of dependency between their objects. Specifically, from the first page (test1.html) to the last page (test4.html), we gradually increase the length of the critical path in their dependency graphs. Fig. 14 depicts the dependency graphs for these 4 pages. Test webpages are available at http://pages.cpsc.ucalgary.ca/~asehati/webpro/.

We hosted our test pages on an Apache web server running on a Linux machine in our lab. Similar to the previous section, dummynet was used to inject 200 ms emulated delay between remote proxy and the web server. Using the same WiFi setting described in Section 3.1, we loaded each test page ten times with WebPro and PBB and computed the average speedup achieved with WebPro.[6] Borrowing the definition from [35], WebPro's speedup relative to PBB, is the ratio of page load time using PBB to the page load time under WebPro. Given that all the 4 pages have the same set of embedded objects and their base HTML files are of the same size, WebPro results in the same page load times for all of the test pages. The reason is that the resource list files of all the pages point to the same set of embedded objects hosted on the same server and also base HTML fetch times are the same. However, different levels of complexity in these pages lead to different page load times under PBB which by using a web engine goes through an iterative process of discovering embedded objects.

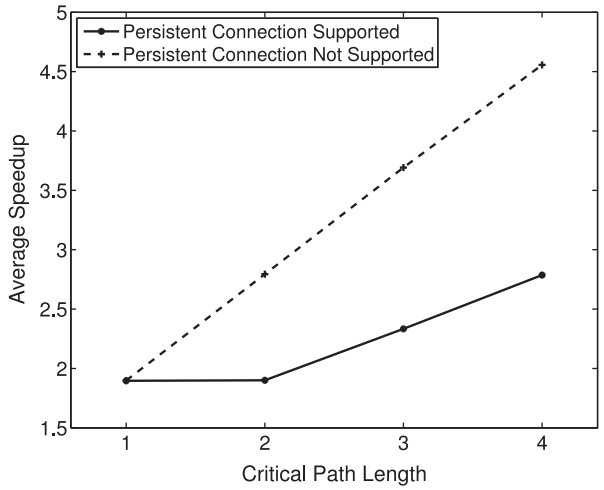


Fig. 15. Speedup of WebPro relative to PBB as a function of critical path length. As the critical path becomes longer, the speedup with WebPro increases. Also, for a given webpage, speedups with WebPro are higher under the setting that does not support persistent connections.

Fig. 15 depicts WebPro's speedup relative to PBB as a function of critical path length under two settings. In one setting (called *first setting*), persistent connections were supported by the web server and in the other setting (called *second setting*), persistent connections were disabled in the web server. Notice that for a given test page, WebPro achieves the same load times under these two settings, but in three out of the four test pages (test 2 through 4), PBB achieves higher page load times under the second setting. The reason is that WebPro's proxy establishes 5 parallel TCP connections to the server right at the beginning and by using each connection only once, does not get affected by the lack of support for persistent connections. However in PBB under the second setting, every time that evaluating one object reveals another object, the proxy will incur one extra RTT to establish a new TCP connection for fetching the newly discovered object. For these reasons, we can see in Fig. 15 that for a given webpage, speedups with WebPro are higher if persistent connections are not used between the proxy and the web server. The only exception is page 1 in which all the objects are referenced in the base HTML and there are no inter-object dependencies.

[5] The size of the HTML files were made equal by inserting the required number of blank spaces after the < /html > tag.

[6] 95% confidence intervals were also computed but are not presented since they were very small.

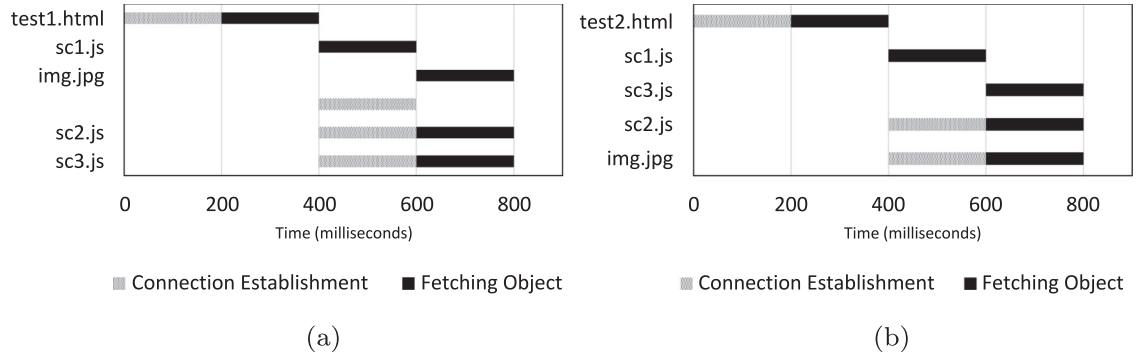


Fig. 16. Waterfall of loading (a) the first and (b) the second test page using PBB with persistent connections enabled. In (a), test1.js, sc1.js and img.jpg and in (b) test2.html, sc1.js and sc3.js are downloaded over the same connection.

From Fig. 15, it can also be observed that speedups with WebPro increase as the critical path becomes longer. By increasing the critical path length from one to four, the speedup of 1.9 reaches 2.79 and 4.56 in the first and second settings, respectively. Given that WebPro achieves the same page load times for all pages, higher speedup comes from higher page load times under PBB. Specifically, in the first setting, increasing the critical path length by one adds one RTT to the page load time of PBB which is the time required to fetch and evaluate the new referencing object inserted in the critical path. In the second setting, extending the critical path by one edge adds two RTTs to the page load time of PBB. One RTT is incurred for establishing a new TCP connection to the server and another RTT is incurred for fetching and evaluating the new referencing object.

We observe that in Fig. 15, under the first setting, WebPro achieves the same speedups for page 1 and 2 which again is due to the fact that same page load times are achieved with PBB for these two pages. Fig. 16 shows the waterfall of loading these two pages using PBB under the first setting. To load test1.html, Web engine first establishes a connection to the server (we call it connection 1) to fetch the base HTML file of the page. After fetching and parsing the base HTML file, the web engine finds links to four new objects (img.jpg, sc1.js, sc2.js, and sc3.js). Because of persistent connections, connection 1 is still available and can be used for fetching one of the four newly discovered objects. Specifically, web engine issues the request for sc1.js over connection 1 and at the same time initiates three new parallel connections to the server (we call them connections 2, 3 and 4). Notice that initiating a connection means the exchange of SYN and SYNACK segments between the proxy and the server which takes one RTT. On the other hand, considering the small size of sc1.js (65 bytes), it takes one RTT to request and receive this file at the proxy. As a result, by the time that those three connections are established, sc1.js has arrived at the proxy and connection 1 has become available again. Therefore, at this point in time (marked as 400 ms in Fig. 16(a)), the proxy has 4 available connections to the server (connections 1, 2, 3 and 4) but there are just 3 objects remaining from the page (img.jpg, sc2.js and sc3.js). The proxy proceeds by using connection 1 for fetching img.jpg and at the same time issues requests for sc2.js and sc3.js over two of the three newly opened connections (connections 3 and 4). Finally, the proxy is able to fetch all the required objects of the page without using connection 2. A similar explanation can be used to describe waterfall of page test2.html which is downloaded at the proxy after four RTTs.

We note that the last three experiments study the behaviour of WebPro and PBB under different system conditions, i.e. concurrency limit, network delay and webpage complexity. 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.

Table 4
Improvement in average radio-on time with bundling.

Webpage	Radio-on time (ms)		Improvement
	WebPro w/ bundling	WebPro w/o bundling	
business.gov.au	2333	3395	31.3%
spark.co.nz	3229	3881	16.8%
mashreghnews.ir	3042	5225	41.8%
zju.edu.cn	3374	4521	25.4%

3.4.7. Energy impact of bundling

As mentioned in Section 2, the remote proxy sends all the required objects of a page in a bundle to the client. Transmitting a batch instead of a sequence of small objects prevents the radio of the device from constant promotions and demotions which can quickly drain the battery of the device. On the other hand, as mentioned in Section 2.4 (Incremental Rendering), bundling is not an indispensable feature of WebPro and we can envision WebPro without bundling.

However, to verify the energy efficiency of WebPro with bundling, we performed a set of experiments using real webpages. To do so, we used most of the experimental setup described in Sections 3.1 and 3.2 of the paper. The only difference was that we emulated LTE network conditions using dummynet network emulator. Specifically, dummynet was used to inject extra delay and throttle bandwidth so that the RTT and throughput of our emulation would be consistent with real-world settings reported in recent measurement studies [36]. In our experiments we measure the radio-on time which starts from the time that the client receives the first set of bytes and ends with the reception of the last byte. In order to consider LTE's Radio Resource Control (RRC) state machine, we also incorporated tail time effect in our analysis. Specifically, idle gaps between consecutive objects that are greater than the tail time, only contribute the amount of tail time to the radio-on time.

Table 4 presents our results for 4 different webpages that are among popular sites in 4 different countries. All measurement results are averaged over 10 runs. It can be seen that bundling reduces radio-on time which implies reduction in energy consumption of mobile web browsing. However, the amount of improvement achieved with bundling varies between different webpages. This is due to a set of reasons such as different amounts of inter-object idle gaps, different number of objects and differences in topological distance between the proxy and the web servers.

4. Discussion

User mobility: While we addressed the challenges arising from long access delays of wireless networks, it should be noted that the mobility of users while surfing the web can make accelerating mobile web even more challenging. Specifically, mobility of users can cause unpredictable network conditions, rate variability and signaling overheads, all of which can contribute to the poor browsing experience of mobile users. Considering the recent efforts on the application of Multi-Path TCP (MPTCP) to mobile devices [37], mobility of users can be facilitated by using multiple interfaces (WiFi and cellular) available in most of the mobile devices. In such a setting, MPTCP's backup mode [38] will be used to provide the user with a seamless transition experience as he/she walks between WiFi APs or walks out of the WiFi coverage.

WebPro in error-prone wireless networks: In WebPro, the transmission of bundles between the client and the remote proxy is performed over TCP. This implies that the TCP protocol will provide a reliable channel service to our system, WebPro. Specifically, TCP will react to any packet loss by retransmitting the missing packets. In other words, the granularity of TCP's retransmission is in the packet level and from this perspective, there is no difference between transmitting individual objects versus transmitting bundles over the wireless link.

5. 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.

Recently, there have been measurement studies to assess the effectiveness of client-side caching and prefetching in improving the performance of mobile web browsing. For instance, Ma et al. [39] conducted comprehensive measurements to characterize the performance of mobile web caching. They identified *redundant transfers* and *miscached resources* (providing out-of-date resources from cache) as the two main problems that negatively affect mobile web caching performance. Their investigations revealed *Same Content* (same resources having different URLs at different times), *Heuristic Expiration* and *Conservative Expiration* as the root causes of unsatisfactory cache performance. Wang et al. [7] used a web dataset collected from 24 iPhone users over a year to quantitatively evaluate client-only caching and prefetching. 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 6.3% of all websites support SPDY [40], its impact so far has been rather limited. Also the next generation HTTP protocol, HTTP/2, which was standardized on February 2015, evolved from SPDY [41] and therefore inherits most of its features. However, there are a few differences between SPDY and HTTP/2.0 for example in their header compression algorithms [42].

With server push being one of the main novelties in SPDY/HTTP 2.0, there have been several proposals to resolve some of its limitations or to further improve the efficiency of this feature. For example, notice that the server is oblivious of client's cache status and by pushing content that already exists in client's cache, it can waste bandwidth and battery of the mobile device. To address this issue, Khalid et al. [11] proposed sending cache hints from client to server in the form of bloom filters. To further adjust the performance of server push, they also proposed the ideas of *half-push* and *half-pull*. In half-push, the server pushes the content to an edge proxy rather than the client and in half-pull the client requests the content to be brought to the proxy without traversing the last mile. Finally, authors of [12] proposed a novel framework that uses the server push feature in HTTP/2 to preemptively push resource lists of the requested page and all its subpages, to the client. By using these cached meta files, a future request for a subpage can be issued in parallel with the subresource fetching requests of that page. In this scheme, client incurs little extra bandwidth overhead due to meta data transfers but can benefit from speedup in downloading subpages.

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. [43] 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. [16,17] 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 [4], offloading compute-intensive operations when loading a webpage has negligible benefits compared to the improvements resulting from resource loading optimizations. Also Sivakumar et al. [18] 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 [15,44], while others employ solutions that directly deal with network access delay [45]. For example, Opera Mini [15] and Amazon Silk [44] are cloud-based browsers that offload portions of the page load process to cloud-based proxies. These browsers are

widely used today based on the common belief that using compression proxies reduces data usage of mobile web and thus, reduces latency. However, the results of a recent measurement study in [46] reveals that using compression proxies in good network conditions can increase page load time rather than improving it. As a result, they design and implement a framework called *FlexiWeb* that decides whether to fetch a resource from middle box or the original server based on the object size and the network conditions. In line with the findings of [46], a recent study by Sivakumar et al. [47] also shows that cloud-based browsers are not always superior in terms of responsiveness and energy consumption, especially in dealing with client interactions.

Instead of directly reducing page load time in mobile web browsing, authors of [32] proposed *KLOTSKI* that aims at improving mobile user's quality of experience by delivering as many high utility resources as possible within tolerance limits of mobile users ($3 - 5$ seconds). To this end, KLOTSKI employs a cloud-based proxy to capture and update different properties of websites such as dependency structure, resource sizes and positions on rendered displays and stores them in the form of a compact fingerprint. When loading a page, those fingerprints are used to dynamically reprioritize delivery of different resources.

The closest work to ours is EEP by Wang et al. [16,17] and PARCEL by Sivakumar et al. [18]. While their focus is on reducing energy consumption by batching and compression [16–18], 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.

6. 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 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. We also intend to build an analysis model of WebPro in an attempt to capture the relationship between different parameters of the system.

References

[1] A. Sehati, M. Ghaderi, WebPro: A proxy-based approach for low latency web browsing on mobile devices, in: Proceedings of IEEE/ACM IWQoS, 2015, pp. 319–328.
[2] S. Souders. Velocity and the bottom line, accessed December 14, 2015. http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html.
[3] J. Huang, et al., Anatomizing application performance differences on smartphones, in: Proceedings of ACM MobiSys, 2010, pp. 165–178.
[4] Z. Wang, F.X. Lin, L. Zhong, M. Chishtie, Why are web browsers slow on smartphones? in: Proceedings of ACM HotMobile, 2011, pp. 91–96.
[5] B.D. Higgins, J. Flinn, T.J. Giuli, B. Noble, C. Peplin, D. Watson, Informed mobile prefetching, in: Proceedings of ACM MobiSys, 2012, pp. 155–168.
[6] D. Lymberopoulos, O. Riva, K. Strauss, A. Mittal, A. Ntoulas, Pocketweb: Instant web browsing for mobile devices, in: Proceedings of ACM ASPLOS, 2012, pp. 1–12.
[7] Z. Wang, F.X. Lin, L. Zhong, M. Chishtie, How far can client-only solutions go for mobile browser speed? in: Proceedings of ACM WWW, 2012, pp. 31–40.
[8] B. Zhao, B.C. Tak, G. Cao, Reducing the delay and power consumption of web browsing on smartphones in 3G networks, in: Proceedings of IEEE ICDCS, 2011, pp. 413–422.
[9] SPDY: An experimental protocol for a faster web, accessed December 14, 2015. http://www.chromium.org/spdy/spdy-whitepaper.
[10] J. Erman, V. Gopalakrishnan, R. Jana, K. Ramakrishnan, Towards a SPDY'ier mobile web, in: Proceedings of ACM CoNEXT, 2013, pp. 303–314.
[11] J. Khalid, S. Agarwal, A. Akella, J. Padhye, Improving the performance of SPDY for mobile devices, Proceeedings of ACM HotMobile (Poster Session), 2015.
[12] B. Han, S. Hao, F. Qian, Metapush: Cellular-friendly server push for HTTP/2, in: Proceedings of ACM Workshop on All Things Cellular, 2015, pp. 57–62.
[13] F. Qian, et al., Web caching on smartphones: ideal vs. reality, in: Proceedings of ACM MobiSys, 2012, pp. 127–140.
[14] H. Shen, Z. Pan, H. Sun, Y. Lu, S. Li, A proxy-based mobile web browser, in: Proceedings of ACM Multimedia, 2010, pp. 763–766.
[15] Opera mini browser, accessed December 14, 2015. http://www.opera.com/mobile.
[16] L. Wang, J. Manner, Energy-efficient mobile web in a bundle, Comput. Netw. 57 (17) (2013) 3581–3600.
[17] L. Wang, B. Yu, J. Manner, Proxies for energy-efficient web access revisited, in: Proceedings of ACM e-Energy, 2011, pp. 55–58.
[18] A. Sivakumar, S. Puzhavakath Narayanan, V. Gopalakrishnan, S. Lee, S. Rao, S. Sen, PARCEL: Proxy assisted browsing in cellular networks for energy and latency reduction, in: Proceedings of ACM CoNEXT, 2014, pp. 325–336.
[19] Alexa Internet Inc. "Top Sites in Canada", accessed December 14, 2015. http://www.alexa.com/topsites/countries/CA.
[20] X.S. Wang, A. Balasubramanian, A. Krishnamurthy, D. Wetherall, Demystifying page load performance with wprof, in: Proceedings of USENIX NSDI, 2013.
[21] A. Gerber, S. Sen, O. Spatscheck, A call for more energy-efficient apps, AT&T Labs Res. (2011).
[22] R. Mahindra, H. Viswanathan, K. Sundaresan, M.Y. Arslan, S. Rangarajan, A practical traffic management system for integrated LTE-WiFi networks, in: Proceedings of ACM MobiCom, 2014, pp. 189–200.
[23] G. Barish, K. Obraczka, World wide web caching: Trends and techniques, IEEE Commun. Mag. 38 (2000) 178–184.
[24] V. Paxson, Bro: a system for detecting network intruders in real-time, Comput. Netw. 31 (23) (1999) 2435–2463.
[25] Is the web getting faster?, accessed August 19, 2015. http://analytics.blogspot.ca/2013/04/is-web-getting-faster.html.
[26] G. Cormode, M. Hadjieleftheriou, Finding the frequent items in streams of data, Commun. ACM 52 (10) (2009) 97–105.
[27] A. Metwally, D. Agrawal, A. El Abbadi, Efficient computation of frequent and top-k elements in data streams, in: Database Theory-ICDT 2005, Springer, 2005, pp. 398–412.
[28] A. Rao, A.M. Kakhki, A. Razaghpanah, A. Tang, S. Wang, J. Sherry, P. Gill, A. Krishnamurthy, A. Legout, A. Mislove, et al., Using the middle to meddle with mobile, Technical Report on NEU-CCS-2013-12-10, CCIS, Northeastern University, 2013.
[29] F. Qian, J. Huang, J. Erman, Z.M. Mao, S. Sen, O. Spatscheck, How to reduce smartphone traffic volume by 30%? in: Proceedings of PAM, 2013, pp. 42–52.
[30] L. Deutsch, J. Gailly, Rfc 1950: Zlib compressed data format specification version 3.3, IETF (May 1996).
[31] J. van den Brande, A. Pras, The costs of web advertisements while mobile browsing, in: Information and Communication Technologies, Springer, 2012, pp. 412–422.
[32] M. Butkiewicz, D. Wang, Z. Wu, H.V. Madhyastha, V. Sekar, Klotski: Reprioritizing web content to improve user experience on mobile devices, in: Proceedings of USENIX NSDI, 2015, pp. 439–453.
[33] B. Thomas, R. Jurdak, I. Atkinson, SPDYing up the web, Commun. ACM 55 (12) (2012) 64–73.
[34] M. Carbone, L. Rizzo, Dummynet revisited, ACM SIGCOMM Comput. Commun. Rev. 40 (2) (2010) 12–20.
[35] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, H. Balakrishnan, Mahimahi: accurate record-and-replay for http, in: Proceedings of USENIX Annual Technical Conference, 2015, pp. 417–429.
[36] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z.M. Mao, S. Sen, O. Spatscheck, An in-depth study of lte: effect of network protocol and application behavior on performance, in: ACM SIGCOMM Computer Communication Review, volume43, 2013, pp. 363–374.
[37] B. Han, F. Qian, S. Hao, L. Ji, N. Bedminster, An anatomy of mobile web performance over multipath TCP, in: Proceedings of ACM CoNEXT, 2015.
[38] S. Deng, R. Netravali, A. Sivaraman, H. Balakrishnan, WiFi, LTE, or both?: Measuring multi-homed wireless internet performance, in: Procedings of ACM IMC, 2014, pp. 181–194.
[39] Y. Ma, X. Liu, S. Zhang, R. Xiang, Y. Liu, T. Xie, Measurement and analysis of mobile web cache performance, in: Proceedings of ACM WWW, 2015, pp. 691–701.
[40] Usage Statistics of SPDY for Websites, accessed September 26, 2015. http://w3techs.com/technologies/details/ce-spdy/all/all.
[41] D. Stenberg, HTTP2 explained, ACM SIGCOMM Comput. Commun. Rev. 44 (3) (2014) 120–128.
[42] M. Varvello, K. Schomp, D. Naylor, J. Blackburn, A. Finamore, K. Papagiannaki, To HTTP/2, or not to HTTP/2, that is the question, in: Proceedings of PAM, 2016.

[43] B. Aggarwal, P. Chitnis, A. Dey, K. Jain, V. Navda, V.N. Padmanabhan, R. Ramjee, A. Schulman, N. Spring, Stratus: energy-efficient mobile communication using cloud support, ACM SIGCOMM Comput. Commun. Rev. 40 (4) (2010) 477–478.
[44] Amazon silk browser, accessed December 14, 2015. http://amazonsilk.wordpress.com/.
[45] R. Chakravorty, A. Clark, I. Pratt, Optimizing web delivery over wireless links: design, implementation, and experiences, Selected Areas Commun. IEEE J. 23 (2) (2005) 402–416.
[46] S. Singh, H.V. Madhyastha, V. Srikanth, R. Govindan, Flexiweb: Network-aware compaction for accelerating mobile web transfers, in: Proceedings of ACM MobiCom, 2015.
[47] A. Sivakumar, V. Gopalakrishnan, S. Lee, S. Rao, S. Sen, O. Spatscheck, Cloud is not a silver bullet: a case study of cloud-based mobile browsing, in: Proceedings of ACM HotMobile, 2014.

Ali Sehati received his B.Sc. and M.Sc. degrees in computer engineering from the University of Tehran, Iran in 2009 and 2012, respectively. He is currently pursuing his Ph.D. in computer science at the University of Calgary, Canada. His current research focuses on energy efficient mobile computing and mobile web performance.

Majid Ghaderi is an Associate Professor in the Computer Science Department at the University of Calgary. Before joining the University of Calgary, he was a Postdoctoral Research Associate in the Computer Science Department at the University of Massachusetts at Amherst. Dr. Ghaderi received B.Sc. and M.Sc. degrees from Sharif University of Technology, and a Ph.D. degree from the University of Waterloo, all in computer science. His research interests include wireless networking and mobile computing with emphasis on modeling and performance analysis of wireless networks.