Measurement and Analysis of Mobile Web Cache Performance

Yun Ma[1], Xuanzhe Liu[1*], Shuhui Zhang[1], Ruirui Xiang[1], Yunxin Liu[2], Tao Xie[3]
[1] School of Electronics Engineering and Computer Science, Peking University, Beijing, China
[2] Microsoft Research, Beijing, China
[3] University of Illinois at Urbana-Champaign, Urbana, USA
{mayun, liuxuanzhe, zhangshuhui, xiangrr0512}@pku.edu.cn; yunxin.liu@microsoft.com; taoxie@illinois.edu

ABSTRACT
The Web browser is a killer app 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, caching has been adopted as a key mechanism. However, the existing passive measurement studies cannot comprehensively characterize the performance of mobile Web caching. 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 paper, we present a proactive approach for a comprehensive measurement study on mobile Web 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 effective 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.

Categories and Subject Descriptors
C.4 [**Computer Systems Organization**]: Performance of Systems — *Design studies*

Keywords
Mobile Web; Cache; Measurement

1. INTRODUCTION
The Web browser is a killer and arguably the most frequently used app on mobile devices such as smartphones and tablet computers. According to a study [26], Web browsers occupy 63% of the window focus time on the subjects' mobile devices. However, the user experience of mobile Web browsing is far from satisfaction. Resource loading is one of the key factors influencing Web browsing performance. It is reported that 65% of page load time is spent on resource loading [21]. Such 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. Furthermore, more resource downloading may consume more energy, which is the scarcest resource on battery-powered mobile devices.

Caching is a widely adopted mechanism to accelerate Web resource loading[1]. By saving the resources of visited webpages into the local storage (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 visited again. As a result, caching reduces the quantity of downloaded resources and thus can improve Web browsing performance. Caching is particularly beneficial to mobile devices because it not only reduces page load time, but also reduces network traffic and energy consumption.

It is critical to understand how mobile Web browsing could benefit from caching and how well caching takes effect in practice. To this end, several measurement studies have been made by analyzing user access traces gathered from ISPs or instrumented client devices [11][15][16][17][24]. These studies have already revealed the performance problems of mobile Web caching caused by imperfect client-side implementations. However, they cannot comprehensively characterize the end-to-end Web cache performance involving not only Web clients but also Web servers and user behaviors. In particular, existing measurement studies on mobile Web cache have the following three main limitations.

- ***These existing studies suffer from 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.
- ***These 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

* Corresponding author

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741114

[1] "Caching" and "cache" are often used interchangeably. Usually, "caching" denotes the mechanism defined by specifications, and "cache" represents the browser component that could store resources temporally. "Cache" is also used as a verb, denoting putting something in the cache. In this paper, caching and cache have the same meaning if not specifically stated.

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.

- ***These 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 data collection approaches 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 issues faced by the existing studies, in this paper, we present a proactive approach for comprehensive measurement and analysis of mobile Web cache performance. In our approach, we proactively crawl 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. Furthermore, 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 make 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 and less energy consumption, compared to a small number of large resources. 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, data traffic, and energy consumption.

This paper makes the following main contributions:

- ***A proactive approach to acquire resource update history.*** 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 Top 100 list. The other set (Random) comes from 100 websites randomly chosen from Alexa Top 1,000,000 list. We periodically crawl their resource update history for one week, forming the basis of our analysis.
- ***An analysis model for 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 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).

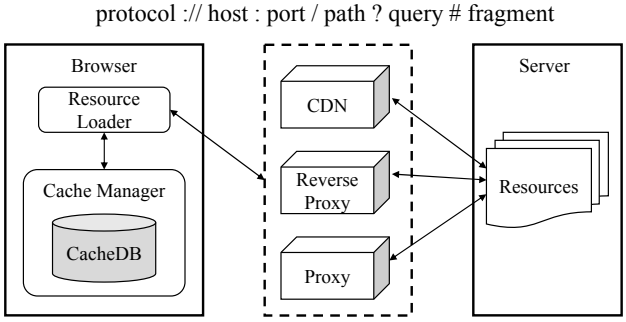


Figure 1. Architecture of Web caching

- ***Empirical results for showing 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.
- ***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.
- ***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).
- ***Implications and recommendations on how to improve cache performance by better cache policies.*** For example, the Application Cache of HTML 5 can be used to make the resource updates visible to clients.

The remainder of this paper is organized as follows. Section 2 motivates our study by analyzing how caching works and identifying three research questions. Section 3 describes our data set and data-collection approach. Section 4 presents the analysis methodology and Section 5 presents the detailed measurement results. Based on the findings, we provide implications and recommendations for Web developers in Section 6. Section 7 surveys related work and Section 8 concludes this paper.

2. BACKGROUND AND GOAL

In this section, we first describe the background on how Web caching works and then present the three research questions that we seek to answer, as the goal of this paper.

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 will load the target webpage by acquiring the corresponding resources from the Resource Loader module.

The Resource Loader will first try to load resources from the local cache through the Cache Manager module. According to RFC 2616 [9], 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 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 mechanism is called Heuristic Expiration.

The Cache Manager checks whether the requested resource (identified by its URL's protocol, host, port, path, and query) can be found in the Cache Database. If not, the browser will set 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 will be 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 decides the corresponding resource has not changed, it responds with a "*304 Not Modified*" message and the resource will be fetched from the cache of the client. Otherwise, the server sends the updated resource back to the browser.

It is worth mentioning 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 "multiple URLs of the same resource" problem causes undesirable Web cache performance and should be investigated.

2.2 Research Questions to Answer

Our goal 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 by 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.

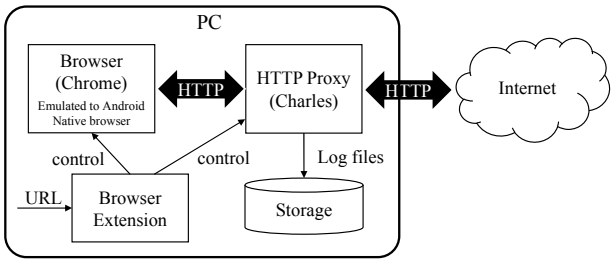


Figure 2. Data collection tool

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

3. DATA COLLECTION

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

3.1 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.

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 (Android 4.2 native browser in our case) so that a website will return 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, and more resources may be loaded by the JavaScript 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 programmability of the browser, which is not easy on Android smartphones.

We use the Charles proxy [3], 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 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 will regard that the website has finished loading. Note that some websites utilize lazy loading, and thus there will still be 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.

3.2 Data Sets

For a comparative study, we choose two sets of websites from Alexa [1] ranking list in January 2014. The first set (Top) contains Alexa 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 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 that we visited. Unreachable websites occurred in only the Random data set.
- 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 function 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. One week means that 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.

4. 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.

4.1 Metrics Definition

Cache only works 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.

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) identifies 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.

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.

Define

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

as the resource set in the browser cache at time t. $C_t(\Delta t)$ represents the resources stored into 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 will first load 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 will load 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, only considering the resource URL can bring mistakes as 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 $r_t \in H_t$, $r_t \in PH_t(\Delta t) \Leftrightarrow \exists r_{t-\Delta t} \in S_{t-\Delta t}, r_t.URL = r_{t-\Delta t}.URL \wedge r_t.Content = r_{t-\Delta t}.Content$, which means that the resource is loaded from the cache and it is the same with 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 $r_t \in H_t$, $r_t \in NH_t(\Delta t) \Leftrightarrow \exists r_{t-\Delta t} \in S_{t-\Delta t}, r_t.URL = r_{t-\Delta t}.URL \wedge r_t.Content \neq r_{t-\Delta t}.Content$, 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 $r_t \in M_t$, $r_t \in PM_t(\Delta t) \Leftrightarrow \forall r_{t-\Delta t} \in S_{t-\Delta t}, r_t.Content \neq r_{t-\Delta t}.Content$, 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 update of a

Table 1. Cache performance metrics

Metric	Description
$PH(\Delta t)$	The proportion of resources that are correctly served by the cache when the website is revisited after Δt.
$NH(\Delta t)$	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.
$PM(\Delta t)$	The proportion of resources that are correctly fetched from the server when the website is revisited after Δt.
$NM(\Delta t)$	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.

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 $r_t \in M_t$, $r_t \in NM_t(\Delta t) \Leftrightarrow \exists r_{t-\Delta t} \in S_{t-\Delta t}, r_t.Content = r_{t-\Delta t}.Content$, 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 $PH_t(\Delta t) \cup NM_t(\Delta t)$:

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

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 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 resource size and number as the quantitative parameters, subscript *s* and *n* will be 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 resource size and number, respectively.

4.2 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 web page is visited, the browser will look 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 will first check 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.

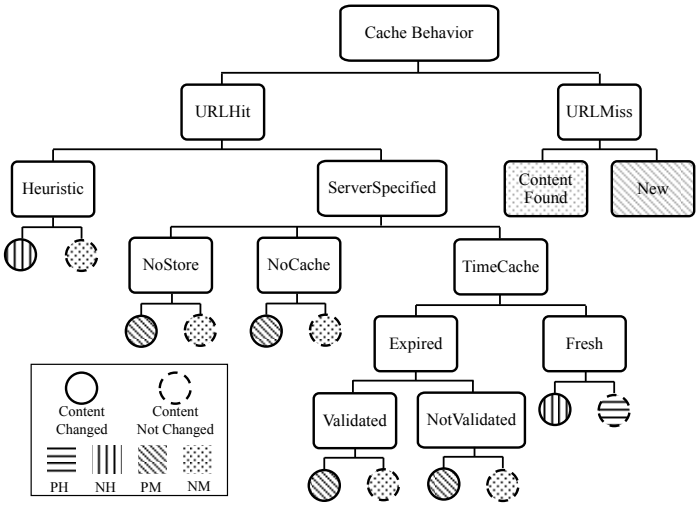


Figure 3. Cache behavior taxonomy

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 parts, all the analyses are for the worst case unless specially stated.

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 such 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.

5. MEASUREMENT RESULTS

This section presents our detailed measurement results from three aspects: the gap between the ideal and the actual cache performance (Section 5.1), the problems caused by undesirable cache performance (Section 5.2), and the root cause analysis (Section 5.3).

Table 2. Ψ of aggregated websites and resource types

	24h		6h		0.5h	
	Ψ_s	Ψ_n	Ψ_s	Ψ_n	Ψ_s	Ψ_n
All_{Top}	56.4%	69.5%	64.7%	76.0%	72.7%	83.2%
All_{Random}	63.2%	72.1%	73.5%	78.8%	79.0%	84.5%
$HTML_{Top}$	4.5%	34.2%	7.2%	39.6%	9.1%	41.1%
$HTML_{Random}$	22.4%	44.0%	27.8%	49.3%	32.7%	51.0%
CSS_{Top}	74.6%	86.9%	79.5%	86.9%	82.4%	89.4%
CSS_{Random}	72.0%	90.3%	81.7%	89.8%	84.8%	91.4%
JS_{Top}	70.9%	84.5%	69.7%	82.4%	72.3%	83.4%
JS_{Random}	54.1%	69.8%	66.6%	77.4%	72.1%	80.6%
$Image_{Top}$	51.0%	67.2%	68.3%	78.3%	81.4%	88.8%
$Image_{Random}$	74.0%	76.0%	83.7%	82.8%	91.4%	89.7%

Table 3. Φ of aggregated websites and resource types

	24h		6h		0.5h	
	Φ_s	Φ_n	Φ_s	Φ_n	Φ_s	Φ_n
All_{Top}	80.7%	48.8%	85.2%	56.6%	88.5%	64.9%
All_{Random}	42.4%	32.7%	48.9%	39.6%	53.8%	44.4%
$HTML_{Top}$	2.8%	3.8%	2.1%	4.9%	16.4%	12.9%
$HTML_{Random}$	39.4%	11.2%	40.9%	13.0%	47.2%	19.2%
CSS_{Top}	82.6%	52.7%	87.3%	55.9%	88.1%	67.7%
CSS_{Random}	44.6%	23.5%	55.2%	42.3%	57.9%	45.7%
JS_{Top}	74.9%	37.8%	76.9%	44.5%	83.7%	52.4%
JS_{Random}	51.6%	29.4%	63.2%	38.9%	69.2%	48.8%
$Image_{Top}$	84.5%	55.7%	90.8%	63.3%	93.0%	70.6%
$Image_{Random}$	36.6%	35.4%	41.9%	42.3%	45.2%	45.4%

5.1 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.

5.1.1 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, as shown in Table 2. From Table 2, we 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.

5.1.2 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. Table 3 shows the results and we have the following findings.

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, not being energy efficient.

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

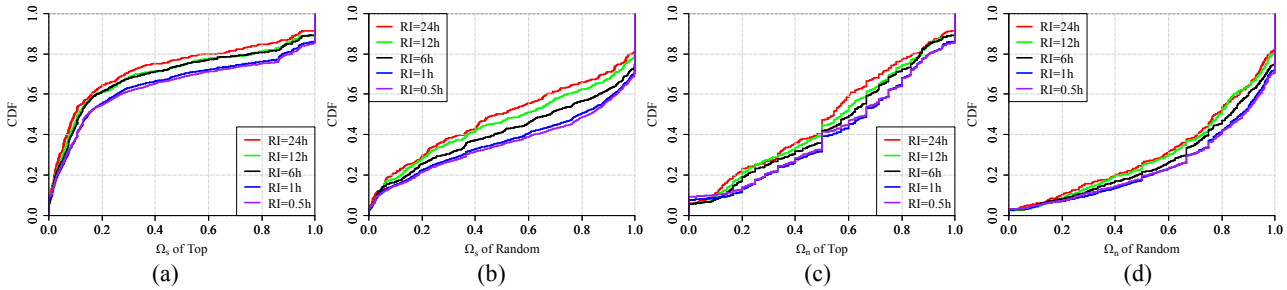


Figure 4. Overall Negative Miss Ratio

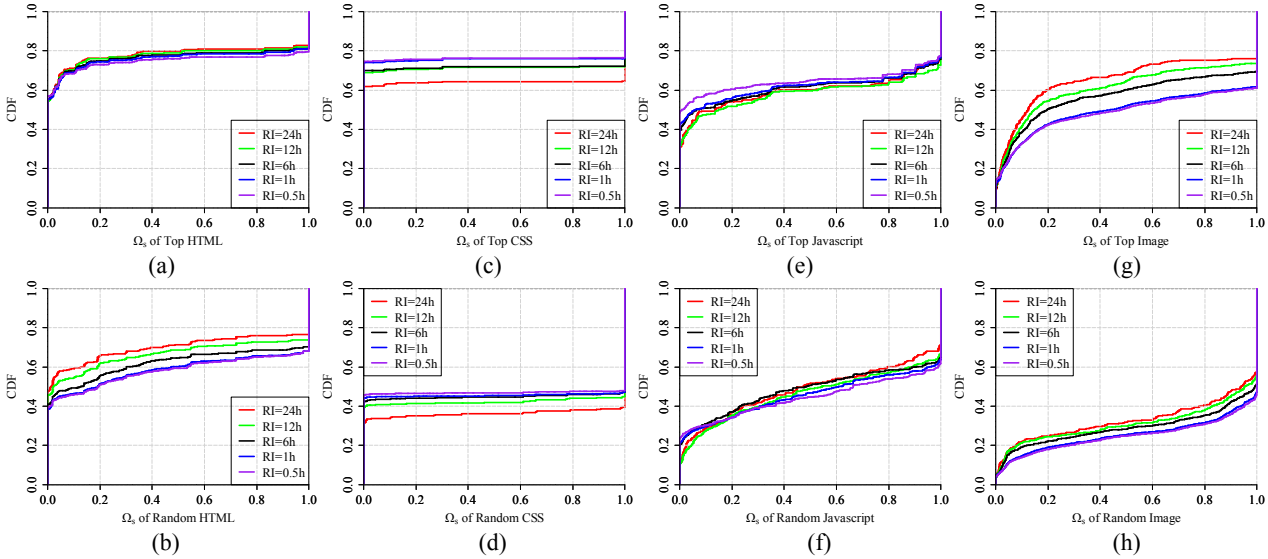


Figure 5. Ω_s of different resource types for Top and Random

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 will be changed. Therefore, popular websites give up the cacheability of HTML.

The resource type with the highest cache performance is Image for Top and JS for Random. 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.

5.2 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 experience of mobile Web browsing.

5.2.1 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 4 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 CDF curves shift to the right when RI decreases in Figure 4. 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 4 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.

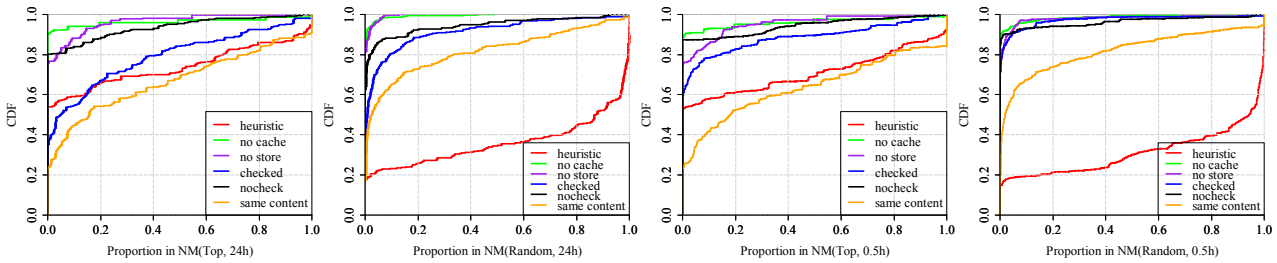
We then examine the Redundant Transfer problem for different resource types. Figure 5 shows Ω_s's CDF with different RIs for both Top and Random, corresponding to the resource types of HTML, CSS, JavaScript, and Image.

Table 4. 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

Table 5. NH of aggregated websites and resource types

	24h		6h		0.5h	
	NH_s	NH_n	NH_s	NH_n	NH_s	NH_n
All_{Top}	6.4%	2.3%	7.8%	3.0%	8.1%	3.0%
All_{Random}	7.7%	2.9%	7.9%	2.7%	9.0%	2.8%
$HTML_{Top}$	35.7%	18.7%	35.8%	18.8%	40.0%	19.9%
$HTML_{Random}$	24.6%	10.3%	23.2%	8.9%	21.5%	9.1%
CSS_{Top}	8.3%	5.6%	13.4%	9.3%	12.1%	7.7%
CSS_{Random}	13.5%	2.8%	9.8%	2.7%	10.5%	2.9%
JS_{Top}	4.8%	2.1%	10.2%	3.7%	10.4%	4.1%
JS_{Random}	10.5%	4.3%	13.4%	3.7%	16.6%	5.3%
$Image_{Top}$	0.1%	0.3%	0.1%	0.3%	0.1%	0.2%
$Image_{Random}$	1.0%	1.3%	0.6%	0.9%	0.5%	0.8%



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

Figure 6. Breakdown of NM

For all kinds of resources, Ω of Random is larger than that of Top, represented by 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.

For HTML (5(a) and 5(b)), 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 (5(c) and 5(d)), 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 will be served by the cache. In contrast, the CSS will be 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 (5(e) and 5(f)), Ω of both Top and Random does not vary so much, indicating there are a fixed set of NM in JS.

For image (5(g) and 5(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.

5.2.2 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 5 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 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 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 border and button) may change when websites get updated.

5.3 Analysis of Root Causes

To study the third research question "*What causes the gap between ideal and actual performance*", we break down the cache behaviors of NM and plot the CDF of each case in Figure 6. As explained in Section 4.2, 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 6 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 similar because the same resources have different URLs at every visit.

Table 6. Resources of Heuristic Expiration

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

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.

We next analyze Same Content, Heuristic Expiration, and Conservative Expiration Time in detail.

5.3.1 Same Content

We first 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.

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. Therefore, every time the webpage with the resource's reference is visited, a new query string will be generated and the browser will treat 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 will be created. Since an update is not likely to change all the resource files, Same Content will happen 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.

5.3.2 Heuristic Expiration

We next 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 6 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.

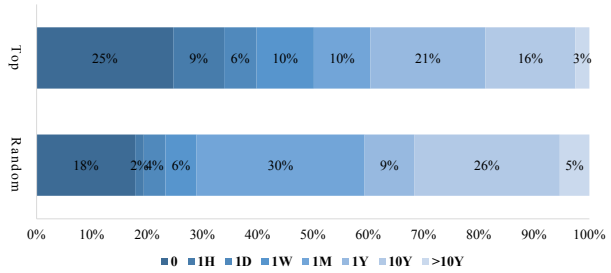


Figure 7. Expiration time distribution of Top and Random

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 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).

5.3.3 Conservative Expiration Time

Figure 7 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.

6. IMPLICATIONS

Table 7 summarizes the findings and implications of our measurement results. Overall, caching is helpful but it is a tradeoff between website functionality and cache performance when cache parameters are configured. To balance the two factors, we suggest using the HTML5 AppCache interface [6]. 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 will always be loaded from the cache until the manifest file is updated.

7. RELATED WORK

With the popularity of mobile computing, research on Web has paced into mobile's era. Studies on mobile Web could roughly be categorized into two branches. One is to measure the performance of mobile Web, trying to understand the latency, data usage, and energy consumption of mobile Web. The other is to improve the performance with novel techniques. As a basic mechanism of Web, measurement and improvement of mobile Web cache performance have driven lots of attention. We next briefly discuss related research on general mobile Web performance and then discuss related research on mobile Web cache specifically.

Table 7. Our major findings and implications of mobile Web cache performance

Findings	Implications
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.	Caching has substantial benefits for mobile Web browsing, especially for ordinary websites. But the cacheability has not been effectively utilized. Popular websites indeed make efforts on caching larger-size resources but do not consider small-size resources very well. Ordinary websites perform worse for both cacheable bytes and objects.
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.	Web developers should configure different caching policies for different types of resources based on the characteristics of websites. Developers from popular websites could consider more for caching JS, while developers from ordinary websites should check the cache configurations for all resources especially the images.
The Redundant Transfer problem is worse when a website is revisited after a shorter period. Some popular websites even have 100% Negative Miss Ratio.	Data traffic and energy of mobile Web browsing are wasted mostly from frequently visited websites. Popular websites should leverage new techniques to further reduce unnecessary network traffic.
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.	Miscached Resource is a common problem for all the websites. Popular websites may have wrong layout and view while ordinary websites could suffer from wrong functionalities. Developers should avoid using the same URL for different images.
Same Content accounts for the main origins 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.	Current techniques to maintain website versions, such as random query strings, are not appropriate for mobile Web browsing. New techniques have to be adopted. 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.
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.	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.
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.	The expiration time is conservatively configured. Developers 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.

Mobile Web Performance. Measurement studies have been conducted to understand mobile Web performance. Mobile HTTP Archive [7] 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. [14] compared smartphone and laptop Web traffic based on a three-week-long wireless communication trace collected in an enterprise environment. Thiagarajan et al. [19] measured the precise energy used by a mobile browser to render Web pages. Many techniques are proposed by researchers and employed by mobile developers to improve mobile Web performance. Jones et al. [12] studied how browser parallelization could help mobile browsers reduce latency and improve energy efficiency. PocketWeb [13] leverages prefetching techniques to load resources based on user behaviors with server support to reduce latency. Some latest commodity browsers such as Amazon Silk Browser [2], Google Chrome Beta [5], and OperaMini [8] are particularly designed for mobile browsing by leveraging server or cloud-based offloading [18]. New protocols, such as SPDY [10][22] and QUIC [4], are designed and deployed to optimize mobile Web performance from lower levels.

Mobile Web Cache. A lot of efforts have been done to study the performance of mobile Web cache. Wang et al. [24] 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. [25] 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. [16] 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 [17], 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 in our methodology: we use a long-period sampling traces and determine the cache efficiency by resource evolution history. Recent work [23] examined how Web browsing can benefit from micro-cache that separately caches layout, code, and data at a fine granularity. It studied how and when these resources are updated and found that layout and code that block subsequent object loads are highly cacheable.

8. CONCLUSION

In this paper, we have presented a comprehensive measurement and analysis of mobile Web cache performance. We have proposed a proactive approach to crawling the resource update traces from two sets of websites periodically with a fine-grained time interval. We have built an analysis model to capture the relationship between cache behavior and resource updates. Based on the model, (1) we have examined the ideal cacheability and found that caching is substantially helpful for mobile Web browsing; (2) we have measured the actual performance and identified a big gap between ideality and reality; (3) we have investigated three main root causes – Same Content, Heuristic Expiration, and Conservative Expiration Time. We have provided recommendations for developers to improve mobile Web cache performance according to the implications of our findings.

9. ACKNOWLEDGMENTS

This work was supported by the National Basic Research Program (973) of China under Grant No. 2014CB347701, the Natural Science Foundation of China (Grant No. 61421091, 61370020, 61222203), and "Star-Track" Young Scholar Program of Microsoft Research Asia. Tao Xie's work was supported in part

by NSF grants CNS-1434582, CNS-1439481, CCF-1349666, CCF-1409423, CCF-1434590, and CCF-1434596.

10. REFERENCES

[1] Alexa. http://www.alexa.com/

[2] Amazon Silk browser. http://aws.amazon.com/cn/documentation/silk/

[3] Charles Proxy. http://www.charlesproxy.com/

[4] Experimenting with QUIC. http://blog.chromium.org/2013/06/experimenting-with-quic.html.

[5] Google Chrome beta browser. https://www.google.com/chrome/browser/beta.html

[6] HTML5. http://www.w3.org/TR/html5/

[7] Mobile HTTP Archive. http://mobile.httparchive.org/

[8] Opera mini browser. http://www.opera.com/mobile/mini/iphone

[9] RFC 2616. http://www.w3.org/Protocols/rfc2616/rfc2616.txt

[10] SPDY Protocol - Draft 3. http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3

[11] J. Erman, A. Gerber, M. Hajiaghayi, D. Pei, S. Sen, and O. Spatscheck. To Cache or not to Cache: The 3G case. *IEEE Internet Computing*, vol. 15, pp. 27-34, 2011.

[12] C. G. Jones, R. Liu, L. Meyerovich, K. Asanovi, R. Bod, and K. Parallelizing the web browser. In *Proc. the 1st USENIX Conference on Hot Topics in Parallelism*, pp. 7-7, 2009.

[13] D. Lymberopoulos, O. Riva, K. Strauss, A. Mittal, and A. Ntoulas. PocketWeb: instant web browsing for mobile devices. In *Proc. the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1-12, 2012.

[14] 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, pp. 423-424, 2012.

[15] F. Qian, J. Huang, J. Erman, Z. M. Mao, S. Sen, and O. Spatscheck. How to reduce smartphone traffic volume by 30%? In *Proc. Passive and Active Measurement Conference*, pp. 42-52, 2013.

[16] 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. the 10th International Conference on Mobile Systems, Applications, and Services*, pp. 127-140, 2012.

[17] F. Qian, S. Sen and O. Spatscheck. Characterizing resource usage for mobile web browsing. In *Proc. the 12th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 218-231, 2014.

[18] A. Sivakumar, V. Gopalakrishnan, S. Lee, and S. Rao. Cloud is not a silver bullet: A case study of cloud-based mobile browsing. In *Proc. the 15th International Workshop on Mobile Computing Systems and Applications*, pp. 1-6, 2014.

[19] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh. Who killed my battery?: analyzing mobile browser energy consumption. In *Proc. the 21st International Conference on World Wide Web*, pp. 41-50, 2012.

[20] X. Wang, X. Liu, Y. Zhang, and G. Huang. Migration and execution of JavaScript applications between mobile devices and cloud. In *Proc. the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*, pp. 83-84, 2012.

[21] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. Demystifying page load performance with WProf. In *Proc. the 10th USENIX Conference on Networked Systems Design and Implementation*, pp. 473-486, 2013.

[22] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. How speedy is SPDY? In *Proc. the 11th USENIX Conference on Networked Systems Design and Implementation*, pp. 387-399, 2014.

[23] X. S. Wang, A. Krishnamurthy and D. Wetherall. How Much Can We Micro-Cache Web Pages? In *Proc. the 2014 Conference on Internet Measurement Conference*, pp. 249-256, 2014.

[24] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie. How far can client-only solutions go for mobile browser speed? In *Proc. the 21st International Conference on World Wide Web*, pp. 31-40, 2012.

[25] Y. Zhang, C. Tan and L. Qun. CacheKeeper: a system-wide web caching service for smartphones. In *Proc. the 2013 ACM International joint Conference on Pervasive and Ubiquitous Computing*, pp. 265-274, 2013.

[26] Y. Zhu and V.J. Reddi. WebCore: Architectural support for mobile Web browsing. In *Proc. the 2014 ACM/IEEE 41st International Symposium on Computer Architecture*, pp. 541-552, 2014.