# xCache: Rethinking Edge Caching for Developing Regions

Ali Raza
New York University Abu Dhabi
Abu Dhabi, UAE
araza@bu.edu

Yasir Zaki
New York University Abu Dhabi
Abu Dhabi, UAE
yasir.zaki@nyu.edu

Thomas Pötsch
New York University Abu Dhabi
Abu Dhabi, UAE
thomas.poetsch@nyu.edu

Jay Chen
New York University Abu Dhabi
Abu Dhabi, UAE
jchen@cs.nyu.edu

Lakshmi Subramanian
New York University
New York, USA
lakshmi@cs.nyu.edu

## ABSTRACT

End-users in emerging markets experience poor web performance due to a combination of three factors: high server response time, limited edge bandwidth and the complexity of web pages. The absence of cloud infrastructure in developing regions and the limited bandwidth experienced by edge nodes constrain the effectiveness of conventional caching solutions for these contexts. This paper describes the design, implementation and deployment of xCache, a cloud-managed Internet caching architecture that aims to proactively profile popular web pages and maintain the liveness of popular content at software defined edge caches to enhance the cache hit rate with minimal bandwidth overhead. xCache uses a Cloud Controller that continuously analyzes active cloud-managed web pages and derives an object-group representation of web pages based on the objects of a page. Using this object-group representation, xCache computes a bandwidth-aware utility measure to derive the most valuable configuration for each edge cache. Our preliminary real-world deployment across university campuses in three developing regions demonstrates its potential compared to conventional caching by improving cache hit rates by about 15%. Our evaluations of xCache have also shown that it can be applied in conjunction with other web optimizations solutions like Shandian, and can improve page load times by more than 50%.

## CCS CONCEPTS

•**Networks** → **Network design principles;** *Public Internet;* Network performance evaluation;

## KEYWORDS

Distributed web caching, Cache management, Web performance

**ACM Reference format:**
Ali Raza, Yasir Zaki, Thomas Pötsch, Jay Chen, and Lakshmi Subramanian. 2017. xCache: Rethinking Edge Caching for Developing Regions. In *Proceedings of ICTD '17, Lahore, Pakistan, November 16–19, 2017,* 11 pages.
DOI: 10.1145/3136560.3136577


*ICTD '17, Lahore, Pakistan*

DOI: 10.1145/3136560.3136577

## 1 INTRODUCTION

Web browsing is exceedingly slow for users in developing regions with Page Load Times (PLTs) that can range from tens of seconds to a few minutes [47]. There are three fundamental factors that cause poor performance.

**1. High Server Response Times:** Most developing regions have poor server infrastructure [13] and lack nearby Content Distribution Networks (CDNs). Object requests, in many cases, are routed to servers around the world and hence cause RTTs of 1−2 seconds [47]. Also, the majority of users in these regions are on high latency cellular networks which further increase PLTs [20].

**2. Limited Edge Bandwidth:** Network congestion in developing regions typically occurs at the network edge in the upstream ISP due to a combination of high contention and limited edge bandwidth [11, 21, 35].

**3. Web Page Complexity:** Modern web pages have evolved dramatically over the past decade with a large number of objects fetched in parallel using several competing TCP flows to different servers around the globe [7]. Apart from the increased number of objects, web pages have become much harder to interpret due to embedded scripts making it difficult for middleboxes to unearth different forms of flow and object level dependencies [27, 41].

The challenge of web performance and coping with increasing complexity has by itself spawned a wide array of systems [31, 42, 48], CDNs [4, 39], web transport acceleration mechanisms [36, 37, 49] and middlebox-based web optimization engines [1, 8, 9, 27, 34]. While these solutions can improve web performance, there are two main differences in developing contexts that make gains insufficient. First, most developing regions lack good cloud infrastructure support [38] due to cost, bandwidth and power constraints. Second, edge nodes in developing regions are behind high latency and low bandwidth networks which limits the amount of network-intensive tasks that can be imposed on edge nodes. In conjunction, these two issues reduce the benefits achievable by existing methods.

This paper describes xCache, a cloud-managed distributed web caching architecture that proactively manages the cacheable web content at the extreme edge of slow networks. xCache is designed around a set of software defined Edge Caches (ECs), managed by Cloud Controllers (CCs) with more resources and aggregate information. The basic design philosophy of xCache is to enable the Cloud Controller to continuously profile and learn the dynamicity of objects in popular web pages and perform page-level cache management of ECs.

Four aspects of the xCache design make it different from conventional solutions that improve web performance:

- *Fine-grained Page Analytics and Object Grouping:* Conventional caching solutions make decisions at an object level. xCache considers caching at the page level. Given a popular web page, xCache profiles the dynamicity of objects in a web page and learns an object-group representation that groups related objects with a similar level of object dynamicity. xCache then makes caching decisions based on these object-groups.
- *Learning Page Dynamicity on the Fly:* xCache uses features of individual objects to learn the dynamicity of objects in a web page and can achieve good predictive accuracy using simple regression techniques.
- *Centralized Edge Cache Management:* Edge Caches have a limited purview of object dynamicity across web pages and may have limited bandwidth to perform advanced analytics on individual pages. For popular web pages, the Cloud Controller can determine the optimal cache configuration of the required set of objects that need to be cached at each Edge Cache.
- *Proactive Cache Management:* xCache's Cloud Controller proactively determines the expected liveness and utility for each object-group. xCache also selectively pushes delta or compressed object updates to the Edge Caches to lower bandwidth overheads.

xCache represents a specific point in the cache design space that is especially relevant for developing contexts that normally experience high end-to-end server latencies, have limited edge bandwidth and face poor performance due to the complexity of web pages. By profiling web pages using a central controller, xCache aims to reduce the burden on ECs and to enhance their utility. xCache untangles the complexity of pages and determines the optimal cache configuration for each individual web page. We have implemented a proof-of-concept prototype and deployed it in university campuses in the UAE, Pakistan and Sri Lanka. Our evaluations have shown that xCache can significantly improve web performance in developing regions, increasing cache hit rates by 15%, and reducing bandwidth usage of certain pages by more than 65%. In addition, xCache can provide additional PLT improvements of more than 50% compared to Shandian [43], a recently proposed proxy-based web performance optimization solution. We also show that xCache can work in conjunction with enhancements like Shandian and that combining the two solutions has the potential to further improve end-to-end web performance.

## 2 THE STATUS OF THE WEB

In this section we use detailed measurements of web page download times from multiple geographic regions to illustrate the key web performance issues in developing regions including: long server response times, increased web complexity (i.e., many web objects across many servers for individual pages), highly variable page dynamicity (across and within pages), and HTTP headers as poor approximations of actual dynamicity. We conducted our measurements from two different geographic regions for comparison: Pakistan and the United States. To get a more holistic view of web page performance, we chose around 80 web pages from different popularity ranges. We selected 20 pages from each of the following Alexa ranges (0-100, 100-200, 500-600, 1000-1100). We downloaded a *version* of a page every 30 minutes for 30 days, consisting of the complete set of objects retrieved from a single iteration for the associated page. The results presented in this section are based on this set of pages.

### 2.1 Object Download Times

Most web pages contain a significant number of objects with different file sizes, object hosting locations, content types, etc. The individual load times of these objects, the order of the object retrieval and the dependencies across objects contribute to the overall PLT of a web page. Also, certain object dependencies may cause a web browser to stall and process specific objects before requesting new objects. In general, the overall time to request a single object can be broken down into five stages: block time, DNS resolution time, connect time, wait time (for server response), and object download time.

Figure 1 shows Cumulative Distribution Functions (CDFs) of the web objects' download time breakdown. We used Chrome's developer tools to collect the results in the form of HTTP Archive (HAR) files. Although there are subtle differences between the locations, they share a number of commonalities. First, the CDFs of the DNS requests and the connect times (cyan and green curves) show that about 70% of the objects do not have any DNS or connect times (indicated by the straight horizontal lines). This is due to the fact that the browser does not need to repeat DNS requests for objects with the same host and for objects that were recently requested. As for the connect time, modern web browsers use persistent connections (the TCP connection is kept alive) to download several objects from the same server, i.e., HTTP/2. For the rest 30% of the objects we observed some variations in the DNS resolution and TCP connection setup times across locations, with the US exhibiting significantly lower RTTs than Pakistan.

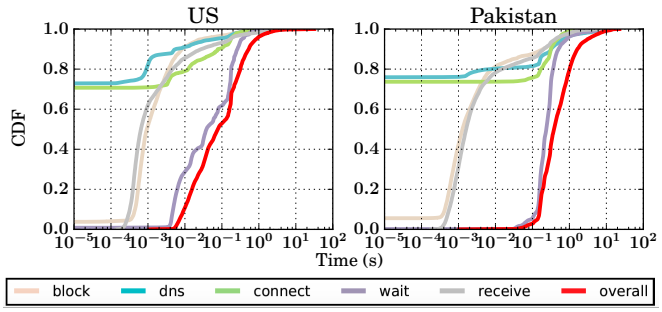


**Figure 1: Breakdown of download times of web objects**

Figure 1 also shows that in both locations the wait time is high and contributes the most to the overall object load time (compared to the red curve). The wait time corresponds to the server response time. The US has a server response time of less than 100 ms in 60% of the cases. In contrast, Pakistan has a server response time of more than 100 ms in all cases. The difference is caused by lack of nearby CDNs and because web servers are configured to respond

differently based on the requester geo-location. Contrary to the common belief that high PLTs are caused by bandwidth limitations and high RTTs, the figure shows that the actual receive time of objects is not the main factor in the overall PLT (both locations show similar object receive times).

We also observe that the portion of block time has significantly decreased over the past few years (as compared to [47]). This improvement is mainly through the introduction of new protocols such as SPDY [26] (and later HTTP/2 [6]), which request multiple web objects simultaneously using a single TCP connection. Modern web browsers have also increased the number of parallel TCP connections that can be opened per page.

## 2.2 Page Load Times

The individual object load times are an integral part of the overall PLT. Despite the fact that objects are loaded in parallel and have dependencies on each other, long object load times cause PLTs to be high. In general, the more objects a web page consists of, the longer it takes to load the entire page. Figure 2 shows a CDF of the number of objects per page for all requested sites. We observed that around 20% of the requested pages in our sample have more than 200 objects.

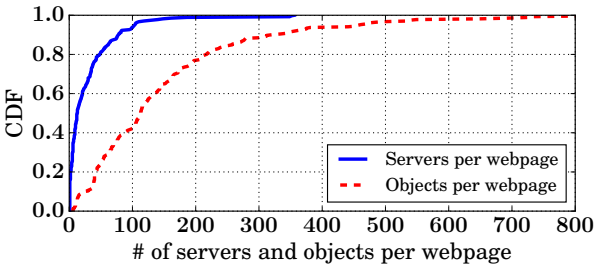


**Figure 2: Number of servers and objects per web page**

We also observed that the majority of web pages' content are hosted on more than a single server. Regardless of the web objects' geographical location, web browser have to initiate multiple DNS requests for every object hosted on a different server than the origin (where the index.html is hosted). Figure 2 shows that around 60% of the pages have their content spread across more than 20 servers.

## 2.3 Web Page Dynamicity

Objects in a web page can exhibit varying levels of dynamicity. We show four example pages from our dataset here to highlight the dynamicity of web pages. Based on their requested URLs and the MD5 hash values of the objects, the collected objects are classified into three different categories:

(1) *Constant objects:* objects that stay the same across versions, i.e., objects requested from the same URL and with the same hash values.
(2) *Changed objects:* objects requested from the same URL, but with different hash values, i.e., their content is updated.
(3) *New objects:* objects that have not been seen before, i.e., newly requested URLs.

We look at the dynamicity of web pages from two different perspectives. First, we compare all objects in every version of a page to the first version that was recorded. Second, we compare all objects in every version to all previous versions of the same web page. Figure 3 shows stacked bar graphs for the four example pages. The upper subfigure shows the comparison between version $n$ and version 1 and the lower subfigure shows the comparison between version $n$ and all previous versions $[1, n-1]$, where $n$ ranges from version 2 to version $N = 1440$ (48 samples per day $*$ 30 days).

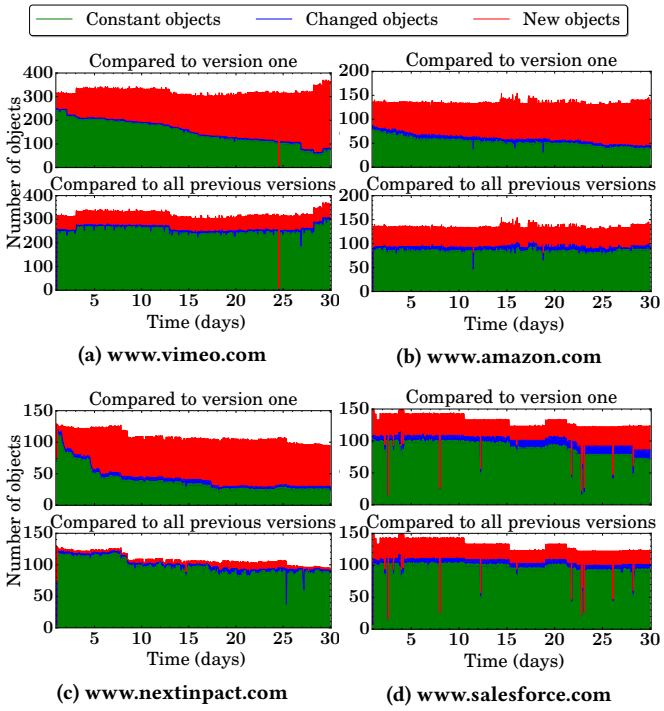


**Figure 3: Web page dynamicity**

We observe that the percentage of constant objects in a web page does not drop to zero, but rather stabilizes around a certain value. After further analysis we found that these objects are part of the page template (e.g., icons, logos, etc.) that do not change frequently. From the lower figure in each sub-figure, we can observe that at least 65% of the content in these web pages does not change and can therefore be cached over long periods of time. Upon examining these changes in more detail, we find that text-based objects (HTML, CSS, JS) often only change in minor ways (sometimes just a few lines), as observed by Wang et al. [42].

## 2.4 Effect of HTTP Headers

Although web caches are already widely deployed, legacy caches suffer from several problems. Existing proxies/caches rely on HTTP cache control headers as an input for their caching policies [17]. These control headers (*cache-control*, *expired at*, *max-age*, *etc.*) are used by caches to decide what objects to cache and for how long. Caches use these headers to enforce revalidation, freshness check and other directives. The *max-age* directive, for example, specifies the time after which the object should be considered stale.

These cache headers are collectively used to determine whether an object is still fresh. If these headers are set inaccurately by the content provider, then either network resources are wasted (due to unnecessary refreshing of content) or stale content is served.

In Figure 4 we compare the actual change rate of an object to the *max-age* set by the content provider. For this analysis we removed objects that had no *max-age* and objects with a *max-age* larger than one month. Also, the lower portion of the figure (in gray) is below our sampling rate of 30 minutes. Ideally, we should observe that most of the collected data points lie on the 45 degree line, where the *max-age* matches the change rate of the objects. Instead, we observe that the *max-age* is highly uncorrelated with an object's life span. Web publishers (or content providers) tend to substantially under (upper left) or over-estimate (lower right) the life span of objects.

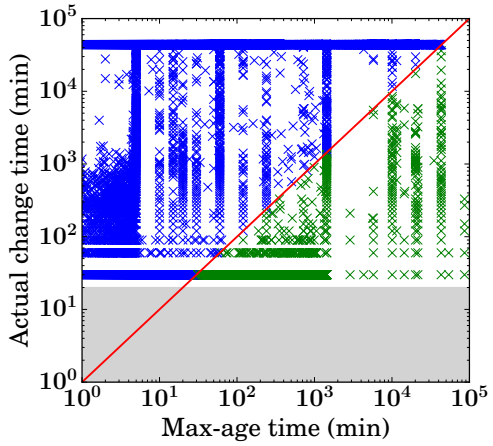


**Figure 4: Actual web object change time vs. *max-age* header**

## 3 XCACHE ARCHITECTURE

The xCache architecture is illustrated in Figure 5 and consists of a set of Edge Caches (ECs) in close proximity to the users. ECs are centrally monitored and managed by a Cloud Controller (CC). The CC collects web page requests and performance information from the ECs to determine the best set of web pages to prefetch and store at each EC.

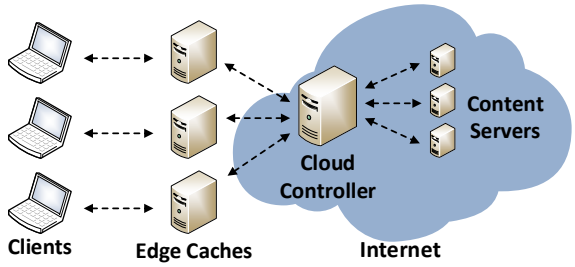


**Figure 5: xCache architecture**

An EC is an enhanced HTTP network level cache that is placed at the edge of the network. ECs could be deployed by ISPs, telcos, or other large institutions that have access at these edge locations (e.g. Facebook, Google, university campuses, etc.). To web clients, an EC functions as an ordinary proxy or cache that receives users' HTTP requests and in the case of a cache hit, serves the objects directly. In the case of a cache miss, the request is relayed to the Internet. ECs differ from conventional caches in two ways: First, ECs are managed by the CC and are periodically updated with web objects. Second, ECs gather access logs and periodically send the aggregated statistics to the CC. The main goal of the CC is to optimize the content of each EC by maximizing the EC's hit rates while minimizing the bandwidth used to update them. The CC has two key components that help to achieve this, namely the Page Profiler and Cache Manager.

### 3.1 Page Profiler

The task of the Page Profiler is to determine the list of active pages to profile and to learn the object-group representation of each active page. The Page Profiler decides on the pages to profile using the access logs received from the ECs. The CC tracks the list of active pages across ECs and continuously evaluates each page to determine the object-group representation by learning the dynamicity of the individual objects and the relationship between dynamically changing objects across time-varying versions of a page. The object-group at a given time represents a grouping of the current set of active objects in a page along with their expected rate of change across future versions of the same page.

*3.1.1 Object-groups.* Web pages are inherently complex to analyze, especially at the object, flow-level, and script-level dependencies. Recent work [27, 41, 43] has attempted to understand these dependencies at these fine-grained granularities with the goal of enhancing end-to-end web performance using a combination of prefetching and pre-rendering content in the cloud. In comparison to these approaches, xCache takes a much simpler approach to derive object-group representations. The dynamicity of an object is determined by its rate of change across different versions of a page. For example, two objects, $X$ and $Y$, are closely related to each other if they share the same dynamicity pattern across different versions of a page. In essence, whenever $X$ changes in a version, $Y$ also correspondingly changes and vice versa. An object is defined to be highly dynamic if it varies across each version of the page and is deemed not cacheable by the CC. A new object that is observed for the first time in a given version of a page has an unknown dynamicity. For simplicity, we define all new objects to be in the highly dynamic object-group.

For a new page, the Page Profiler performs a bootstrapping process to gather initial statistics of the page. When a new page is profiled, the bootstrapping process requests the page several times over a certain time interval (e.g., every 10 minutes, 15 times). This information determines objects that belong to a page, the web objects' estimated change rate, and the variations across page requests. Based on this information, the CC decides which part of the page is constant, cacheable, and can be sent to the EC. After the bootstrapping phase, the CC determines how many objects of a page are cacheable and also decides on the pages are the most useful to prefetch. The CC is at a well-resourced location with high bandwidth and low delay connectivity. The ECs, on the other hand,

are located in developing regions. Thus, the connection between the CC and the ECs might have limited bandwidth. As a result, the CC has to decide carefully which pages to prioritize and send to the ECs.

*3.1.2 Learning Object-groups.* In order to prevent the CC from continuously downloading full pages over short time periods to determine the dynamicity of web objects, xCache automatically learns the characteristics of object-groups with low training effort. Based on observing the dynamicity of several pages, web content can be broadly divided into three categories: objects that do not change over multiple days (constant objects), objects that change within hours (changing objects), and new objects.

In addition, we have observed that web objects in these categories have distinct features. These features can be used as input to standard machine learning algorithms to dynamically predict the cacheability of web objects (i.e. their category).

**Feature Extraction per Object:** Each web object's dynamicity can be linked to various specific features that are used with a Standard Support Vector Machine (SVM) model [32]. These features are fed to the SVM in the form of individual feature vectors and contain the following information:

*1. Special Characters in URLs:* From our collected dataset we observed that objects with URLs that do not contain special characters have a higher probability to belong to the group of constant objects. On the other hand, objects with URLs that contain special characters are of dynamic nature, i.e., they tend to change frequently (sometimes even per request). This trend comes from the fact that most of these web objects are requested for dynamic purposes like jQuery requests, which are user specific and mostly used for web analytics. Responses to these requests vary depending on several factors, such as user-agent, access time, geo-location, etc. This makes these resources difficult to cache.

*2. Content Type:* The content type is also directly related to the web object dynamicity. We have observed that certain web object types such as images fall more frequently within the constant objects group. This is because whenever an image is changed, it appears under a different URL. In contrast, text-based web objects such as a HTML, CSS, or JS files are more likely to partially change.

*3. Max-age header:* The *max-age* is a measure of dynamicity of a web object provided by the content provider. Although this is not often an accurate measure, it is still helpful for the predictive model.

*4. Time-since-last-changed header:* The time since an object has last changed is mainly introduced to track the web objects that have cyclic changes or changes at some constant intervals.

Using these features, xCache trains an SVM with longitudinal data of a collection of versions of a given page and uses the trained SVM to make caching decisions.

## 3.2 Cache Manager

The CC Cache Manager uses three methods to maintain the content of ECs: page-level cache signaling, prefetching of dynamic objects, and distribution of objects.

**Page-level Object Caching Signals:** The base design of xCache involves a simple signaling protocol where the CC uses the object-group information for a given page determined by the Page Profiler to inform the caching decisions of each EC. Given that an EC has observed a page request, the EC retains all objects of a page in its cache for a limited period until the CC provides an updated object-group information for this page to the corresponding EC. xCache can also provide meaningful caching signals based on quickly training an SVM across a few versions of the page over a short time-period and then proactively updating the EC's predictions. This process can be applied to pages that are requested even for the first time.

**Prefetcher:** The CC determines the changes in the dependency tree in terms of which part is cacheable/non-cacheable over time, while updating the ECs using as little bandwidth as possible. The Prefetcher's role is to prioritize pages based on the following information: page popularity, page rate of change, and page transfer cost. Ideally, the Prefetcher would download the pages at a high frequency to perfectly track all the changes, or simply be notified by the web publisher in a push-based model. Instead, xCache assumes a pull-based model operating over fixed time epochs $T$. At the beginning of every epoch, all known pages are ranked based on their utility values (see Section 3.3). The Prefetcher downloads these pages in a ranked order of their utility values and updates their objects. After time $T$ elapses, the Prefetcher recomputes the utilities and ranks the pages again. However, the Prefetcher is best-effort and pages of low rank will not be prefetched before $T$ expires. Also, pages that require certificates or use end-to-end encryption are non-cachable and hence cannot be prefetched.

The Prefetcher cannot track all changes of each page since objects may change at a rate faster than the polling frequency. To determine whether this is the case, the Page Profiler's bootstrapping phase provides the CC with an initial estimate of the rate of change of an object. Using this estimated change rate, the CC decides whether the object should be tracked. We use the time epoch, $T$, as a threshold of the minimum rate of change that will be tracked (considered for prefetching) by the CC.

**Distributor:** Every object-group of a page is associated with an utility value that is computed by the Prefetcher as we elaborate in the next section. For each page, the Distributor determines the object-groups that should be transferred to each EC based on their utility values. The Distributor further uses two simple optimizations for reducing the bandwidth overhead for distribution. First, the Distributor determines the best form for transmitting each object to each EC. Certain objects (especially CSS, HTML, etc.) can be updated using deltas. To perform delta updates, the CC and EC maintain version numbers and delta updates encode the changes across versions. In our evaluation, we show that delta updates can reduce the bandwidth requirement for specific types of objects. Other objects that cannot be delta updated are compressed and transmitted to the EC.

## 3.3 xCache Utility Functions

The Prefetcher determines the rank of each page based on two utility functions: time utility $U_T$ and bandwidth utility $U_{BW}$. Both utility functions are defined as the ratio of the time (or bandwidth) of conventional caching over xCache prefetching, i.e., the savings in time or bandwidth that xCache provides over conventional caching solutions. When computing the rank of a page, the CC can determine the desirable trade-off between bandwidth and time savings

by setting different weight-factors to these utilities. In our current design, we gave equal weights to both utilities.

Given a time-period $T$, we divide the object-groups of a page $x$ into $X_1$ and $X_2$, where $X_1$ is the set of tracked objects with a change time $\geqslant$ T, and $X_2$ is the set of untracked objects with a change time $< T$. Let $N_{req}$ be the number of requests of a page within the time period $T$ reported by the EC. Let $|x|$ denote the size of the objects in bits. Other parameters in our utility calculation include: $RTT$ is the round trip time per object from the CC to the web server, $BW$ is the bandwidth between the EC and CC, $q_i$ is an indicator variable of object $i$'s cache header expiration (e.g., if *max-age* is exceeded, $q_i = 1$), and $p_i$ is the probability that object $i$ has changed since the object was last updated. Given these parameters, the time utility $U_T$ is defined as follows:

$$U_T = \frac{\alpha}{\beta} \tag{1}$$

where $\alpha$ is the time cost of a page without xCache

$$\alpha = N_{req} \cdot \sum_{x \in X} q_i \left(RTT_i + p_i \cdot \frac{|x|}{BW}\right) \tag{2}$$

and $\beta$ is the time cost with xCache

$$\beta = \sum_{x \in X_1} p_j \cdot \frac{\delta|x|}{BW} + N_{req} \cdot \sum_{x \in X_2} q_i \left(RTT_i + p_i \cdot \frac{|x|}{BW}\right) \tag{3}$$

The bandwidth utility $U_{BW}$ is defined similarly as the ratio of the bandwidth used to download a page with a legacy cache over the bandwidth used to download a page with xCache. The bandwidth utility $U_{BW}$ is defined as follows:

$$U_{BW} = \frac{\gamma}{\sigma} \tag{4}$$

where $\gamma$ is the bandwidth cost of a page without xCache

$$\gamma = N_{req} \cdot \sum_{x \in X} q_i \cdot p_i \cdot |x| \tag{5}$$

and $\sigma$ is the bandwidth cost of a page with xCache

$$\sigma = \sum_{x \in X_1} p_j \cdot \delta|x| + N_{req} \cdot \sum_{x \in X_2} q_i \cdot p_i \cdot |x| \tag{6}$$

The utility calculations are bound by a single parameter, time period $T$, which dictates the dynamicity of pages that the CC is willing to operate for a given page. Our current design chooses an operational value of $T = 30$ min across popular pages. Based on both utilities, the CC prioritizes the pages that are prefetched and distributed to the ECs.

## 4 IMPLEMENTATION

**Cloud Controller:** The CC is implemented in Python in conjunction with the Selenium web-driver [14] and a web proxy. The Page Profiler goes through each web page within the access logs. If the page profile already exists, the Page Profiler updates the popularity of the web page within the page profile; otherwise, it creates a new profile using the bootstrapping phase to build the initial object-group page statistics. These statistics are later used by the Page Profiler to calculate the page utility and are stored on a per object basis and include object content, content-hash, cache-header, and the object change history.

In the bootstrapping phase, each page is regularly requested for 15 times every 10 minutes. The Prefetcher requests a page from the origin (content server) and returns all of its associated objects. These objects are then indexed and further processed. In order to store and index all retrieved objects, the Prefetcher requests go through a simple web proxy. The web proxy functionality is extended to store a copy of the objects within a local database before returning them to the Selenium web-driver.

Object changes are detected using a simple MD5 hash value and the value is stored in the page profile of each page. For every object change, the Distributor decides whether to update the corresponding ECs based on the page profiling, utility prioritization, and the bandwidth limitation, or not. The Distributor maintains long-lived TCP connections directly with the ECs, thus saving costly connection setup delays.

**Edge Cache:** The EC is implemented as a modified version of the Apache Traffic Server (ATS) [18] with an extended set of functionalities (corresponding to the EC module). This module implements features to report access patterns to the CC and to receive objects from the CC. The access patterns of the EC are obtained from the ATS logs every 30 minutes. An ordered popularity list is also shared with the CC, which contains all requested web pages as well as their popularities (i.e., number of times each page was requested). The ATS was configured to allow the EC module to add received object updates from the CCs Distributor to the ATS cache.

## 5 EVALUATION

In this section, we evaluate xCache through a series of experiments to verify its performance gains. Three evaluations were performed:

(1) A real-world deployment environment where we evaluated xCache in terms of cache hit rate, object serve time, bandwidth savings, and PLT.
(2) An evaluation on the impact of the object-group prediction algorithm for object-level caching signals and stale content.
(3) A sketch of how xCache can be applied in conjunction with Shandian.

Across all our evaluations, we use the term *legacy cache* to refer to an optimized localized HTTP cache that relies on *cache-control* and *expires* headers for its eviction policy for objects.

### 5.1 Real-world Deployment

We have tested the xCache performance under diverse network conditions by deploying xCache on university campuses in the UAE, Pakistan and Sri Lanka. In each deployment, we compare xCache with a legacy cache to highlight the performance gains of xCache. In these real-world experiments, xCache does not include the SVM model, which we evaluate separately in Section 5.2.

In order to evaluate this experiment in a repeatable fashion, we considered a simple setup where we used end-user client machines to emulate user requests. In this setup, we configured two machines

to act as clients to generate simultaneous requests with one client configured to use the xCache cache and another client configured to use a legacy cache as their local proxy. The bandwidth and network conditions between the clients and their respective caches were set to be identical. Both caches were connected to the Internet to fetch objects that are not cached. Since both machines were operating simultaneously and to avoid any interference from other network level caches along the path, the *cache-control: no-cache, max-age=0* headers are appended to each outgoing HTTP request on both caches. Consequently, intermediate caches ignore the requests and do not return any cached objects. The EC of xCache was connected to a CC running in the UAE which had the fastest Internet connectivity among the three locations. The legacy cache used a default ATS implementation, whereas the EC used the modified ATS implementation, as described in Section 4.

In order to automate the user requests, we used the Selenium web-driver to emulate user-like behavior. In this configuration, both clients request web pages from a predefined list that contains the requested URLs and a relative timestamps when the each URL was accessed. The list was derived from anonymized access logs that we collected over 30 hours from a university campus with more than 3,000 students in Pakistan. The access logs represent a variety of web pages with a wide range of interests (sports, news, entertainment, etc.).

**Cache Hit Rate:** Cache hit rate is a basic metric that is used to evaluate the performance of caches. A higher cache hit rate generally translates directly into faster PLTs and reduced bandwidth consumption. In general, a good cache will not only cache objects but it should also provide valid (not stale) objects and use bandwidth and storage efficiently. Table 1 summarizes the caching performance of xCache and the legacy cache. A *cache hit* indicates that a cache finds the requested object and a *cache miss* means that the requested object is either not present in the cache or has already expired. Consequently, the cache will first go to the origin (content server) or a cache along the way to retrieve this object. *Other* includes all other requests that were received by the cache and that were not HTTP GET requests, e.g. HTTP POST requests, DNS requests, etc.

| | Region | Legacy cache | xCache |
|---|---|---|---|
| Cache hit | UAE | 35.5% | 56.9% |
| | Sri Lanka | 38.4% | 54.3% |
| | Pakistan | 35.1% | 52.9% |
| Cache miss | UAE | 55.5% | 33.7% |
| | Sri Lanka | 52.4% | 34.9% |
| | Pakistan | 49.4% | 31.9% |
| Other | UAE | 9.0% | 9.4% |
| | Sri Lanka | 9.3% | 10.8% |
| | Pakistan | 15.5% | 15.2% |

**Table 1: Cache hit rate comparison**

Table 1 shows that xCache consistently achieves a higher hit rate by about 15% across all regions. The hit rate is better than the legacy cache as we would expect because xCache proactively updates the cache with the latest objects.

Figure 6 depicts a breakdown of the previous experiment by their six most frequent object types. We observe that xCache provides the most improvement for text-based objects and GIFs.

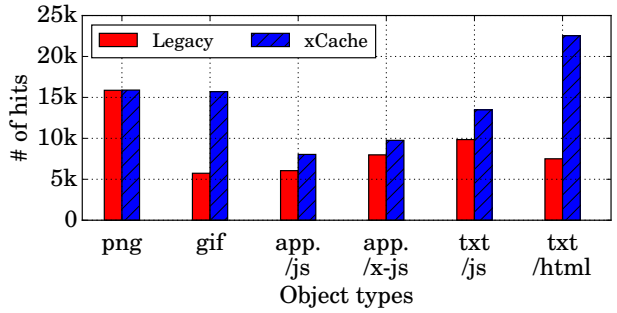


**Figure 6: Cache hits breakdown across data types**

**Bandwidth Savings:** Since one of the objectives of xCache is to save bandwidth, the CC updates the ECs using delta updates. HTML, JS, etc., can be easily delta updated because these objects often change only by a few lines or characters. However, objects like images and videos cannot be delta updated because they mostly change completely.

Although HTTP 1.1 defines the option of delta updating, it is often not implemented by web servers. Figure 7 shows the CDF comparison between the objects' diff size (represented by blue) compared to the original object size (represented by red). The comparison shows that with delta updates, about 90% of objects become smaller than 5 kBytes (up from 70%) and thus bandwidth is saved.

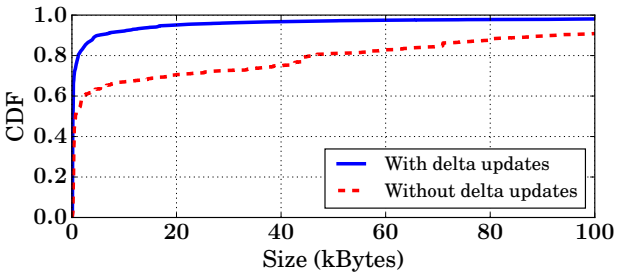


**Figure 7: Web objects diff size comparison**

**Object Serve Time:** When a client requests a page, the client browser sends several HTTP requests for the individual objects. Some of these requests are sent in parallel and others must sent sequentially only after the browser receives certain objects. The PLT depends on the individual object serve times that constitute a page so improving this time will improve the overall PLT. Figure 8 shows the object serve times of the legacy cache and xCache for our three locations. The object serve time in the figure represents the time from the incoming request at the cache to the time the last byte is sent to the client. We observe that with xCache, more

than 50% of the objects are served almost instantly. In contrast, the legacy caches have significantly higher object serve times due to cache misses. Figure 9 shows a breakdown by object type for the UAE results.

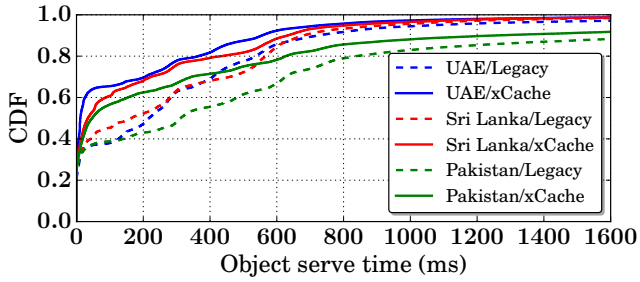


**Figure 8: Objects serve time comparison**

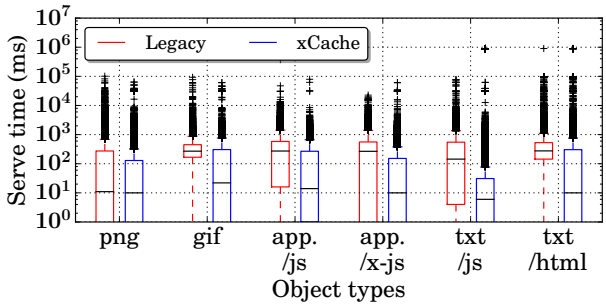


**Figure 9: Object's serve time (UAE) – breakdown across types**

**Page Load Times:** The PLT of a page depends on the individual serve times of the collective assets of the page. Since xCache provides a higher cache hit rate and minimizes the HTTP object's serve time, the performance gain of xCache is reflected in the overall PLT.

Figure 10 shows the PLTs for our experiments. We find that xCache improves the PLTs substantially in the UAE and Pakistan. However, the PLTs are only slightly improved in Sri Lanka because the PLTs and DNS resolution times in Sri Lanka are considerably lower than from the UAE and Pakistan.

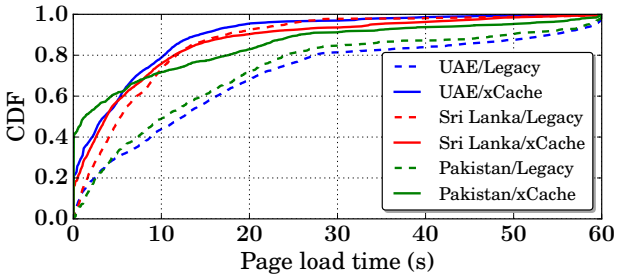


**Figure 10: Page load time across all regions**

## 5.2 Object-level Caching Signals

In this analysis, we evaluate the performance of the SVM prediction model. We trained the SVM model with the first 20 versions of our web pages dataset. The dataset consists of web pages from different categories, i.e., news, entertainment, shopping etc. and were manually selected from Alexa top web pages. Further, each web page $W$ is a collection of versions $v_1, v_2, ..., v_n$, where consecutive versions are 30 minutes apart and every version consists of objects $v_i = o_1, o_2, ..., o_n$. To train the SVM model, the features (as listed in Section 3.1.2) for each $o_i$ were extracted from the first 20 versions of each page. This evaluation was configured to pick a random web page from our dataset and use its access pattern to emulate user requests on the EC. We then run the SVM trained model on all objects within a specific web page version and calculate the hit rate as a percentage of the total number of objects that belong to that page. In addition, we also compute the percentage of stale content that is served by the EC, i.e., when a wrong prediction has been made. This is done by comparing the EC served objects to the absolute truth of the web page. We observe that a SVM model trained on 20 versions can predict the object-group it belongs to with an accuracy of more than 85% and also maintains a staleness of less than 5% across most pages. We report results for evaluating our SVM model against 1000 versions of the web page that follow the first 20 versions.

**Dynamicity vs. Accuracy:** xCache predicts the dynamicity of web objects and based on these predictions, an EC will decide whether to serve the objects from the cache or fetch fresh copies. Figure 11 shows the percentage of stale content served by xCache as a consequence of using the SVM prediction. Stale content is served when a resource is cached and obsolete but the SVM predicts it to be cached and unchanged. We observe that stale content accounts for less than 5% of the served objects even for highly dynamic pages.

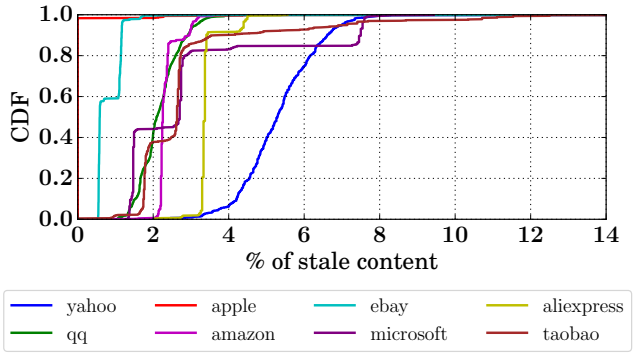


**Figure 11: Stale content using SVM prediction**

## 5.3 xCache with Shandian

Shandian [43] is a web optimization technique that pre-evaluates web pages to enhance PLT performance. Shandian can greatly benefit from xCache, because the main goal of xCache is to bring the web content closer to the client to reduce latencies. Shandian provides clients a partially pre-built DOM structure by preprocessing

JS and CSS objects in advance. Thus, if xCache can keep the JS and CSS objects updated at the EC, Shandian would be able to preprocess them without suffering from performance hits (fetching new copies) when they change. Figure 12 shows the prediction error of our SVM for JS and CSS objects across 30 versions of 30 popular Alexa pages. We observe that xCache's SVM engine predicts most of the JS and CSS files in a page correctly.

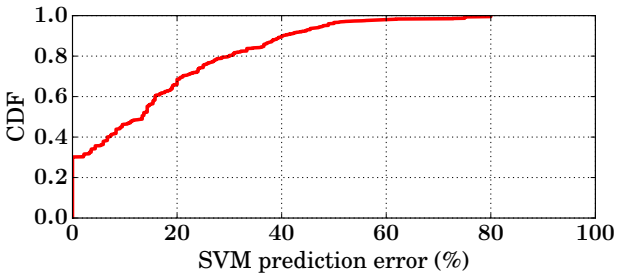


**Figure 12: SVM prediction error of JS and CSS files for 30 versions of 30 popular Alexa pages**

We contacted the Shandian authors and the source code was unavailable, so we emulated the behavior of Shandian under the following assumptions: (a) we analyzed the waterfall model of the HAR file download patterns to infer potential dependencies of a web page (these inferences may not be perfect); (b) we have a sufficient number of TCP connections for parallel downloads while maintaining the per server upper limit imposed by the browser. We call this emulated system Shandian*, which we consider here as an upper bound of Shandian performance.

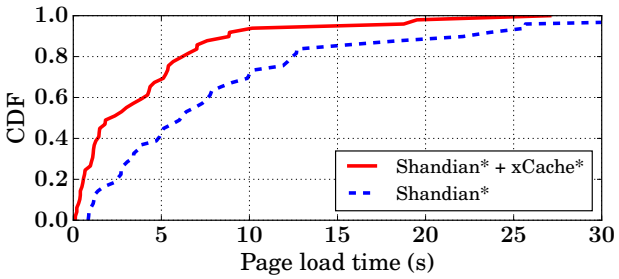


**Figure 13: xCache enhancements to Shandian***

Using the HAR files of the 80 pages described in Section 2, we consider how Shandian* performance can be improved by combining it with xCache*.[1] From our results in Figure 13, we observe that the median page load time is reduced by 50% with the addition of xCache* to Shandian*. Although we have made a number of optimistic assumptions in our emulation for both Shandian* and xCache*, our results suggest that the two techniques could complement each other to yield performance benefits.

[1]For these emulations we assumed that all pages are in the active list of the xCache CC and all cacheable objects within the page are indeed cached by the EC (hence xCache*).

## 6 RELATED WORK

A number of recent works have explored the network bottlenecks in developing regions that cause poor web performance [2, 13, 23, 33, 47]. Chetty et al. [13] and our own previous work [47] showed how high latencies to destination servers can introduce performance bottlenecks in South Africa and Ghana. Koradia et al. [23] found that high latency, bufferbloats, and packet losses are the central causes of poor performance in India. Sen et al. [33] showed that people using Free Basic Service in developing regions experience low QoS, while Ahmad et al. [2] performed a detailed analysis of mobile devices being used to access the Internet in Pakistan.

A few solutions have also been proposed to tackle these various challenges. Wang et al. [45] studied the impact of a client-only solution in improving the user experience and show that excessive revalidation can affect the cache performance [44]. ASAP [49] tries to lower the RTT by shortcutting DNS requests and eliminating TCP's three-way handshake. RuralCafe [12] attempted to improve web search and browsing using a pair of coordinating caches. Each of these projects tackle a specific aspect of the poor web performance in developing regions, but unlike xCache, they do not provide a general end-to-end system that can enhance the performance of PLTs as a whole.

xCache is naturally related to a variety of mainstream research on web analytics, caching, prefetching, and CDNs. While xCache may share design choices with specific works, xCache is specifically designed for developing region conditions where end-hosts experience high server latencies, have limited edge bandwidth and the CC enhances end-to-end performance by pushing the right granularity of content to the EC in a bandwidth efficient manner. Also, in contrast to other recent work on web optimization, xCache operates on a page level through the use of aggregated object-groups.

**Web Page Analytics, Prioritization and Middlebox Optimizations:** Wang et al. [41] showed that computation takes up significant time while browsing and approaches like SPDY [26], caching and mod pagespeed have limited effect on reducing PLTs. Later Wang et al. [42] demonstrated the potential of micro-caching content at a fine granularity. Flywheel [1] compresses data between the user and server to save bandwidth by using an HTTP proxy server. FlexiWeb [34] is a hybrid framework of both approaches (normal browsing and middleboxes), which dynamically decides whether to use middleboxes or normal browsing, depending on the network conditions and data types. Although these systems target modern web performance issues, they do not address the main performance bottlenecks in developing regions.

Shandian [43] also attempts to reduce the PLT by downloading and evaluating a web page at the middlebox. There are number of works that focus on prioritizing the critical path resources in order to decrease the user perceived PLT [8, 9]. Polaris [27] uses a dynamic client-side scheduler that runs on unmodified browsers; using a fully automatic compiler, Polaris enables servers to translate normal pages into ones that load themselves. Unlike the reactive approach of Shandian and Polaris, xCache can proactively prefetch and analyze web pages before they are requested by users.

**Caching in Developing Regions:** RuralCafe [12] focuses on providing offline Internet for clients in regions with intermittent network conditions. Unlike xCache, RuralCafe does not target to

have the most up-to-date version of web pages, but rather focuses on providing the availability of web pages, thus sacrificing liveness. Interactive Caching [10] is a similar approach where content is grouped together under a certain topic. HashCache [5] introduced techniques for scaling up caching for cheap commodity laptops with limited memory. Collaborative caching techniques have also been tested in developing contexts [31] with limited success. Smart Caching [48] is an in-browser cache which caches stable style data and layout data for DOM elements, but this approach only benefits individual users.

**Prefetching:** Many studies have shown how cloud-based prefetching and client prefetching solutions can produce performance benefits [15, 24, 25, 28]. In the Mowgli system [25], a user marks the pages of interest while browsing and the system starts prefetching them in the background. Some previous works [16, 37, 46] used historical access patterns and popularity metrics to decide which content should be prefetched, including optimizations such as DNS prefetching and caching the TCP connection based on the previous access patterns. HTML5 [40] allows developers to specify prefetching of certain important resources. Stark et al. [36] proposed prefetching and validation of SSL certificates in order to minimize the TLS handshake time. xCache also uses prefetching, however its prefetching strategy differs from these works in that it optimizes for the prioritization of content that should be sent to the ECs. Additionally, xCache is not user-based, but rather for multiple ECs that serve groups of users.

**Content Distribution Networks (CDNs):** CoBlitz [29] efficiently distributes large files using a CDN designed for HTTP without requiring any modifications to standard web servers and clients. Lsync [22] is a latency-sensitive file transfer system used for synchronizing WANs. CoDNS [30] is a lightweight cooperative DNS lookup service that can be used to augment existing nameservers. CoralCDN [19] is a decentralized and self-organizing peer-to-peer web-content distribution network that redistributes and replicates data that people find useful. Aperjis et al. [3] propose a price-assisted content exchange system that leverages a mechanism for exchanging currency for desired content with a single and decentralized price per peer. Ariyasinghe et al. [4] combined prefetching and caching in CDNs to reduce user perceived latency. xCache aims to provide a flexible and deployable architecture that brings the functionality of caches closer to that of CDNs.

## 7 CONCLUSIONS

This paper describes the design, implementation and deployment of xCache, a cloud-managed Internet caching architecture for developing regions. xCache addresses two specific challenges that make the Internet caching problem different in developing regions: (a) the absence of cloud infrastructure in close proximity to end-hosts thereby causing high end-to-end latency; (b) the presence of limited per user edge bandwidth and high network contention at the edge. xCache addresses these challenges by aggregating access patterns in the cloud across Edge Caches and proposing a cloud-managed caching model where a Cloud Controller maintains liveness of popular objects of web pages. A key building block of the xCache design is to estimate the utility of xCache over legacy caching solutions and to apply the xCache cloud-managed caching model only to objects which have high utility. Using a detailed evaluation across three different developing regions, we show that xCache can significantly improve web performance over legacy caching.

## ACKNOWLEDGMENTS

We thank the NYU Abu Dhabi Research Institute, the Center for Technology and Economic Development (CTED) in NYU Abu Dhabi, and the Cisco Research Grant for supporting Lakshminarayanan Subramanian on this project. We thank Muhammad Saqib Ilyas for his help in Pakistan and Thushara Weerawardane and the Kotelawala Defence University (KDU) for their help in Sri Lanka. We would also like to thank the anonymous referees for their valuable comments and helpful suggestions.

## REFERENCES

[1] Victor Agababov, Michael Buettner, Victor Chudnovsky, Mark Cogan, Ben Greenstein, Shane McDaniel, Michael Piatek, Colin Scott, Matt Welsh, and Bolian Yin. 2015. Flywheel: Google's Data Compression Proxy for the Mobile Web. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI '15)*. Berkeley, CA, USA.

[2] Sohaib Ahmad, Abdul Lateef Haamid, Zafar Ayyub Qazi, Zhenyu Zhou, Theophilus Benson, and Ihsan Ayyub Qazi. 2016. A View from the Other Side: Understanding Mobile Phone Characteristics in the Developing World. In *Proceedings of the 2016 ACM on Internet Measurement Conference (IMC '16)*. ACM, New York, NY, USA.

[3] Christina Aperjis, Michael J. Freedman, and Ramesh Johari. 2008. Peer-assisted Content Distribution With Prices. In *Proceedings of the 2008 ACM Conference on Emerging Network Experiment and Technology, CoNEXT 2008, Madrid, Spain, December 9-12, 2008.*

[4] L. R. Ariyasinghe, C. Wickramasinghe, P. M. A. B. Samarakoon, U. B. P. Perera, R. A. P. Buddhika, and M. N. Wijesundara. 2013. Distributed Local Area Content Delivery Approach with Heuristic Based Web Prefetching. In *8th International Conference on Computer Science Education (ICCSE '13)*.

[5] Anirudh Badam, KyoungSoo Park, Vivek S. Pai, and Larry L. Peterson. 2009. HashCache: Cache Storage for the Next Billion. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI '09)*. Berkeley, CA, USA.

[6] M. Belshe, R. Peon, and M. Thomson. 2015. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540 (Proposed Standard). (May 2015). http://www.ietf.org/rfc/rfc7540.txt

[7] Michael Butkiewicz, Harsha V. Madhyastha, and Vyas Sekar. 2011. Understanding Website Complexity: Measurements, Metrics, and Implications. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference (IMC '11)*. ACM, New York, NY, USA.

[8] Michael Butkiewicz, Daimeng Wang, Zhe Wu, Harsha V. Madhyastha, and Vyas Sekar. 2015. Klotski: Reprioritizing Web Content to Improve User Experience on Mobile Devices. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*. Oakland, CA.

[9] Tae-Young Chang, Zhenyun Zhuang, A. Velayutham, and R. Sivakumar. 2007. Client-side Web Acceleration for Low-bandwidth Hosts. In *Fourth International Conference on Broadband Communications, Networks and Systems (BROADNETS '07)*.

[10] Jay Chen and Lakshmi Subramanian. 2013. Interactive Web Caching for Slow or Intermittent Networks. In *Proceedings of the 4th Annual Symposium on Computing for Development (ACM DEV '13)*. ACM, New York, NY, USA, Article 5.

[11] Jay Chen, Lakshmi Subramanian, Janardhan Iyengar, and Bryan Ford. 2014. TAQ: Enhancing Fairness and Performance Predictability in Small Packet Regimes. In *Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14)*. ACM, New York, NY, USA.

[12] Jay Chen, Lakshminarayanan Subramanian, and Jinyang Li. 2009. RuralCafe: Web Search in the Rural Developing World. In *Proceedings of the 18th International Conference on World Wide Web (WWW '09)*. ACM, New York, NY, USA.

[13] Marshini Chetty, Srikanth Sundaresan, Sachit Muckaden, Nick Feamster, and Enrico Calandro. 2013. Measuring Broadband Performance in South Africa. In *Proceedings of the 4th Annual Symposium on Computing for Development (ACM DEV '13)*. ACM, New York, NY, USA.

[14] Selenium Contributors. 2017. SeleniumHQ Browser Automation, Version 3.5.2. http://www.seleniumhq.org/. (2017). Accessed: 2017-09-09.

[15] Dan Duchamp et al. 1999. Prefetching Hyperlinks.. In *USENIX Symposium on Internet Technologies and Systems*. 12–23.

[16] Li Fan, Pei Cao, Wei Lin, and Quinn Jacobson. 1999. Web Prefetching Between Low-bandwidth Clients and Proxies: Potential and Performance. In *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '99)*. ACM, New York, NY, USA.

[17] R. Fielding, M. Nottingham, and J. Reschke. 2014. *Hypertext Transfer Protocol (HTTP/1.1): Caching*. RFC 7234. RFC Editor. http://www.rfc-editor.org/rfc/rfc7234.txt http://www.rfc-editor.org/rfc/rfc7234.txt.

[18] The Apache Software Foundation. 2017. Traffic server. http://trafficserver.apache.org/. (May 2017). Accessed: 2017-05-06.

[19] Michael J. Freedman, Eric Freudenthal, and David Mazires. 2004. Democratizing Content Publication with Coral. In *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation (NSDI '04)*. San Francisco, CA, USA.

[20] ITU. 2013. Mobile users in developing regions:. https://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2013-e.pdf. (January 2013). Accessed: 2017-05-09.

[21] David Lloyd Johnson. 2013. *Re-architecting Internet Access and Wireless Networks for Rural Developing Regions*. Ph.D. Dissertation. Santa Barbara, CA, USA. Advisor(s) Belding, Elizabeth M. AAI3559800.

[22] Wonho Kim, KyoungSoo Park, and Vivek S. Pai. 2012. Server-assisted Latency Management for Wide-area Distributed Systems. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference (USENIX ATC '12)*. Berkeley, CA, USA.

[23] Zahir Koradia, Goutham Mannava, Aravindh Raman, Gaurav Aggarwal, Vinay Ribeiro, Aaditeshwar Seth, Sebastian Ardon, Anirban Mahanti, and Sipat Triukose. 2013. First Impressions on the State of Cellular Data Connectivity in India. In *Proceedings of the 4th Annual Symposium on Computing for Development (ACM DEV '13)*. ACM, New York, NY, USA.

[24] Zhenhua Li, Christo Wilson, Tianyin Xu, Yao Liu, Zhen Lu, and Yinlong Wang. 2015. Offline Downloading in China: A Comparative Study. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference (IMC '15)*. ACM, New York, NY, USA.

[25] M. Liljeberg, H. Helin, M. Kojo, and K. Raatikainen. 1996. Mowgli WWW Software: Improved Usability of WWW in Mobile WAN Environments. In *Global Telecommunications Conference (GLOBECOM '96)*. London, UK.

[26] Roberto Peon Mike Belshe. 2016. SPDY Protocol - Draft 3.2. https://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3-2. (2016). Accessed: 2017-02-04.

[27] Ravi Netravali, James Mickens, and Hari Balakrishnan. 2016. Polaris: Faster Page Loads Using Fine-grained Dependency Tracking. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16)*. Santa Clara, CA, USA.

[28] Venkata N Padmanabhan and Jeffrey C Mogul. 1996. Using Predictive Prefetching to Improve World Wide Web Latency. *ACM SIGCOMM Computer Communication Review* 26, 3 (July 1996).

[29] KyoungSoo Park and Vivek S. Pai. 2006. Scale and Performance in the CoBlitz Large-file Distribution Service. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation (NSDI '06)*. Berkeley, CA, USA.

[30] KyoungSoo Park, Vivek S. Pai, Larry Peterson, and Zhe Wang. 2004. CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation (OSDI '04)*. Berkeley, CA, USA.

[31] Charles E. Perkins, Elizabeth M. Belding, and Ravi Jain (Eds.). 2008. *Proceedings of the 2008 ACM Workshop on Wireless Networks and Systems for Developing Regions, San Francisco, California, USA, September 19, 2007*. ACM.

[32] Scikit-learn. 2016. Scikit-learn: Support Vector Machines. http://scikit-learn.org/stable/modules/svm.html. (May 2016). Accessed: 2017-02-05.

[33] Rijurekha Sen, Hasnain Ali Pirzada, Amreesh Phokeer, Zaid Ahmed Farooq, Satadal Sengupta, David Choffnes, and Krishna P. Gummadi. 2016. On the Free Bridge Across the Digital Divide: Assessing the Quality of Facebook's Free Basics Service. In *Proceedings of the 2016 ACM on Internet Measurement Conference (IMC '16)*. ACM, New York, NY, USA.

[34] Shailendra Singh, Harsha V. Madhyastha, Srikanth V. Krishnamurthy, and Ramesh Govindan. 2015. FlexiWeb: Network-Aware Compaction for Accelerating Mobile Web Transfers. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom '15)*. ACM, New York, NY, USA.

[35] Internet Society. 2014. Global Internet Report 2014. https://www.internetsociety.org/sites/default/files/Global_Internet_Report_2014.pdf. (April 2014). Accessed: 2017-05-09.

[36] Emily Stark, Lin-Shung Huang, Dinesh Israni, Collin Jackson, and Dan Boneh. 2012. The Case for Prefetching and Prevalidating TLS Server Certificates. In *19th Annual Network and Distributed System Security Symposium, NDSS*. San Diego, California, USA.

[37] Srikanth Sundaresan, Nazanin Magharei, Nick Feamster, and Renata Teixeira. 2012. Accelerating Last-mile Web Performance with Popularity-based Prefetching. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '12)*. ACM, New York, NY, USA.

[38] UNCTAD. 2013. Lack of infrastructure in developing regions:. http://unctad.org/en/PublicationsLibrary/ier2013_en.pdf. (December 2013). Accessed: 2016-05-09.

[39] Arun Venkataramani, Praveen Yalagandula, Ravi Kokku, Sadia Sharif, and Michael Dahlin. 2002. The Potential Costs and Benefits of Long-term Prefetching for Content Distribution. *Computer Communications* 25, 4 (2002), 367–375.

[40] W3C. 2016. HTML5 prefetching. https://www.w3.org/TR/2014/REC-html5-20141028/links.html#link-type-prefetch. (May 2016). Accessed: 2016-05-09.

[41] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. 2013. Demystifying Page Load Performance with WProf. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI '13)*. Berkeley, CA, USA.

[42] Xiao Sophia Wang, Arvind Krishnamurthy, and David Wetherall. 2014. How Much Can We Micro-Cache Web Pages?. In *Proceedings of the 2014 Conference on Internet Measurement Conference (IMC '14)*. ACM, New York, NY, USA.

[43] Xiao Sophia Wang, Arvind Krishnamurthy, and David Wetherall. 2016. Speeding up Web Page Loads with Shandian. In *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016*. 109–122.

[44] Zhen Wang, Felix Xiaozhu Lin, Lin Zhong, and Mansoor Chishtie. 2011. Why Are Web Browsers Slow on Smartphones?. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications (HotMobile '11)*. ACM, New York, NY, USA.

[45] Zhen Wang, Felix Xiaozhu Lin, Lin Zhong, and Mansoor Chishtie. 2012. How Far Can Client-only Solutions Go for Mobile Browser Speed?. In *Proceedings of the 21st International Conference on World Wide Web (WWW '12)*. ACM, New York, NY, USA.

[46] Qiang Yang, Haining Henry Zhang, and Tianyi Li. 2001. Mining Web Logs for Prediction Models in WWW Caching and Prefetching. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)*. ACM, New York, NY, USA.

[47] Yasir Zaki, Jay Chen, Thomas Pötsch, Talal Ahmad, and Lakshminarayanan Subramanian. 2014. Dissecting Web Latency in Ghana. In *Proceedings of the ACM Internet Measurement Conference (IMC '14)*. Vancouver, BC, Canada.

[48] Kaimin Zhang, Lu Wang, Aimin Pan, and Bin Benjamin Zhu. 2010. Smart Caching for Web Browsers. In *Proceedings of the 19th International Conference on World Wide Web (WWW '10)*. ACM, New York, NY, USA.

[49] Wenxuan Zhou, Qingxi Li, Matthew Caesar, and P. Brighten Godfrey. 2011. ASAP: A Low-latency Transport Layer. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies (CoNEXT '11)*. ACM, New York, NY, USA.