Demystifying the Imperfect Client-Side Cache Performance of Mobile Web Browsing

Xuanzhe Liu, *Member, IEEE,* Yun Ma, Yunxin Liu *Member, IEEE,* Tao Xie *Senior Member, IEEE,* and Gang Huang *Senior Member, IEEE*

Abstract—The Web browser is one of the most significant applications on mobile devices such as smartphones. However, the user experience of mobile Web browsing is undesirable because of the slow resource loading. To improve the performance of Web resource loading, client-side cache has been adopted as a key mechanism. However, the existing passive measurement studies cannot comprehensively characterize the "*client-side*" cache performance of mobile Web browsing. For example, most of these studies mainly focus on client-side implementations but not server-side configurations, suffer from biased user behaviors, and fail to study "*miscached*" resources. To address these issues, in this article, we present a proactive approach to making a comprehensive measurement study on client-side cache performance. The key idea of our approach is to proactively crawl resources from hundreds of websites periodically with a fine-grained time interval. Thus, we are able to uncover the resource update history and cache configurations at the server side, and analyze the cache performance in various time granularities. Based on our collected data, we build a new cache analysis model and study the upper bound of how high percentage of resources could potentially be cached and how effectively the caching works in practice. We report detailed analysis results of different websites and various types of Web resources, and identify the problems caused by unsatisfactory cache performance. In particular, we identify two major problems- ***Redundant Transfer*** and ***Miscached Resource***, which lead to unsatisfactory cache performance. We investigate three main root causes: ***Same Content***, ***Heuristic Expiration***, and ***Conservative Expiration Time***, and discuss what mobile Web developers can do to mitigate those problems.

Index Terms—Mobile Web, Cache, Measurement.

1 Introduction

Among the numerous mobile applications, a.k.a, apps, the Web browser is the most frequently used one. According to a recent study [1], as of 2013, Web browsers occupy 63% of the window focus time on the subjects' mobile devices. Recently, besides the standard browser, native applications also involve built-in browser facilities, such as `WebView` on Android, to realize part of its functionalities including detailed information exhibition, link sharing, etc.

However, the user experience of mobile Web browsing is far from satisfaction. Resource loading is one of the key factors influencing Web browsing performance. A webpage consists of a set of resources, such as HTML, CSS, JavaScript, and images. After users type in a URL and press the "*Go*" button or click through a hyperlink, the target webpage is loaded into the browser. During the page load, all related resources are fetched from the Web server, and then parsed or evaluated to finally render the page. As a result, downloading resources dominates the resource loading process, and thus affects Web browsing performance. The more resources need downloading from the network, the more data traffic it consumes to load the webpage. It is reported that 65% of page load time is spent on resource loading [2]. Mobile users suffer from slow page load, which leads to long time of screen light-up and inevitably consumes more energy [3]. Such a problem becomes even worse on mobile devices because of the unreliable network conditions, particularly when the network is congested or the user is moving around.

- *Xuanzhe Liu, Yun Ma, Gang Huang are with the Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing, China, 100871. Email: {liuxuanzhe, mayun, hg}@pku.edu.cn*
- *Yunxin Liu is with Microsoft Research. Beijing, China, 100084. Email: yunxin.liu@microsoft.com*
- *Tao Xie is with the University of Illinois at Urbana-Champaign. Email: taoxie@illinois.edu*

Manuscript received June, 2015

In practice, client-side cache[1] is particularly helpful for resource loading on mobile devices. By saving the resources of visited webpages into the local storage facilities (memory or disk), these webpages may be served from the local storage rather than being re-fetched from the Web servers when the webpages are accessed again. As a result, client cache can reduce data transmitted over the last mile, i.e., the radio access network, which is shown to be the bottleneck of a cellular network [4].

Indeed, it is significant to understand how mobile Web browsing could benefit from "*client-side cache*" and how well "client-side cache" takes effect in practice. To this end, several measurement studies have been conducted by analyzing user access traces gathered from Internet Service Providers (ISPs) or instrumented client devices [5], [6], [7], [8], [9].

Although the existing studies have already revealed the performance problems of client-side cache on mobile devices, they focus on only the imperfect client-side implementations. They do not comprehensively characterize the end-to-end Web cache performance involving not only Web clients, but also Web servers and user behaviors. More specifically, the existing measurement studies on mobile Web cache have the following main limitations.

•***Existing studies are limited in terms of biased user behaviors***. The traces collected from ISPs or instrumented client devices are passive observations on users' Web access behaviors. These traces are just fragmented snapshots of the websites and thus cannot cover the full resource update history of these websites. As a result, measurements based on these traces could imply only how caching works on the sampled access of the sampled users, but cannot reflect how the websites themselves can essentially benefit from caching.

1. Caching is a widely adopted mechanism on the Web and cache can be deployed at many places between browsers and Web servers. This article focuses on client cache. We use caching, cache, and Web cache interchangeably to denote client-side cache if not specifically stated.

• ***Existing studies do not investigate the influences of server-side caching configurations on cache performance***. Similar to the first issue, as the passively collected data traces cannot cover the full resource update history of the websites, these studies concentrate on client-side implementations but not server-side configurations. Although how the browsers support caching on mobile devices is important, Web cache performance is largely determined by the cache configurations at the server side. For example, if resources are configured as *no-cache* (indicating that they should not be cached), client-side caching cannot help at all. It is desirable to examine how server-side configurations may impact Web caching.

• ***Existing studies do not investigate the problem of miscached resources***. Miscached resources are those that have already been updated on the server, but the browser wrongly uses the expired resources from the cache rather than fetching the updated ones from the server. The passive approaches of data collection cannot capture miscached resources because the passive approaches collect only the resources actually fetched by browsers. However, miscached resources may significantly affect user experience. In particular, if the functional resources (e.g., JavaScript or CSS) are miscached, the corresponding website may not work correctly. Therefore, there is a strong need of studying miscached resources.

To address the preceding limitations faced by the existing studies, in this article, we present a ***proactive*** approach for comprehensive measurement and analysis of mobile Web cache performance. We crawl the resources from hundreds of websites periodically with a fine-grained time interval for a long time duration. In this way, we are able to uncover the resource update history and cache configurations at the server side, and analyze the cache performance in various time granularities, without bias caused by limited samples of users' behaviors. We build an analysis model to answer three research questions. The first one is *how much of a website can be cached*, giving an upper bound of how caching could help improve mobile Web browsing experience. More cacheable resources indicate more performance improvements that the website can benefit from caching. The second question is *how well caching has been supported to reach the upper bound*, providing comprehensive characterization of the state-of-the-art of mobile Web caching. The third question is *what causes the gap between the upper bound and the actual performance*. For various reasons, resources that can be cached may not be actually cached and there exist miscached resources.

To conduct a quantitative study, we measure the performance of Web caching in both the **resource size** and **resource number**. Larger size of resources loaded from the cache indicates less data traffic that needs downloading from the network. However, the resource size itself cannot fully quantify the performance of Web caching. Given the same total size of resources, a larger number of small resources loaded from the cache indicates fewer network connections to the server. Therefore, we consider both the resource size and resource number as the evaluation metrics on how caching influences mobile Web browsing experience in terms of page load time and data traffic.

This article[2] makes the following main contributions:

- ***We propose a proactive approach to acquiring detailed history of resource update***. We choose two sets of websites for a comparative study to uncover both the state-of-the-art and ordinary status of mobile Web cache performance. One set (Top) comes from Alexa's Top 100 list. The other set (Random) comes from 100 websites randomly chosen from Alexa's Top 1,000,000 list. We periodically crawl their resource update history for one week, forming the basis of our analysis.
- ***We design a measurement model for analyzing cache performance***. Based on the resource update history, we formally define the metrics to quantitatively measure cache performance. We then build a cache behavior taxonomy to correlate cache configurations to resource updates. Such a taxonomy maps each possible case to one of the four behavior patterns: Positive Hit (PH), Negative Hit (NH), Positive Miss (PM), and Negative Miss (NM).
- ***We present the empirical results for demonstrating the gap between ideal and actual mobile Web cache performance***. On average, more than half of the total resources can be cached and the Random websites are more cacheable than the Top ones. However, in practice, only a limited portion of the cacheable resources are cached, not more than 50% for the Random websites. Different types of resources have different cacheabilities.
- ***We present the empirical results for highlighting two main problems caused by undesirable cache performance***. The ***Redundant Transfer*** problem comes from NM resources (the resources that can be served from the cache but are actually downloaded from the network). The median Negative Miss Ratios in terms of the resource size of Top and Random websites are 10% and 52% for one-day revisiting interval, respectively. The ***Miscached Resource*** problem comes from NH resources (the resources that have been updated on the server but are actually served from the cache). NH resources have a similar proportion in both Top and Random websites.
- ***We demystify the root causes for undesirable mobile Web cache performance***. Three main root causes are identified: ***Same Content*** (the same resources that have different URLs when requested at different times), ***Heuristic Expiration*** (the caching policy is not explicitly defined by the server and thus it depends on browsers to infer an expiration time), and ***Conservative Expiration Time*** (the expiration time is set to be too short).
- ***We suggest some implications and recommendations on how to improve cache performance by using better cache policies***. For example, the Application Cache of HTML5 can be used to make the resource updates visible to clients.

To the best of our knowledge, our work provides the first study that reveals the imperfect cache performance of mobile Web browsing by proactively exploring the resource update history and cache configurations at the server side. The remainder of this article is organized as follows. Section 2 motivates our study by analyzing how caching works and relevant factors influencing cache performance. Section 3 presents three research questions. Section 4 describes our dataset and data-collection approach. Section 5 presents the analysis methodology and Section 6 presents the detailed measurement results. Based on the findings, we provide implications and recommendations for Web developers in Section 7. Section 8 presents threats to validity. Section 9 surveys related work and Section 10 ends this article with concluding remarks and future work.

2. This article is an extended version of our previous work published at WWW 2015 [10].

2 BACKGROUND

In this section, we describe the background on how Web caching works and possible reasons that may influence the cache performance.

2.1 How Web Caching Works

Figure 1 shows a simplified procedure to illustrate how Web caching works. A webpage consists of a set of resources, such as HTML, CSS, JavaScript, and images. When users type in a URL and press the "*Go*" button or click through a hyperlink, the browser loads the target webpage by acquiring the corresponding resources from the Resource Loader module.

The Resource Loader first tries to load resources from the local cache through the Cache Manager module. According to RFC 2616 [11], HTTP/1.1 defines a cache expiration model and a cache validation model. The cache expiration model allows a Web server to set an expiration time for each resource, indicating how long time the resource can be cached by a client device. Setting expiration time is not mandatory. If a resource does not have an expiration time explicitly set by the server, browsers may apply their own heuristic algorithms to decide the cache expiration time of the resource. Such a mechanism is called Heuristic Expiration.

The Cache Manager checks whether the requested resource (identified based on its URL's protocol, host, port, path, and query) can be found in the Cache Database. If not, the browser sets up a HTTP connection to fetch the resource from the remote Web server. If the resource is found and not expired according to the cache expiration model, it is returned directly from the cache. If the resource is expired, the HTTP/1.1 cache validation model requires the browser to check with the server whether the resource is changed or not (via the last modified time or Etag identifier). If the server determines that the corresponding resource has not changed, it responds with a "*304 Not Modified*" message and the resource is fetched from the cache of the client. Otherwise, the server sends the updated resource back to the browser.

Note that a website may be much more complex than a single Web server. A large website (e.g., Google) may have many backend Web servers. There may also be Content Delivery Network (CDN) servers to enable scalability and geographical distributions. In such a case, a URL's query part often contains a hash code to indicate which CDN server should process the request, and thus the same resource may have different URLs served by different CDN servers. Furthermore, proxy servers and reverse proxies often act in the middle to route the requests to different backend servers for workload balance. Different backend servers could also attach different URLs to the same resource. As discussed in Section 5.3, this problem of "multiple URLs of the same resource" causes undesirable Web cache performance and should be investigated.

2.2 Factors Influencing Cache Performance

Based on the principle of Web cache, we identify five factors that could possibly influence the mobile Web cache performance.

•**Website Types**. Websites of different types have different kinds of resources, which are updated with different frequencies. For example, a news website may update its texts and images in a very short time period to meet the requirement of timeliness, while resources of a search website could remain stable for quite some time to provide consistent

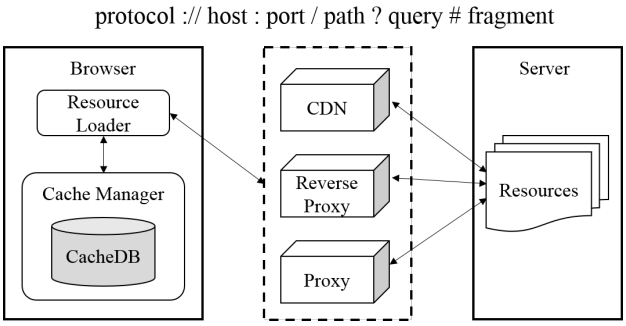


Fig. 1. Architecture of Web caching

experience for users. Therefore, cache performance of news websites is usually lower than that of search websites.

•**Web Administrators**. Web administrators are responsible for deploying websites. During the deployment, they are supposed to configure the servers to support cache. For example, they may configure different cache parameters for different resources that may have different lifetimes. To enable geographical distribution and load balance of backend servers, Web administrators could also adopt CDNs and reverse proxies to dispatch requests. Such middle points between browsers and servers can also influence the cache performance.

•**Web Developers**. Some web techniques may actually disable cache or change its original configuration. For example, in order to achieve version management, developers would like to attach a version number after the URL of resources (e.g., http://foo.com/resource.js?v=1). Sometimes developers choose to randomly generate the number with JavaScript code. Since the HTTP specification regards the whole URL including the query string as the cache entry index, resources with different version numbers are treated as different resources. As a result, even if the resource is configured to be cached and does not change at all, it has to be requested again.

•**Browser Runtime.** Cache is ultimately implemented by browsers. How well browsers support caching has a direct impact on the cache performance, especially given the fact that there are various kinds and versions of browsers. The implementation includes the size of the cache space, whether the cache is volatile, and the accordance to specifications.

• **End User Behaviors.** End users have their own habits for mobile Web browsing. These habits include which websites they usually visit, how often they visit each website, how much of a website they would like to visit, etc. End user behaviors also include how end users interact with a browser, such as whether they clear browsing histories when closing the browser, whether they often use the refresh button to reload a page etc. Caching performance heavily depends on user habits and behaviors.

The type of website dominates the cacheability of resources and thus gives an upper bound of cache performance. While browser runtime is responsible for implementing the cache logic, Web administrators and developers have the most impact on the utilization of cacheability since they have to configure the cache parameters. Otherwise, even if the browser perfectly implements the cache mechanisms, cache can hardly make a difference when configured imperfectly or even wrong. End-user behaviors dominate the actual cache performance that can be achieved. For

example, given the same website, the actual cache performance may vary a lot among different revisiting intervals.

3 Motivation and Problem Statement

Indeed, cache is one of the most conventional technologies adopted widely in computer science. The traditional Web caching research mainly targets at how to reduce the overall network traffic for the Internet. Infrastructure supports are proposed to improve the global Web's performance and reduce server workloads [12], [13], [14], [15]. For mobile computing, cache was mainly investigated in the MANET environment, such as a push-oriented [16], pull-oriented [17], or hybrid-based [18] approach. These studies try to maintain cache consistency between MANET and the original servers with the lowest cost, i.e., data traffic and energy consumption. When it comes to mobile Web, studies on Web cache are shifted to the performance for end users.

Compared to previous efforts, our work is to conduct a comprehensive study on the performance of mobile Web caching. Specifically, we intend to answer the following three research questions.

- ***RQ1: What percentage of Web resources can be cached ideally?*** By answering this question, we study the upper bound of the benefit of Web caching. We consider not only the redundant transmission of the same resource (identified based on its unique URL), but also the redundancy of the resources with different URLs.
- ***RQ2: What percentage of the cacheable Web resources are actually cached if the client is perfectly implemented according to the specification?*** By answering this question, we uncover how well developers leverage the caching mechanism in practice, and reveal the gap between the ideal and actual performance of mobile Web caching.
- ***RQ3: What causes the gap between ideal and actual performance?*** By answering this question, we investigate the root causes of the undesirable actual performance of Web caching, and give Web developers recommendations on how to improve the performance of mobile Web caching.

Answering these questions requires tracking the whole resource update history of websites. This requirement cannot be met by existing passive approaches of data collection. Therefore, we propose a new proactive approach of data collection. We next present how it works.

4 Data Collection

In this section, we describe how we proactively crawl Web resources and what data sets are used to conduct our study.

4.1 Proactive Data Collection Approach

We propose a proactive approach to capture the resource update history of a specific website. For a given URL, we periodically visit the corresponding webpage with a short time interval and for a long time duration. For each visit, we first clean the cache and then record all the returned resources. The short revisiting interval not only makes it possible to obtain the complete resource update history, but also enables us to make flexible analysis with various time granularities. The long data-collection time duration ensures enough samples and lowers the bias. We develop a tool to realize our approach and Figure 2 shows its architecture.

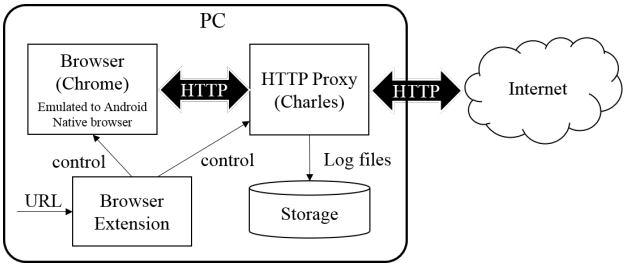


Fig. 2. Data collection tool

Our tool runs on a PC using the Chrome browser. We enable the Chrome browser's emulation mode to make it act as a mobile browser (an Android 4.2 native browser in our case) so that a website returns its mobile-version webpages (if the website has a mobile version). It is important to use a real browser to crawl data. The reason is that many websites embed JavaScript code, and more resources may be loaded by the JavaScript-code execution. Only when the webpage is actually *rendered* by a browser, can we obtain the whole set of resources. Furthermore, using a PC-version browser helps us leverage the browser's programmability, which is not easy on Android smartphones.

We use the Charles proxy [19], a commercial product, to act as the middle tier for data collection. Charles can record each TCP connection passed through it and save HTTP traces in structured data. The data structures split HTTP protocol elements into different fields, making it easy to perform further analysis. In addition, Charles has a programming interface to ease the data recording and saving. We install Charles on the same PC and configure the Chrome browser to use Charles as the proxy. We disable the caching feature of Charles to ensure the completeness of crawled resources.

We build a Chrome extension to automate our data collection. The extension takes a list of URLs and controls the Chrome browser to visit the URLs one by one. Before each visit, the browser cache is cleaned in order to ensure that all the resources are loaded from the network. When a Chrome *TabComplete* event is detected, the extension regards that the website has finished loading. Note that some websites utilize lazy loading, and thus there are still resource requests after *TabComplete* is fired. To mitigate such resource incompleteness, the extension is set to wait for two seconds after a *TabComplete* event. Finally, the extension invokes the Charles programming interface to save all the resource request records on the disk. We can also configure the extension to periodically crawl a list of URLs with a given time interval.

4.2 Data Sets

For a comparative study, we choose two sets of websites from the Alexa [20] ranking list in January 2014. The first set (Top) contains Alexa's top 100 websites. We expect that their cache performance is highly optimized and regard them as the state-of-the-art of Web caching. The other set (Random) contains 100 websites randomly selected from Alexa's top 1,000,000 list. We assume that their cache performance is not fully optimized and regard them as the average level of cache performance.

We manually checked each website in the two sets and filtered out the following ones.

• Unreachable websites. Some websites were not reachable because they were shut down at the time when we visited.

Unreachable websites occurred in only the Random data set.
• The same websites with different domain names. For example, in the Top data set, Google appears 16 times such as google.in, google.de, and google.jp. Since they have the same functionality and appearance, we keep only google.com in our Top data set. However, we treat yahoo and yahoo.jp as two different websites because their contents are totally different.
• HTTPS-only websites. As HTTPS prevents us from capturing the Web resources transmitted over the encrypted network connection, we exclude those websites that support only the HTTPS protocol.

After the filtering, 146 websites remain in total, 55 in the Top data set, and 91 in the Random data set. Note that not all of the 146 websites have a mobile version customized for mobile browsing. According to our manual checking, 7 of the 55 Top websites and 56 of the 91 Random websites do not have their mobile versions. However, the goal of our measurement is to study how Web caching performs on mobile devices for all websites, not only the ones with a mobile version. Therefore, we do not distinguish mobile or non-mobile websites.

For these 146 websites, we used our data-collection tool to periodically crawl their homepages for one week. The crawling time interval was 30 minutes, being short enough to capture all the resource changes. Thus, in this one week, we visited each website for more than 300 times. In total we crawled 157 GB data, 73 GB for the Top websites, and 84 GB for the Random websites.

5 Measurement Methodology

In this section, we formally define the measurement metrics used to study Web cache performance. We also design a taxonomy to map the relationship between cache specifications and actual resource updates.

5.1 Metric Definition

Cache works only when a website is revisited. So we consider two visits. At the first visit, the browser cache is empty and all the resources needed to render the website have to be loaded from the network. After a given Revisiting Interval (RI) Δt, the same website is visited for the second time, where some of the resources are loaded from the browser cache while others are loaded from the network. We examine how the browser cache behaves for the second visit.

We define $S_t = \{r_t\}$ as the resource set fetched by the browser at time t under the condition that the browser cache is empty. $r_t \in < URL, Content >$ is a resource entry, where URL (including the domain, port, path, and query) is used to identify the resource, and $Content$ is its actual entity. A resource can be updated when time goes on. S_t represents the up-to-date resources of the website at time t.

We define

$$CA_t(\Delta t) = \{r_t \in S_t | \exists r_{t-\Delta t} \in S_{t-\Delta t}, r_t.Content = r_{t-\Delta t}.Content\}$$

as the resources (at time t) that have already been loaded from the network at the previous visit before Δt. Note that we do not enforce whether the resource's URL is the same or not. We consider only the resource's content that is actually used by browsers to render websites. In fact, $CA_t(\Delta t)$ are the resources that can benefit from caching.

Based on the preceding definitions, cacheablity $\Psi(\Delta t)$ can be formulated as the proportion of $CA_t(\Delta t)$ in S_t:

$$\Psi(\Delta t) = \frac{CA_t(\Delta t)}{S_t}$$

$\Psi(\Delta t)$ measures what percentage of a website can be cached after Δt. Larger $\Psi(\Delta t)$ means more resources are cacheable. It should be pointed out that cacheablity depends on Δt because resource updates are different when a website is revisited after different Δt.

We define

$$C_t(\Delta t) = \{r_{t-\Delta t} \in S_{t-\Delta t} | r_{t-\Delta t} \text{is configured to be cached}\}$$

as the resource set in the browser cache at time t. $C_t(\Delta t)$ represents the resources stored in the cache on the previous visit at time $t - \Delta t$. Note that $C_t(\Delta t) \subseteq S_{t-\Delta t}$ because some resources in $S_{t-\Delta t}$ may be configured not to be stored in the browser cache.

According to the cache mechanism, $C_t(\Delta t) = EX_t \cup FR_t$, indicates that some of the cached resources are expired (denoted as EX_t) while others are still fresh (denoted as FR_t). The browser first loads a resource from the cache if its URL is found in FR_t, in which case we say the resource is *hit*. Define $H_t = \{r_t \in S_t | \exists r \in FR_t, r_t.URL = r.URL\}$ as the resource set loaded from the cache. Otherwise, no matter when the resource URL is found in EX_t or cannot be found in $C_t(\Delta t)$, the browser loads it from the network, in which case we say the resource is *miss*. Define $M_t = \{r_t \in S_t | (\exists r \in EX_t, r_t.URL = r.URL) \vee (\forall r \in C_t(\Delta t), r_t.URL \neq r.URL)\}$ as the resource set loaded from the network.

However, the cache policy may not be set properly so that expired resources could still be up-to-date. In addition, considering only the resource URL can bring mistakes because the content is what actually matters for rendering websites. Therefore, we further define positive and negative patterns for H_t and M_t, respectively.

Positive Hit ($PH_t(\Delta t)$). Given $r_t \in H_t$, $r_t \in PH_t(\Delta t) \iff \exists r_{t-\Delta t} \in S_{t-\Delta t}, r_t.URL = r_{t-\Delta t}.URL \wedge r_t.Content = r_{t-\Delta t}.Content$, which means that the resource is loaded from the cache and it is the same as the up-to-date version on the server. Larger $PH_t(\Delta t)$ means more resources are correctly served from the cache.

Negative Hit ($NH_t(\Delta t)$). Given $r_t \in H_t$, $r_t \in NH_t(\Delta t) \iff \exists r_{t-\Delta t} \in S_{t-\Delta t}, r_t.URL = r_{t-\Delta t}.URL \wedge r_t.Content \neq r_{t-\Delta t}.Content$, which means that the resource is loaded from the cache but its content has already been updated on the server. In this case, the browser should acquire the updated resource. Larger $NH_t(\Delta t)$ means more stale resources are used to render the website, and wrong behaviors are more likely to happen.

Positive Miss ($PM_t(\Delta t)$). Given $r_t \in M_t$, $r_t \in PM_t(\Delta t) \iff \exists r_{t-\Delta t} \in S_{t-\Delta t}, r_t.Content \neq r_{t-\Delta t}.Content$, which means that the resource is loaded from the network and its content has never occurred in $S_{t-\Delta t}$. It is either a new resource or an updated version of a cached resource. Larger $PM_t(\Delta t)$ indicates more resources are newly involved by the website or changed during the time.

Negative Miss ($NM_t(\Delta t)$). Given $r_t \in M_t$, $r_t \in NM_t(\Delta t) \iff \exists r_{t-\Delta t} \in S_{t-\Delta t}, r_t.Content = r_{t-\Delta t}.Content$, which means that the resource is loaded from the network but its content has ever been loaded by the browser before. It is either stored in the cache or configured not to be cached. In this case, the browser should directly use the previously loaded resource. Larger $NM_t(\Delta t)$ indicates more redundant transfers.

Based on the preceding definitions, we can formulate the actual cache performance $\Phi(\Delta t)$ as the proportion of $PH_t(\Delta t)$ in $PH_t(\Delta t) \cup NM_t(\Delta t)$:

$$\Phi(\Delta t) = \frac{PH_t(\Delta t)}{PH_t(\Delta t) \cup NM_t(\Delta t)}$$

TABLE 1
Cache performance metrics

Metric	Description
$PH_t(\Delta t)$	The proportion of resources that are correctly served by the cache when the website is revisited after Δt.
$NH_t(\Delta t)$	The proportion of resources that should have been fetched from the server but are actually provided by the cache when the website is revisited after Δt.
$PM_t(\Delta t)$	The proportion of resources that are correctly fetched from the server when the website is revisited after Δt.
$NM_t(\Delta t)$	The proportion of resources that should have been provided by the cache but are actually fetched from the server when the website is revisited after Δt.
$\Psi(\Delta t)$	Cacheability of a website with revisiting interval Δt, giving an upper bound of how much a website can benefit from caching.
$\Phi(\Delta t)$	Actual cache performance of a website with revisiting interval Δt, showing how well a website has utilized caching.
$\Omega(\Delta t)$	Negative Miss Ratio with revisiting interval Δt, indicating how much data traffics are wasted by redundant transfers.

In addition, we can measure the problems of redundant transfer by Negative Miss Ratio $\Omega(\Delta t)$:

$$\Omega(\Delta t) = \frac{NM_t(\Delta t)}{PM_t(\Delta t) \cup NM_t(\Delta t)}$$

Larger Ω indicates more resources loaded from the network can actually be loaded from the cache.

Table 1 summarizes the metrics and their meanings to measure mobile Web cache performance. Meanwhile, as we choose the resource size and number as the quantitative parameters, subscripts s and n are attached to the metrics representing which parameter is used to calculate the metric values. For example, Ψ_s and Ψ_n denote the cacheability calculated by the resource size and number, respectively.

5.2 Cache Behavior Taxonomy

In order to calculate the preceding metrics, we correlate the cache configurations and resource updates by building a cache behavior taxonomy (Figure 3).

When a webpage is visited, the browser looks up in its cache table for each resource represented by the URL.

If the URL is found in the cache table (*URLHit*), the cache mechanism is ready to work. The browser first checks expiration to determine the freshness of the cached resources. There are two expiration policies. One is the server-specified expiration relying on the origin server to provide an explicit expiration time. The other is the heuristic expiration relying on the browser to assign an estimated value for those without explicit server-specified expiration time.

For the heuristic expiration, no matter whether the resource content is changed or not, any of the four cache behavior patterns is possible to happen as its freshness cannot be explicitly determined. In the worst case, we can assume the *ContentChanged* state of heuristic expiration as NH and the *ContentNotChanged* state as NM. In the following discussion, all the analyses are for the worst case unless specially stated.

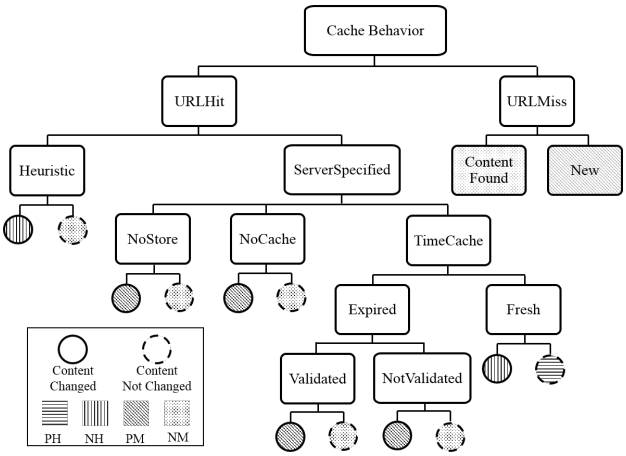


Fig. 3. Cache behavior taxonomy

For the server-specified expiration, there are three possible policies. The first is *NoStore*, indicating that the resources cannot be persistently stored on the hard disk. The second is *NoCache*, indicating that although the resources should not be cached by the browser, they can be persistently stored on the hard disk. The *ContentChanged* and *ContentNotChanged* states of both *NoStore* and *NoCache* are classified as PM and NM, respectively. In the third case, according to the expiration time, we can determine whether the cached resources get expired or stay fresh. If expired, two further branches *Validated* and *NotValidated* can be derived from the cache validation model, where the last modified time or Etag is used for the freshness check with the origin server. Combined with resource update states, we can classify each leaf node with the *ContentChanged* state as PM or *ContentNotChanged* state as NM. For the Fresh node, NH happens if the content is changed, while PH happens if the content is not actually changed.

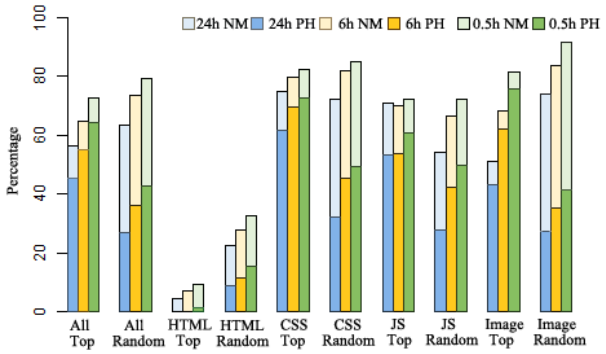
If the URL is not found in the cache table (*URLMiss*), it does not mean that the resource cannot be served by the cache. As aforementioned, resources with different URLs may have the same content. We use *ContentFound* to denote this case and classify its cache behavior pattern as NM. The last branch of *New* indicates completely new resources that have to be downloaded from the network, and thus we classify it as PM.

6 Measurement Results

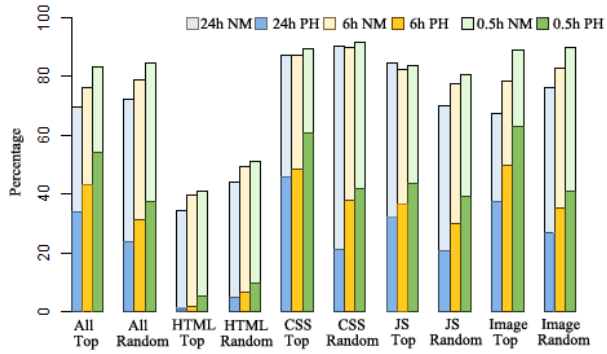
This section presents our detailed measurement results from three aspects: the gap between the ideal and the actual cache performance (Section 6.1), the problems caused by undesirable cache performance (Section 6.2), and the root cause analysis (Section 6.3).

6.1 Ideal and Actual Cache Performance

We first show the cacheability of websites and then present how much cacheability is realized in practice, uncovering the gap between the ideal and actual cache performance. Figure 4 depicts Ψ and Φ in a bar chart with different RIs and for different resource types. Based on our methodology, each bar represents the cacheability (denoted by Ψ) of corresponding resource types or all resources with the given RI, where the dark-color part stands for those actually cached (PH) resources and the light-color part represents resources that have not been cached (NM). Therefore, the actual performance (Φ) can be easily observed as the proportion which the dark-color part occupies in the whole bar.



(a) Ψ_s and Φ_s of aggregated websites and resource types



(b) Ψ_n and Φ_n of aggregated websites and resource types

Fig. 4. Ideal and actual cache performance

6.1.1 Potential Benefits of Caching

To answer the first research question of "*What percentage of Web resources can be cached ideally*", we calculate Ψ of the websites in the two data sets, shown as the height of each bar in Figure 4. We can make the following observations.

A high percentage of the total resources are cacheable and thus caching may be substantially helpful. For the Top websites, 56.4%-72.7% of the total resource size (Ψ_s) and 69.5%-83.2% of the total resource number (Ψ_n) are cacheable, depending on the values of RI. For the Random websites, the percentages are even higher: 63.2%-79.0% in Ψ_s and 72.1%-84.5% in Ψ_n. These results demonstrate that caching has great potential to help improve the performance of mobile Web browsing if caching is correctly configured and implemented. For the websites that users visit daily or even more frequently (e.g., news websites and search engines), caching can substantially reduce the cost of Web browsing on mobile devices, in terms of network bandwidth, computation, and energy. Furthermore, as RI increases, Ψ declines. This phenomenon is reasonable because resources are more likely to change when users revisit a website after a longer period and thus the probability of cache hit becomes lower.

Caching is more helpful for the Random websites than for the Top websites. No matter what value RI has, Ψ_s and Ψ_n of the Random websites are always larger than those of the Top websites. For example, when RI is 24 hours, Ψ_s of the Random set is 6.8% higher than Ψ_s of the Top set. In other words, the Random websites are more cacheable than the Top websites. The reason is that resources are less likely to change for ordinary websites than for popular websites.

Next, we examine how different types of resources can benefit from caching.

HTML is the least likely to benefit from caching. Interestingly, HTML is the least cacheable type of Web resources. For example, Ψ_s of HTML for the Top websites is as small as 4.5%-9.1%. This result may be due to that modern websites are becoming increasingly dynamic to meet the ever-growing user requirements. Rich client technologies, such as Asynchronous JavaScript + XML (AJAX) programming, can change HTML DOM trees on-the-fly and make HTML less cacheable. Ψ_n of HTML (34.2%-41.1% for the Top websites) is larger than Ψ_s of HTML, because small HTML pages are less likely dynamic. In addition, Ψ of HTML for the Random websites is larger than the one for the Top websites, indicating that the Random websites are relatively more stable than the Top websites.

CSS is the most cacheable resources. Compared to other types of resources, Ψ of CSS is the highest: 74.6%-89.4% for the Top websites, and 72.0%-91.4% for the Random websites. This observation can be explained by the nature of CSS: although the content of a webpage may change frequently, the layout and style of the webpage may be more stable.

JS of the Top websites benefits more than that of the Random websites. The cacheability of JS for the Top and Random websites is very different from overall results and other resource types. Given an RI, Ψ of JS for the Top websites is always larger than that of the Random websites. Furthermore, Ψ of JS for the Top websites does not change much as RI changes (70.9% to 72.3%), but the variance of Ψ of JS is rather large for the Random websites (54.1% to 72.1%). This observation reflects that the Top websites remain more stable on functionalities than the Random websites.

Images mostly benefit from caching for short RI. As RI increases, Ψ of images decreases a lot (about 30% for the Top websites and 20% for the Random websites). This phenomenon is reasonable since new images may take place of old ones after a longer time.

6.1.2 Actual Cache Performance

We further calculate Φ of the two data sets, to explore "*What percentage of the cacheable Web resources are actually cached*" for the websites that we captured, shown as the percentage of the dark-color part in each bar in Figure 4.

Compared to the Top websites, the Random websites do not make best use of caching. Φ of Random is always smaller than Φ of Top. More than 80% of the cacheable bytes for Top have been actually cached in all cases, but just above 50% of the cacheable bytes for Random have been actually cached in the best case within 0.5 hour RI.

Small-size resources are not taken into consideration very well. Although Φ_s of Top is high, Φ_n is relatively low. For 24 hours RI, Φ_s is above 80% while Φ_n is just not more than 50%. This observation implies that the Top websites pay more attention on larger-size resources but there are still a number of resources not being considered. They occupy small proportion in the total resource size but constitute the large proportion of the total resource number. Network connections may be wasted to download these small resources.

The cacheability of HTML is almost not utilized at all. Both Φ_s and Φ_n of HTML for Top are smaller than those of Random. In the worst case with 24 hours RI, Φ_s of Top

is just 2.8%. For Random, Φ_s is not above 50% in the best case with 0.5 hours RI. Generally, it seems not possible to make HTML cached as it can hardly be predicted when the HTML is changed. Therefore, popular websites give up the cacheability of HTML.

The resource type with the highest cache performance is image for Top (84%-93%) and JS for Random (51%-70%). It is surprising that images of the Random websites do not make the best use of cache, where both Φ_s and Φ_n are even not above 50% in the best cases.

In summary, there is a big gap between ideal and actual cache performance. For the Top websites, the gap is mostly originated from the large number of small-size resources. For the Random websites, all types of resources' cacheability have not been properly considered.

6.2 Problems Caused by Undesirable Cache Performance

We then identify two major problems caused by undesirable cache performance: *Redundant Transfer* introduced by NM and *Miscached Resource* introduced by NH. These problems can negatively affect the user experience of mobile Web browsing.

6.2.1 Redundant Transfer

The Redundant Transfer problem comes from NM when resources could be fetched from the cache but are actually downloaded from the network. Figure 5 shows the CDF of Ω_s and Ω_n of the Top and Random websites with different RIs.

Redundant Transfer is a significant issue for mobile Web browsing and the issue is worse for the Random websites than the Top ones. Considering 24 hours RI, the medium Ω_s of Top is about 10% while it reaches more than 40% for Random, i.e., 40% of the transferred bytes are redundant. The result indicates that Redundant Transfer is a common problem for mobile Web browsing while the Top websites indeed make some efforts to reduce it. However, in terms of the resource number, Ω_n of both Top and Random is relatively high where the medium value is about 50% and 75%, respectively.

Redundant Transfer is worse for short RI. Ω becomes higher as RI decreases for both Top and Random, represented by that CDF curves shift to the right when RI decreases in Figure 5. In the worst case, the medium Ω_s of Random is more than 80% with 0.5 hour RI. As a result, a larger proportion of resources requested from the network are redundant when users revisit a website after a shorter interval. However, as discussed earlier, caching is more effective when RI is shorter. Therefore, caching is far from its capacity and it could be improved a lot if NM was reduced.

Some websites have 100% Ω, indicating that all the requested resources are redundant for these websites. For example, about 10% of Top and 20% of Random have 100% Ω with 24 hours RI. Table 2 lists the websites of 100% Ω for Top considering 24 hours RI. It is surprising that some popular websites are in the list, such as Baidu, Wikipedia, and Apple. Since users may visit these popular websites many times every day, the waste of data traffic and energy could be very large.

We then examine the Redundant Transfer problem for different resource types. Figure 6 shows Ω_s's CDF with different RIs for both Top and Random, corresponding to the resource types of HTML, CSS, JavaScript, and Image.

For all kinds of resources, Ω of Random is larger than that of Top, represented by that CDF curves shift to the right bottom by comparing the two figures of the same resource type for Top and Random. This result implies that ordinary websites do not consider cache much for each kind of resources.

TABLE 2
Top websites with 100% Ω

Website	Rank	Category
Baidu	5	Search
wikipedia	6	News & Reference
Wordpress	20	Misc
360	22	Corporation
Soso	29	Search
Ask	35	Search
Instagram	37	Entertainment
Apple	45	Corporation
Alibaba	68	E-commerce

For HTML (6(a) and 6(e)), Ω of Top does not vary a lot while Ω of Random decreases as RI increases. This observation implies that HTML of the Top websites is more dynamic while HTML of the Random gets updated in a fixed period.

For CSS (6(b) and 6(f)), when RI increases, Ω of both Top and Random increases, implying that more redundant transfers could occur after a longer RI. The reason may be that both Top and Random leverage the cache expiration model for CSS but the expiration time is too conservative. As a result, when the CSS is revisited after a shorter interval, it is served by the cache. In contrast, the CSS is loaded from the network when the page is revisited after a longer interval, but the resource is actually not changed. It is also interesting that CDF of CSS is a traverse line, indicating a polar effect that some websites' CSS are no NM while others are all NM.

For JS (6(c) and 6(g)), Ω of both Top and Random does not vary so much as RI changes, indicating that there are a fixed set of NM in JS.

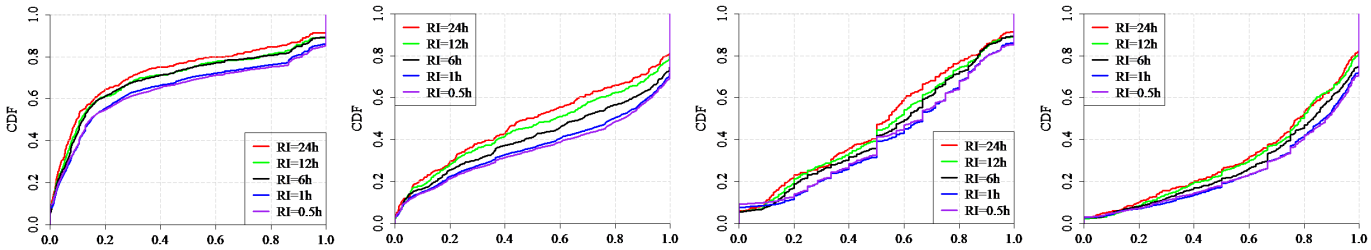
For image (6(d) and 6(h)), when RI increases, the Ω decreases. Random decreases in a small margin while Top decreases larger, implying that images of the Top websites change more frequently than the Random websites.

6.2.2 Miscached Resource

Due to the expiration model, it is likely to provide the out-of-date resources from the browser cache. In most cases, developers expect that users would tolerate these miscached resources such as acquiring information that is not the latest. However, in some circumstances, miscached resources could lead to incorrect functionalities. For example, during the evolution of Baidu, it changes many JS and CSS resources to improve its user interaction. However, some previous JS and CSS resources are configured to a considerable long expiration period. Therefore, users are even not able to use Baidu's search functionality until the users clear the browser cache, or force to refresh the page, or wait until the expiration time comes. It is worth exploring the Miscached Resource problem. In fact, NH measures the extent of this problem. Table 3 shows the result of NH.

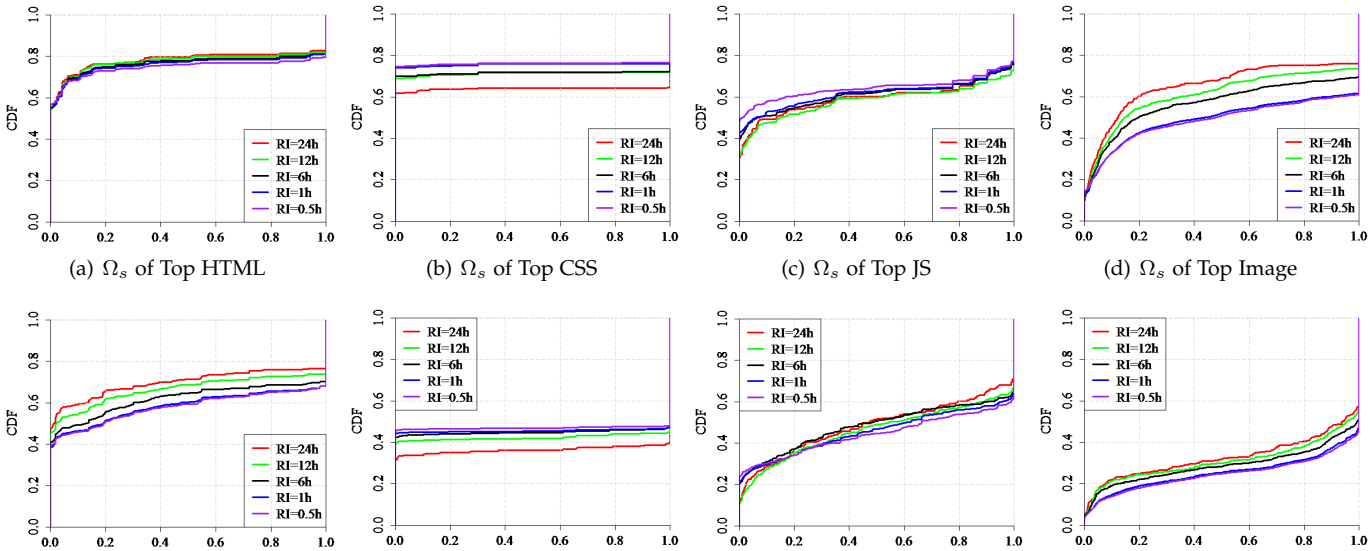
NH takes a small proportion but we cannot ignore it. NH of Top and Random share a similar proportion. Although NH of the resource number is no more than 3%, NH of the resource size is about 7%. Note that NH of Top and Random share a similar proportion, implying that both Top and Random have not considered much for NH.

NH of HTML is relatively large. NH introduces a lot of expired content. Developers may think that users could



(a) Ω_s of Top (b) Ω_s of Random (c) Ω_n of Top (d) Ω_n of Random

Fig. 5. Overall Negative Miss Ratio



(a) Ω_s of Top HTML (b) Ω_s of Top CSS (c) Ω_s of Top JS (d) Ω_s of Top Image

(e) Ω_s of Random HTML (f) Ω_s of Random CSS (g) Ω_s of Random JS (h) Ω_s of Random Image

Fig. 6. Ω_s of different resource types for Top and Random

TABLE 3
NH of aggregated websites and resource types

	24h		0.5h	
	NH_s	NH_n	NH_s	NH_n
All_{Top}	6.4%	2.3%	8.1%	3.0%
All_{Random}	7.7%	2.9%	9.0%	2.8%
HTML_{Top}	35.7%	18.7%	40.0%	19.9%
HTML_{Random}	24.6%	10.3%	21.5%	9.1%
CSS_{Top}	8.3%	5.6%	12.1%	7.7%
CSS_{Random}	13.5%	2.8%	10.5%	2.9%
JS_{Top}	4.8%	2.1%	10.4%	4.1%
JS_{Random}	10.5%	4.3%	16.6%	5.3%
Image_{Top}	0.1%	0.3%	0.1%	0.2%
Image_{Random}	1.0%	1.3%	0.5%	0.8%

bare the expired content so that temporally expired content is also acceptable.

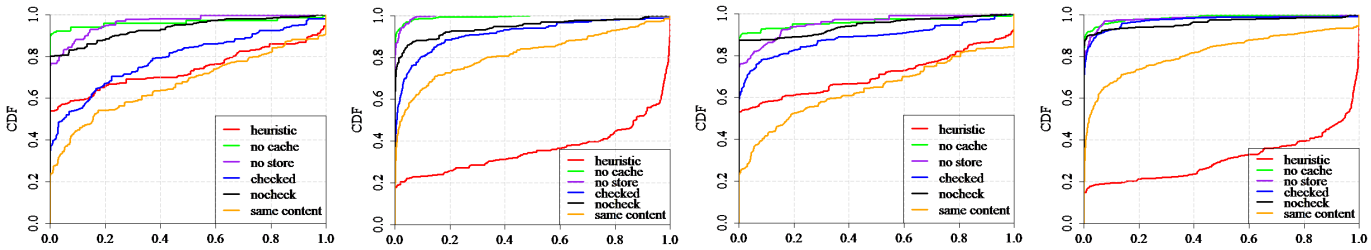
NH of image is the lowest but not zero because some functional images (such as borders and buttons) may change when websites get updated.

6.3 Analysis of Root Causes

To study the third research question "*What causes the gap between ideal and actual performance*", we investigate the cache behaviors belonging to the NH and NM patterns, finding misconfigurations that lead to undesirable performance. For NH, according to the taxonomy in Figure 3, two states, Heuristic and Fresh, are related with NH. By calculating the proportion of these two states in the total NH, we find that Heuristic accounts for 60% and Fresh is 40%. As a result, if all resources are assigned a proper expiration time, then the proportion of NH resources could be reduced. However, it is not possible to eliminate NH resources under the current time-based expiration policy. Although longer expiration time results in NH, shorter expiration time can bring NM as well. Therefore, it could be better if new URLs are assigned to changed resources.

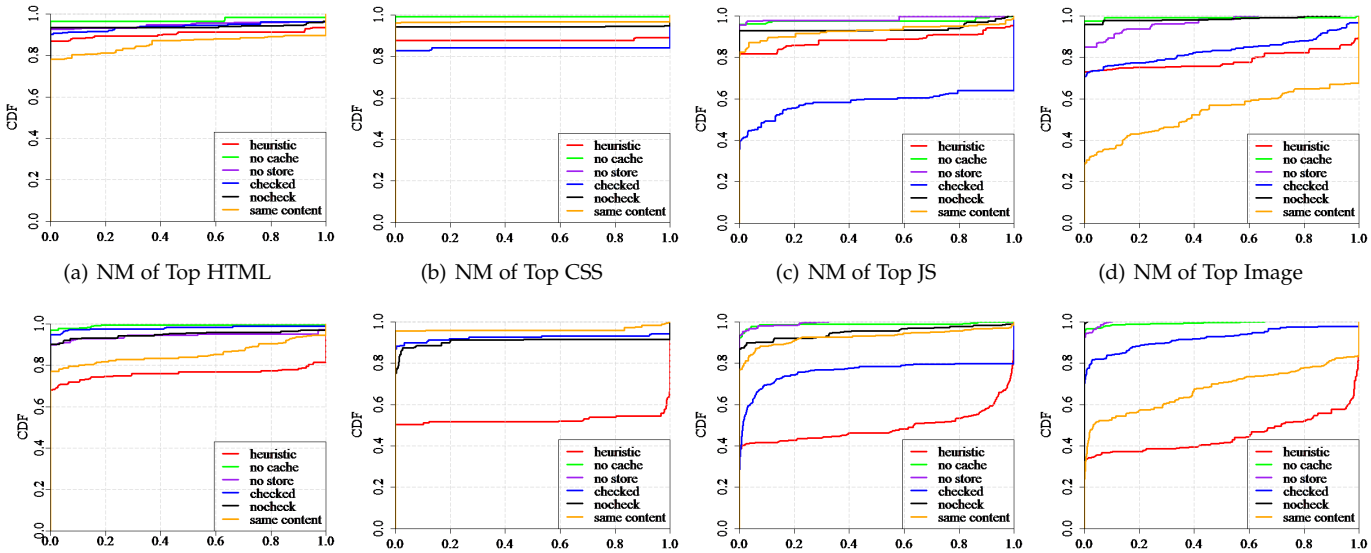
For NM, we break down the cache behaviors of NM and plot the CDF of each case in Figure 7. As explained in Section 5.2, there are six cache behaviors belonging to the NM pattern. We calculate the proportion of each cache behavior in the total NM and plot them on a CDF graph. Figure 7 shows the cases of the resource size for Top and Random with 24 hours and 0.5 hours RI.

Heuristic and Same Content are the top two causes to NM for both Top and Random. For Top, Same Content is the most significant issue. For Random, Heuristic is the most significant issue while Same Content is in the second place. The distributions of Heuristic and Same Content are not affected by RI. For Heuristic, as we consider the worst case, RI is not an influencing factor and thus it occupies the same proportion in each RI. For Same Content, the distribution is



(a) Proportion (Top, 24h) (b) Proportion (Random, 24h) (c) Proportion (Top, 0.5h) (d) Proportion (Random, 0.5h)

Fig. 7. Breakdown of NM for all resources



(a) NM of Top HTML (b) NM of Top CSS (c) NM of Top JS (d) NM of Top Image

(e) NM of Random HTML (f) NM of Random CSS (g) NM of Random JS (h) NM of Random Image

Fig. 8. Breakdown of NM for different resource types

similar because the same resources have different URLs at every visit.

Checked is the third reason for NM. Checked happens when the resources are expired in the cache (but actually are up-to-date on the server) and validated with the original server by last-modified time or Etag identifiers. Although the validation is successful to eliminate sending a full response, the protocol overheads account for some redundant size. Therefore, validating each resource's status may be harmful for mobile Web browsing. The reason of Checked is that the expiration time is configured too short.

No-store and No-cache occupy small proportion, indicating that developers are careful when configuring resources to no store or no cache. Therefore it is less important to consider such cases to improve mobile Web cache performance.

Figure 8 shows the breakdown of NM for different resource types with 24 hours RI. For HTML, just like the total resources, Heuristic and Same Content are the top two causes. But the No-check and No-store have higher percentages than Check. This observation reflects that developers are less likely to use a validation mechanism for HTML because HTML could always be changing for new contents. For CSS and JS, Checked takes a higher proportion than the other resource types as well as comparing to the total resources. Checked is even the most significant issue of Top CSS and Top JS. For CSS, Same Content is not an important issue, occupying a rather small proportion. Image has the most similar distribution with the total resources. This result is reasonable since images dominate the largest share of the total resource size.

We next deeply analyze Heuristic Expiration, which is the main source for both NH and NM, as well as Same Content and Conservative Expiration Time, which are the main sources of NM.

6.3.1 *Heuristic Expiration*

We first investigate the Heuristic Expiration issue, which often happens when developers are unaware of the importance of cache and do not explicitly configure cache parameters. Heuristic Expiration is an important issue for the Random websites because their developers may be unskilled. However, Heuristic Expiration also takes the second place in the Top websites.

Table 4 shows the proportion of Heuristic Expiration resources as well as the average updating cycle (the interval between two updates). Totally, 6% and 33.7% of Top and Random resources are configured as Heuristic Expiration. For Top, HTML is the largest (37.4%). For Random, all proportions are above 20%, meaning that more than 1/5 of resources have not explicitly utilized the caching mechanism.

Since different Web browsers have different algorithms to deal with Heuristic Expiration, we just consider the worst case in the previous analysis. Here we calculate the average updating cycle to study what is the proper expiration time for these resources. We can see that there is a big difference

TABLE 4
Resources of Heuristic Expiration

	Top		Random	
	Pct.	Cycle	Pct.	Cycle
HTML	37.4%	3.7 h	29.9%	29 h
CSS	1.9%	1.0 h	37.3%	134 h
JS	3.8%	7.2 h	20.4%	143 h
Image	2.6%	1.0 h	45.2%	133 h
Others	2.6%	4.0 h	24.5%	90 h
Total	6.0%	5.9 h	33.7%	107 h

between Top and Random. Considering all the Heuristic Expiration resources, the cycle is 5.9 hours for Top while it is 107 hours for Random. This observation implies that resources configured as "Heuristic Expiration" can benefit a lot from caching if their expiration time is set to day-level. In contrast, the proper expiration time for Heuristic Expiration resources of Top is hour-level. Considering different resource types, JS has the longest updating cycle for both Top (7.2 hours) and Random (143 hours). HTML has the shortest cycle for Random (29 hours).

6.3.2 Same Content

Next, we study the Same Content issue, as it has a significant influence on both Top and Random. As we described earlier, Same Content represents a case that a resource's content is found in the cache but the URL is different. Since the HTTP specification regards the URL as the index of resources for caching, there is no means to use the cached resource if the URL cannot be found in the cache. By examining the resource request records, we find that there are two major causes of Same Content.

```
(function() {
  var APC_test_rand = Math.floor(Math.random() * 10000);
  if (APC_test_rand < 10000) {
    setTimeout (function() {
      var s = document.createElement("script");
      document.getElementByTagName('head')[0].appendChild(s
          );
      s.src = "http://jqmt.qq.com/cdn_dianjiliu.js?a=" +
          Math.random();
    }, 5000);
  }
})();
```

Listing 1. An example of random strings attached to URLs leading to Same Content

Version management of resources. In order to provide end users the up-to-date resources, some best practices of Web development suggest that developers attach random query strings to URLs. Listing 1 shows an example of such random strings. It is a piece of JS code whose function is to load another JS from the URL "http://jqmt.qq.com/cdn_dianjiliu" (Lines 5-7). A random string generated by *Math.random()* is attached to the URL. Therefore, every time the webpage with the resource's reference is visited, a new query string is generated and the browser treats the resource as a new one to download from the server. In other cases, some websites, such as Youku, use different directories to manage resource versions. Each time the website gets updated, a new directory with the new version number is created. Since an update is not likely to change all the resource files, Same Content happens if those unmodified resources are requested from the new directory.

Web hierarchy architecture. Figure 1 has shown the Web hierarchy architecture that there are middle-boxes to improve the Web performance such as CDN and reverse proxy. To leverage these middle-boxes, Web servers have to generate different URLs or attach hash code of route paths to resources. As a result, one resource could have different URLs when requested at different times or at different places, leading to Same Content.

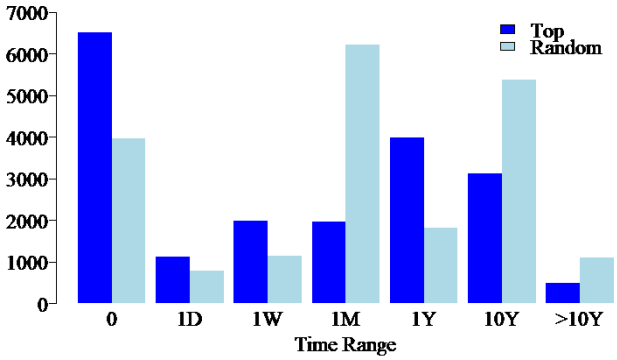


Fig. 9. Expiration time distribution of Top and Random

6.3.3 Conservative Expiration Time

Figure 9 shows the resource expiration time (indicated by Max-age primitive) distribution. 40% and 24% of the resources for Top and Random, respectively, are configured by less than one-day expiration time. This result implies that the Top websites are more conservative on expiration because their resources change more frequently than the Random websites. However, we find that as many as 89% and 86% resources with less than one-day expiration time have no updates for Top and Random, respectively. Therefore, the expiration time can be extended longer. However, longer expiration time may introduce the Miscached Resource problem. As a result, it is actually a tradeoff between website functionality and cache performance.

7 Implications

Table 5 summarizes the findings of our measurement results. Based on these findings, we can draw several implications.

• **Cache mechanisms.** According to Finding 1, caching has substantial benefits for mobile Web browsing, especially for ordinary websites. But the cacheability has not been effectively exploited. There exists a big gap between the ideal and actual cache performance. According to Finding 3, data traffic of mobile Web browsing is wasted mostly for frequently visited websites due to Negative Miss resources. According to Finding 4, miscached resource is a common problem for all the websites due to the current time-based expiration mechanism. Popular websites may have wrong layout and view while ordinary websites could suffer from wrong functionalities.

• **Web administrators**. According to Finding 1, administrators from popular websites indeed make efforts on caching large-size resources but do not consider small-size resources very well. Administrators from ordinary websites perform worse for both cacheable bytes and objects. According to Finding 2, Web administrators should configure different caching policies for different types of resources based on the characteristics of websites. Administrators from popular websites could consider more for caching JS, while administrators from ordinary websites should check the cache configurations for all resources especially the images.

TABLE 5
Summary of findings

No.	Findings from Measurement Study
1	More than half of the total resource bytes and about 70% of the resource objects can be cached after 24 hours. Although 80% of the cacheable bytes are actually cached for Top, the percentage of cached objects is under 50%. For Random, both cached bytes and objects occupy not more than 50% of the cacheability.
2	The cacheability varies for different types of resources, and the actual performance is also different. The results between Top and Random are not the same either.
3	The Redundant Transfer problem is worse when a website is revisited after a shorter period. Some popular websites even have 100% Negative Miss Ratio.
4	NH of Top and Random share a similar proportion. HTML accounts for the largest NH. NH of CSS for Top is larger than for Random, while JS is on the contrary. NH of images is not 0.
5	Same Content accounts for the main source of NM. More than 20% of NM resources are resulted from Same Content for half of Top and 25% of Random. Same Content is caused by the version management and Web hierarchy architecture.
6	6% and 33.7% of Top and Random resources are configured as Heuristic Expiration, but the average updating cycle is 5.9 hours and 107 hours for Top and Random, respectively.
7	40% and 24% of the resources for Top and Random are configured by less than one-day expiration time. But 89% and 86% of them have no updates for Top and Random, respectively.

According to Finding 5, although the Web hierarchy architecture reduces the overall network traffic and releases server workload, it is harmful for user-centric mobile Web browsing. End-user based content distribution may help solve the problem. According to Finding 6, it is better to set an explicit expiration time than using the Heuristic Expiration. Ordinary websites can set day-level expiration time to their resources while popular websites can set hour-level time. According to Finding 7, administrators can extend the expiration time for some resources that are not likely to change for a considerably long time. But longer expiration time could cause the Miscached Resource problem.

• **Web developers**. According to Findings 3 and 5, current techniques to maintain website versions, such as random query strings, are not appropriate for mobile Web browsing. New techniques have to be adopted such as the HTML5 AppCache interface [21]. A manifest file can be attached to each webpage, specifying what to cache and what to load from the network. Resources in the cache list are always loaded from the cache until the manifest file is updated.

Our results are derived from only the Chrome browser in this article. However, the measurement approach itself can be generalizable to other Web browsers such as Safari and FireFox, and hybrid apps that are built atop browser engines such as WebKit. It would be interesting to explore the "imperfectness of client-side cache mechanisms on other browsers."

8 Threats to Validity

Considerable care and attention should be addressed to ensure the rigor of this measurement study. However, as with any chosen research methodology, it is hardly without limitations. In this section, we present threats to validity in our study.

A threat to validity is the size of our dataset. In our dataset, we have the 51 websites from Alexa's Top 100 list and the 91 websites randomly chosen from Alexa's Top 1,000,000 list. The total number of websites may not be large enough to represent the average situation, leading to conclusions that may not be generalizable. However, since we intend to conduct a comparative study, it is essential to choose representative websites for the two counterparts. We assume that the more popular a website is, the more effort could be taken to optimize its cache performance. Therefore, it is reasonable to consider that the websites from Top 100 show the state-of-the-art cache performance.

We consider only the home page (or the landing page) for each website. The home page is the first view that users can see when they visit a website. So Web developers have to make a lot of efforts to optimize their home page for fast and efficient page load. In addition, resources of the home page are likely to have a large convergence with those in subsequent pages, which are mostly designed to be consistent in page style with the home page. Therefore, the cache performance of the home page can also reflect the status of other pages.

Our study excludes the websites transferred via the HTTPS protocol. However, it is a big trend that more and more websites, especially their mobile versions, are being migrated from HTTP to HTTPS for enhancing security. Excluding HTTPS may neglect some valuable findings. At the time when we collected the data, we filtered out only 4 websites from Top (Facebook, Twitter, Linkedin, and Netflix) and 3 websites from Random (Delta, Smartrecruiters, and Polarpersonaltrainer) that are transmitted via HTTPS. So our current results are not significantly influenced by such exclusion. One ongoing plan is to compare the differences of cache performance between HTTP and HTTPS.

Our data collection lasted for only one week so that we can report the cache performance with only less than 24-hour revisiting interval. The reason is that fewer samples can be used for longer revisiting interval, given the fixed duration. For example, we collect the traces of each website every 0.5 hour and continue the collection for one week. Therefore, the number of samples used to analyze cache performance with 0.5 hour revisiting interval is 2*24*7=336, while there are only 7 samples that could be used for 24 hour revisiting interval. Fewer samples could lead to bias of our findings. The limited duration also prevents us from analyzing revisiting interval longer than 1 day, which may be more general for the Random websites.

Our measurement targets at the client-side cache mechanism, so we assume that the proxy used in our framework can intercept all resources fetched from Web server. It is possible to obtain different measurement results by using other proxies than Charles. We plan to compare the impacts introduced by different proxies in future work.

9 Related Work

The combination between mobile computing and the Web has raised new research questions and opportunities. On one hand, measurement studies are conducted to profile

the performance of mobile Web, aiming to understand the latency, data usage, and energy consumption when surfing the Web on mobile devices. On the other hand, researchers invest efforts to improve the performance of mobile Web with novel techniques from different layers and tiers.

We next discuss related research on general mobile Web performance and then present related work on mobile Web cache specifically.

9.1 Mobile Web Performance

Measurement studies have been conducted to understand mobile Web performance. Huang *et al.* [22] measured mobile Web browsing performance. Wang *et al.* [23] examined the issues specific to mobile Web browsers. Mobile HTTP Archive [24] records mobile Web performance information of about 5000 mobile websites. But its recording period is 15 days, and is too coarse-grained to analyze the cache performance. Papapanagiotou *et al.* [25] compared smartphone and laptop Web traffic based on a three-week-long wireless communication trace collected in an enterprise environment. Thiagarajan *et al.* [26] measured the precise energy used by a mobile browser to render webpages.

Many techniques were proposed by researchers and employed by mobile developers to improve mobile Web performance. Jones *et al.* [27] studied how browser parallelization could help mobile browsers reduce latency and improve energy efficiency. Dong *et al.* [28] implemented a color adaptive Web browser Chameleon that renders webpages with power-optimized color schemes under user-supplied constraints to reduce energy consumption of mobile Web browsing. Lymberopoulos *et al.* [29] leveraged prefetching techniques to load resources based on user behaviors with server support to reduce latency. Some research work targeted at leveraging proxies or cloud to improve the performance of mobile Web browsing, such as VMP [30], FlyWheel [31], and Klotski [32]. Some latest commodity browsers such as Amazon Silk Browser [33], Google Chrome Beta [34], and OperaMini [35] are also designed for mobile browsing by leveraging server or cloud-based offloading. But the cloud-based mobile Web browsing is not always helpful [36]. New protocols, such as HTTP2 [37], SPDY [38], [39] and QUIC [40], are designed and deployed to optimize mobile Web performance from lower levels.

9.2 Mobile Web Cache

A lot of efforts have been invested to study the performance of mobile Web cache. Wang *et al.* [9] examined three client-only solutions to accelerate mobile browser speed: caching, prefetching, and speculative loading, by using Web usage data collected from 24 iPhone users over one year. They found that caching has very limited effectiveness: 60% of the requested resources are either expired or not in the cache. Zhang *et al.* [41] performed a comprehensive measurement study on Web caching functionality of 1300 top ranked Android apps, not just Web browsers. Results revealed that imperfect web caching is a common and serious problem for Android apps generating Web traffic. They also implemented a system-wide service called CacheKeeper to effectively reduce overhead caused by poor Web caching of mobile apps. Qian *et al.* [7] conducted a measurement study on Web caching in smartphones. By examining a one-day smartphone Web traffic dataset collected from a cellular carrier and a five-month Web access trace collected from 20 smartphone users, the study revealed that about 20% of the total Web traffic examined is redundant due to imperfect cache implementations. In their subsequent work [8], they studied caching efficiency for the most popular 500 websites. They found that caching is poorly utilized for many mobile sites. 26% of the top mobile sites mark their main HTML files as non-storable, leading to potential bandwidth waste. About 23% of objects in mobile sites are cacheable but have freshness lifetime of less than 1 hour. Our work differs from existing studies in our methodology: we use long-period sampling traces and determine the cache efficiency by resource evolution history. Wang *et al.* [42] examined how Web browsing can benefit from micro-cache that separately caches layout, code, and data at a fine granularity. They studied how and when these resources are updated and found that layout and code that block subsequent object loads are highly cacheable.

10 Conclusion

In this article, we have presented a comprehensive measurement and analysis of mobile Web cache performance. By proposing a proactive approach and an analysis model, we found that a big gap existed between the actual performance and ideal performance of client-side cache. We summarized three main causes of the observations, i.e., Same Content, Heuristic Expiration, and Conservative Expiration Time. We provided recommendations for developers to improve mobile Web cache performance according to the implications of our findings.

In future work, we plan to compare the differences of client-side cache performance between mobile and desktop website versions when accessing them on mobile devices. We also plan to study how to adaptively configure the cache by predicting when a resource is going to be updated based on its changing history and user behaviors.

Acknowledgment

This work was supported by the High-Tech Research and Development Program of China under Grant No.2015AA01A203, the Natural Science Foundation of China (Grant No. 61370020, 61421091, 61222203), and the Key Program of Ministry of Education, China under Grant No.313004. Tao Xie's work was supported in part by National Science Foundation under grants no. CCF-1349666, CCF-1409423, CNS-1434582, CCF-1434590, CCF-1434596, CNS-1439481, and CNS-1513939.

References

[1] Y. Zhu and V. J. Reddi, "Webcore: Architectural support for mobile web browsing," in *Proc. of ISCA'14*, 2014, pp. 541–552.
[2] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "Demystifying page load performance with WProf," in *Proc. of NSDI'13*, 2013, pp. 473–486.
[3] D. H. Bui, Y. Liu, H. Kim, I. Shin, and F. Zhao, "Rethinking energy-performance trade-off in mobile web page loading," in *Proc. of MobiCom'15*, 2015, pp. 14–26.
[4] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "Characterizing radio resource allocation for 3G networks," in *Proc. of IMC'10*, 2010, pp. 137–150.
[5] J. Erman, A. Gerber, M. Hajiaghayi, P. Dan, S. Sen, and O. Spatscheck, "To cache or not to cache: The 3G case," *IEEE Internet Computing*, vol. 15, no. 2, pp. 27–34, 2011.
[6] M. Roughan, R. Chang, F. Qian, J. Huang, J. Erman, Z. M. Mao, S. Sen, and O. Spatscheck, "How to reduce smartphone traffic volume by 30%?" in *Passive and Active Measurement*, 2013, pp. 42–52.
[7] F. Qian, K. S. Quah, J. Huang, J. Erman, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "Web caching on smartphones: ideal vs. reality," in *Proc. of MobiSys'12*, 2012, pp. 127–140.
[8] F. Qian, S. Sen, and O. Spatscheck, "Characterizing resource usage for mobile web browsing," in *Proc. of MobiSys'14*, 2014, pp. 218–231.
[9] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie, "How far can client-only solutions go for mobile browser speed?" in *Proc. of WWW'12*, 2012, pp. 31–40.

[10] Y. Ma, X. Liu, S. Zhang, R. Xiang, Y. Liu, and T. Xie, "Measurement and analysis of mobile web cache performance," in *Proc. of WWW'15*, 2015, pp. 691–701.
[11] "RFC 2616." [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616.txt
[12] P. Cao and C. Liu, "Maintaining strong cache consistency in the world wide web," *IEEE Transactions on Computers*, vol. 47, no. 4, pp. 445–457, 1998.
[13] J. Wang, "A survey of web caching schemes for the Internet," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 5, pp. 36–46, 1999.
[14] G. Barish and K. Obraczke, "World wide web caching: trends and techniques," *IEEE Communications magazine*, vol. 38, no. 5, pp. 178–184, 2000.
[15] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. C. Mogul, "Rate of change and other metrics: a live study of the world wide web," in *Proc. of USITS'97*, 1997.
[16] G. Cao, "A scalable low-latency cache invalidation strategy for mobile environments," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 5, pp. 1251–1265, 2003.
[17] K. Fawaz and H. Artail, "DCIM: distributed cache invalidation method for maintaining cache consistency in wireless mobile networks," *IEEE Transactions on Mobile Computing*, vol. 12, no. 4, pp. 680–693, 2013.
[18] K. Mershad and H. Artail, "SSUM: smart server update mechanism for maintaining cache consistency in mobile environments," *IEEE Transactions on Mobile Computing*, vol. 9, no. 6, pp. 778–795, 2010.
[19] "Charles proxy." [Online]. Available: http://www.charlesproxy.com/
[20] "Alexa." [Online]. Available: http://www.alexa.com/
[21] "HTML5." [Online]. Available: http://www.w3.org/TR/html5/
[22] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl, "Anatomizing application performance differences on smartphones," in *Proc. of MobiSys'10*, 2010, pp. 165–178.
[23] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie, "Why are web browsers slow on smartphones?" in *Proc. of WWW'11*, 2011, pp. 91–96.
[24] "Mobile HTTP archive." [Online]. Available: http://mobile.httparchive.org/
[25] I. Papapanagiotou, E. Nahum, and V. Pappas, "Smartphones vs. laptops: comparing web browsing behavior and the implications for caching," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1, pp. 423 – 424, 2012.
[26] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh, "Who killed my battery?: analyzing mobile browser energy consumption," in *Proc. of WWW'12*, 2012, pp. 41–50.
[27] C. G. Jones, R. Liu, L. Meyerovich, K. Asanovic, and R. Bodik, "Parallelizing the web browser," in *Proc. of HotPar'09*, 2009, pp. 7–7.
[28] M. Dong and L. Zhong, "Chameleon: a color-adaptive web browser for mobile OLED displays," *IEEE Transactions on Mobile Computing*, vol. 11, no. 5, pp. 724–738, 2012.
[29] D. Lymberopoulos, O. Riva, K. Strauss, A. Mittal, and A. Ntoulas, "PocketWeb: instant web browsing for mobile devices," in *Proc. of ASPLOS'12*, 2012, pp. 1–12.
[30] B. Zhao, B. Tak, and G. Cao, *Mobile Web Browsing Using the Cloud*, ser. Springer Briefs in Computer Science. Springer, 2014.
[31] V. Agababov, M. Buettner, V. Chudnovsky, M. Cogan, B. Greenstein, S. McDaniel, M. Piatek, C. Scott, M. Welsh, and B. Yin, "Flywheel: Google's data compression proxy for the mobile web," in *Proc. of NSDI'15*, 2015, pp. 367–380.
[32] M. Butkiewicz, D. Wang, Z. Wu, H. V. Madhyastha, and V. Sekar, "Klotski: Reprioritizing web content to improve user experience on mobile devices," in *Proc. of NSDI'15*, 2015, pp. 439–453.
[33] "Amazon Silk browser." [Online]. Available: http://aws.amazon.com/cn/documentation/silk/
[34] "Google Chrome beta browser." [Online]. Available: https://www.google.com/chrome/browser/beta.html
[35] "Opera Mini browser." [Online]. Available: http://www.opera.com/mobile/mini/iphone
[36] 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. of HotMobile'14*, 2014, pp. 1 – 6.
[37] "HTTP2." [Online]. Available: https://http2.github.io/
[38] "SPDY protocol - draft 3." [Online]. Available: http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3
[39] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How speedy is SPDY?" in *Proc. of NSDI'14*, 2014, pp. 387–399.
[40] "Experimenting with QUIC." [Online]. Available: http://blog.chromium.org/2013/06/experimenting-with-quic.html
[41] Y. Zhang, C. Tan, and L. Qun, "CacheKeeper: a system-wide web caching service for smartphones," in *Proc. of UbiComp'13*, 2013, pp. 265–274.
[42] X. S. Wang, A. Krishnamurthy, and D. Wetherall, "How much can we micro-cache web pages?" in *Proc. of IMC'14*, 2014, pp. 249–256.

Xuanzhe Liu is an Associate Professor in the School of Electronics Engineering and Computer Science, Peking University, Beijing, China. He is also with Beida (Binhai) Information Research. His research interests are in the area of services computing, mobile computing, web-based systems, and big data analytic. He is the corresponding author of this article.

Yun Ma is a Ph.D. student in the School of Electronics Engineering and Computer Science, Peking University, Beijing, China. His research interests include services computing and web engineering.

Yunxin Liu is a Lead Researcher in Microsoft Research, Beijing, China. His current research interests include mobile systems and networking.

Tao Xie is an Associate Professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign, USA. His research interests are software testing, program analysis, software analytics, software security, and educational software engineering. He is a senior member of IEEE.

Gang Huang is a Full Professor in Institute of Software, Peking University. He is also with Beida (Binhai) Information Research. His research interests are in the area of middleware of cloud computing and mobile computing. He is a member of IEEE.