Interactive Web Caching for Slow or Intermittent Networks

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

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

ABSTRACT

We explore the limitations of existing caching mechanisms in slow networks and propose a new model of web caching designed for developing regions called *interactive caching*. Unlike conventional caching, interactive caching makes interacting with the cache the focus of web browsing when the connection is bad. Interactive caching achieves this by organizing the cache into topics for presentation to the user, optimizing for latency, and unaliasing cached content. In this paper we implement a prototypical version of interactive caching that includes: topic identification and presentation, a latency aware value function, DNS caching, and missing hyperlink suggestions. We evaluate our system based on a system implementation and web traces from multiple web cache deployments across different geographic locations in developing regions. We show how interactive caching can dramatically improve the user experience for slow connections by allowing users to explore the cache using trending topics that cover 60 - 80% of requests and reducing page load times by up to 72.86%.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]: Network Architecture and Design

General Terms

Performance, Human factors, Design, Experimentation

Keywords

caching; developing regions; latency; web

1. INTRODUCTION

The web is largely unusable or prohibitively slow in many regions in the developing world due to poor network connectivity. Basic connectivity and bandwidth continue to be limited in these regions [11, 38]. When connections do exist, intermittent connectivity and latencies to servers become the bottleneck. Under poor connectivity conditions such as these, conventional web optimizations provide limited benefits. Typical network-level optimizations are severely constrained by the lack of reasonable bandwidth and conventional caching results in decreasing cache hit rates as the web grows in both size and complexity.

These *global* trends in web growth and degrading performance despite bandwidth improvements are gradually being recognized by the mainstream internetworking community; several recent works argue for a significant rearchitecting of the Internet [27]. Unfortunately, these fundamental trends are even more detrimental to web performance in developing regions. While there are works that have proposed changes to web access under poor connectivity [9, 13, 17, 19, 32–34, 46], more aggressive solutions are needed if long-term web growth continues to outpace the trailing tail of connectivity in developing regions.

In this work we propose a novel way of thinking about caching in constrained network conditions, called *interactive caching*. In interactive caching the focus of the user experience is the interaction with the cache when the connection is slow. To make caches interactive we maximize their utility in three ways: organize the cache into topics for presentation to the user, optimize for user-perceived latency, and unalias content.

Two observations motivate the idea of organizing and presenting cached contents. First, it has been observed that people in small communities in developing regions tend to have similar interests [14,17,32,43]. Second, the asynchronous web browsing model proposed in recent works [19,46] present an interesting opportunity to interject additional functionality into the web interaction loop not previously possible. Following these two observations, we propose organizing the cache into *topics* so that it exposed opened up to the user to be more easily explored. Just as conventional caching exploits the spatial and temporal locality across object requests to predict the usefulness of objects for future requests, interactive caching similarly exploits topic locality to predict interesting themes for the user. In this work we propose to identify topics by identifying patterns from three streams of user request information: URL domain names, search query terms, and text content on requested pages.

Optimizing for latency is a shift in focus from bandwidth as the bottleneck being the assumption to latency or intermittency being the bottleneck. Examples of this include latency aware caching metrics and DNS caching. Unaliasing content follows from the observation that when a user requests a specific web page indicated by a hyperlink and that page does not exist in the cache, it is possible that the *information* the user cares about is actually available on a cached page. We should help users find that information, whether it is through search or alternate pages.

Interactive caching improves both system level cache performance and user perceived performance. From an end-user perspective in-

ACM DEV 4, December 6–7, 2013, Cape Town, South Africa.
Copyright 2013 ACM 978-1-4503-2558-5/13/12 ...$15.00.
http://dx.doi.org/10.1145/2537052.2537057.

teractive caching aims to provide three direct benefits: (a) Users are presented with an organized view of trending topics in the cache to help them discover its contents; (b) Users experience faster page loads overall because the cache is optimized for latency; (c) Users are given suggestions for similar pages when a specific hyperlinked URL is not in the cache, which streamlines browsing.

This paper is organized as follows: First, in Section 2, we motivate the need for further study of caching for developing regions and discuss the limitations of existing solutions. In Section 3 we describe the design of a interactive caching framework and explain why organizing the cache, optimizing for latency, and unaliasing content make sense. We then focus on cache topics and presentation in Section 4; we describe the three types of topics that we considered: domain, query, and content topics. In Section 5 and Section 6 we discuss latency-centric caching ideas and unaliasing content, respectively. In Section 7 we describe our implementation and in Section 8 we demonstrate the potential of topics as the organizational basis for presentation using a trace-driven analysis from four real-world web logs gathered from different web cache deployments. Finally, we discuss related work in Section 9 and conclude in Section 10.

2. MOTIVATION

To motivate the need for a new caching abstraction for networks in developing regions, we outline two fundamental trends pertaining to the rapid growth in the complexity of web pages as compared to the relatively slow growth of connectivity. We then summarize the various problems with web browsing and discuss why conventional solutions are not sufficient.

2.1 Web Growth vs Connectivity

Web pages have significantly grown in size over the years and have out-paced the growth of connectivity in many parts of the developing world. In addition, an average web page has also increased considerably in complexity with a large number of objects assembled from different domains. Information gathered from several sources [3,6,12,23] indicating a $30-50$ fold increase of these metrics in the past decade.

In contrast, connectivity has not grown at the same rate in the developing world, and growth has been non-uniform across countries and regions. Global dialup and broadband penetration growth in developing regions is isolated to a select few countries in each continent [1,7]. Broadband penetration in developing countries is only 4.4 subscriptions per 100 people compared to 24.6 in developed countries in 2010; Africa, in particular has penetration rates of less than 1% [7]. Furthermore, affordable connectivity in these countries is typically limited to high-density urban areas. This mismatch between web growth and available connectivity is the primary cause for poor web performance in developing regions. Based on these trends, we believe that waiting for connectivity to catch up with web growth world-wide is not an option.

Increasing web page complexity is a problem that plagues not only slow network connections, but fast networks as well. When loading a single page causes subsequent requests for many assets from multiple servers performance generally degrades. Protocol improvements such as Google's SPDY attempt to address some of these issues for already fast networks [5], but in slower networks performance degradation is much more pronounced and SPDY's optimizations do not focus on the these network conditions.

Figure 1 illustrates the requests that are required for loading a typical webpage ("yelp.com"). We requested this page from a 3G connection in Accra, the capital of Ghana (August 2012). The DNS wait time is in blue, server response wait time in purple, and the transfer of data is in gray. We observe that the single initial page request resulted in 78 objects being downloaded from 11 separate domains requiring as many DNS requests. The total page load time is 25.66s. We can also see how despite the optimizations of multiple TCP connections opened by the browser and HTTP pipelining, the performance bottleneck in this scenario is actually DNS resolution latency and TCP handshakes.

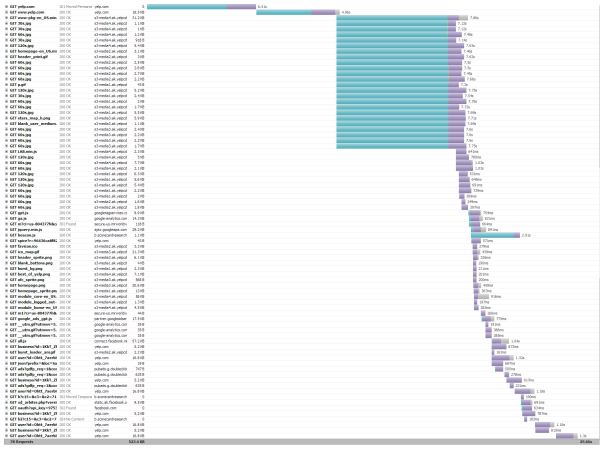


Figure 1: Waterfall chart of object requests for rendering the request to "yelp.com" from a 3G connection in Accra, Ghana (August 2012). DNS wait time is in blue, server response wait time in purple, and reception of data is in gray. The main performance bottlenecks are the latency associated with DNS requests for multiple domains and responses from many servers.

2.2 Limitations of Existing Solutions

Beyond the evolution of the web, the problems surrounding web access typically have to do with a lack of consideration by the mainstream internetworking community of the differences in the networking in these under-resourced regions. Unfortunately, what are conventionally the corner cases are the common case for the next billion internet users. In the previous example, web optimizations such as parallel TCP connections at the application layer, and TCP pipelining in the HTTP protocol are well intentioned, but illustrate how optimizations for high bandwidth low latency networks are not universally effective. We briefly outline why the application of some of straightforward existing ideas to this problem are insufficient.

Why not improve network connectivity? The first question that may arise is why not focus on improving network connectivity? Networks in developing regions suffer from numerous technical challenges ranging from power quality issues to poor connectivity to low bandwidth [11]. Improving connectivity is the most direct approach [42,45], but such solutions often may not feasible in all contexts due to basic economics and a variety of other factors. Mobile data connectivity has penetrated in many developing regions, but are limited by relatively high prices. Even universities that have high bandwidth connectivity often experience extremely high levels of sharing [17]. Finally, the fact that content servers are simply very far away geographically contributes to the latency problem.

Why not conventional caching? One conventional optimization essential to web performance is web caching. Unfortunately, standard web cache deployments in many developing regions typically yield very poor cache hit rates ranging from 15% to 30% [30]. In slow networks even a few cache misses can result in long la-

tencies. The existing caching standard [25] was simply not designed for or suitable for complex web services on networks with low bandwidths, extremely high latencies or periods of complete disconnectivity.

Caching in the context of developing regions: The body of literature regarding web caching is extensive, but caching mechanisms for developing regions have been explored only to a limited extent [9,14,32,34]. As web content becomes more fragmented and dynamic objects are commonplace, stale and nocache objects are already being cached by aggressive solutions to trade-off content freshness for availability. Isaacman and Martonosi's C-LINK deployment in 2009 showed that the cache hit rate at a rural school in Nicaragua connected to the Internet by a mechanical backhaul was only approximately 20% [33]. However, if collaborative caching were included for these small communities, the cache hit rate dramatically rose to 80%. In 2010, Chen found (ELF dataset) that the cache hit rate at an urban school in India was only approximately 31.1% if nocache and stale pages were *included* [14]. Other caching works have focused on low cost hardware or specific network configurations [9, 31, 32].

3. INTERACTIVE CACHING OVERVIEW

The basic goal of interactive caching, as the term suggests, is to provide a usable and interactive web experience for users behind slow and intermittent networks by enabling users to interact with a large cache. The physical setup of our interactive caching system is a single or a collection of end-users behind a slow network that all share a common proxy cache located near the edge of the slow network link. Our interactive caching system is built around four key ideas:

A New Presentation Layer: All users in the system are aware that they are behind a slow or intermittent network and explicitly use an intermittency-aware asynchronous web browsing system which is aware of where the interactive cache resides. When the network is too slow, intermittent, or unavailable, the interactive cache can explicitly serve content to the users. The logical components of an interactive cache could all run on a single physical machine.

Organizing the Cache based on Topics: To enable users to easily search and navigate the cached contents, interactive caching organizes caches based on the abstract concept of *topics*, which simply represents a group of related pages with a common theme. We discuss different types of topics that our current system supports. Users can search for specific topics or be made explicitly aware of the trending topics which in turn increases their potential interaction with the cache.

Optimizing for Latency: Since we operate in environments where latency is often the primary bottleneck, interactive caching supports specific optimizations that are tailored to optimize for latency. Given the observation of high DNS response times in these network conditions, interactive caching supports DNS caching. In addition, we introduce a specific latency-sensitive caching mechanism which allows cache eviction and management at different granularities (pages, topics, domains) using latency weighted values.

Unaliasing Content: Under high page load time conditions, when users are requesting specific pages not present in the cache, interactive caching provides a simple way for users to identify alternative cached pages that might provide similar content or in some cases, identical content. Instead of relying solely on existing hyperlinks for navigation, interactive caching uses an *unaliasing content* mechanism that allows potentially interchangeable content to be presented as possible alternatives when the original hyperlinked pages do not exist in the cache.

In the next sections, we will describe these design ideas in greater detail.

4. TOPICS AND PRESENTATION

One key challenge in interactive caching is identifying topics within a cache. Once the topics are identified, then the interactive cache can present them to the user in an organized fashion.

Conceptually, a topic is simply a set of pages that describes an interest of users. In our interactive caching system topics of interest are expressed in three different ways: (a) Domain; (b) Query; (c) Content. Domain topics cluster all requests of the same domain and sub-domains together. This is useful when users are interested in all web pages authored by the same content creator. Query topics are extracted as top search terms across users. This is designed to aggregate trending search topics. Content topics are extracted by analyzing actual content from the cached pages.

In our system each topic consists of: (a) name of the topic, (b) a set of pages belonging to a topic, and (c) value of the topic. The name of the topic is simply a label for the topic used for identification and . The description may be automatically extracted from the pages belonging to the topic. Our system automatically generates topics from three sources of information: requested URLs, issued search queries, and text content from downloaded pages. The set of pages that belong to a topic are extracted automatically and are used by the cache to display pages belonging to a topic during navigation. The value of a topic is used by the cache to prioritize the most important topics for presentation to the user.

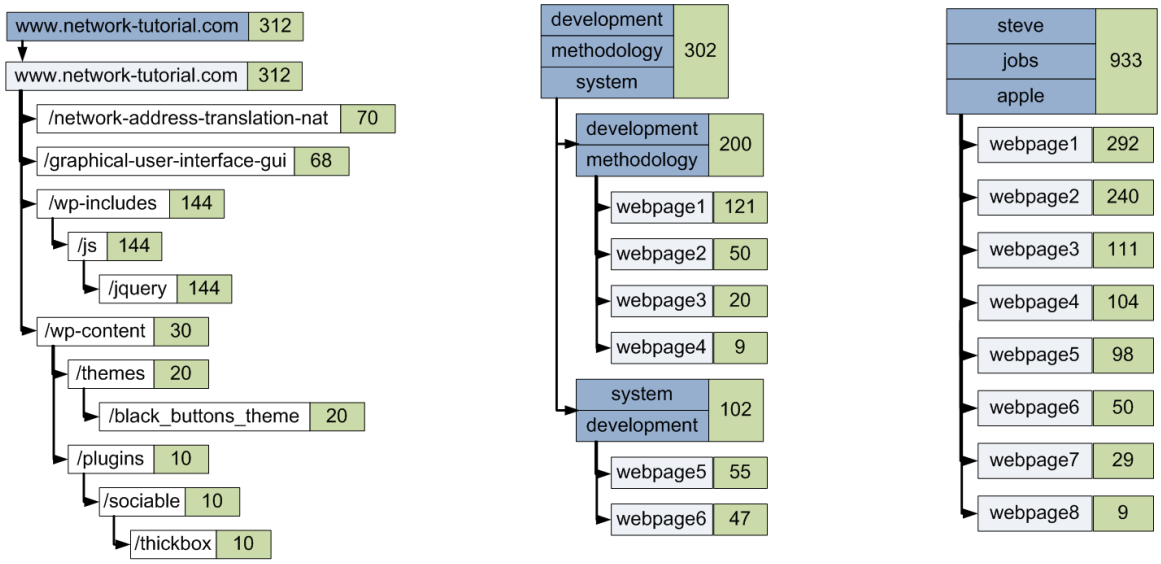
4.1 Domain Topics

Domain topics aggregate different requests to the same domain. A good example of a useful domain topic is a popular news site. Different users may be interested in different articles in the same news site; grouping all such requests as one domain-topic allows users to get a better idea of the content from the news site that has already been cached. To identify domain topics, the system examines the URL patterns that occur to identify domains and sub-domains appearing frequently enough in the cache.

The domain-topic extraction engine groups web page requests for a given domain into a hierarchical tree of buckets. At the top of the tree is the domain and branching out from these are the sub-domains, and sub-directories within the domains. The leaves of this tree are specific URLs within the domain. Associated with each bucket is a value measure V_b per domain/sub-domain. The leaves of the tree are individual pages with a value measure as a direct function of the time spent fetching the page's objects. The value of any node in the tree is the sum of all the leaves in the sub-tree rooted at the node. An example of a tree structure for a domain topic is shown in Figure 2(a). A domain topic is considered "identified" once a threshold, T, of requests are made that belong to the topic. Once a topic is identified, it is eligible for presentation.

4.2 Search Topics

Search topics are constructed from user search sessions. The system tracks individual search query requests from the users and split requests into search sessions based on queries to search engines, query overlap, and time of request. On a per-client basis, if a query is made that overlaps with a previous query made within the last time threshold, S seconds, then the query is considered as part of the same search session. Otherwise, a new search session is created. All requests made while a search session is active belong to the current search session. The union of all search terms in a search session represents a "bag of search words" that can potentially be



(a) Domain Topic (b) Search Topic (c) Content Topic

Figure 2: Example values for (a) domain topics, (b) search topics, and (c) content topics. Values are hierarchical and cumulative. Dark blue topics are displayed names, light blue highlights indicate web links for display, and object values are in green.

used as the topic name label.

In our system, we take a simple and scalable approach to define search topics without involving any complicated computational linguistic steps to understand the meaning and relationships across words. We use a simple dictionary to remove all commonly occurring words from the list of search words and extract a list of 'query-like' phrases from the user requests in a search session. A search session is also associated with the individual sites visited by the user during the session. Thus, each search session can be represented as a 2-level tree comprising of a filtered collection of search words at the root and the websites visited in the session along with their individual value measures. The value of the root is the sum of the values of the leaf nodes.

We use two simple rules for merging search topic trees. If the list of words in one search topic tree exactly overlap with another search topic tree, we can merge the two roots into a single node with the combined values and join all the child nodes to the new root. If two search topic trees partially overlap in the word set, then we can merge the search trees by creating a new root node with the union of the search terms of the individual search topics. An example of such a merged search topic tree is shown in Figure 2(b). From a user perspective, depending on the search queries, they can be shown the search topic tree as a means of navigating relevant pages corresponding to a search topic.

4.3 Content Topics

The output of content topics, as with search topics, is a set of descriptions that convey the areas of interest that users spend the most time on. However, given the time and resource constraints of the cache system, performing a clustering calculation on the full set of document content is impractical. This restriction implies a need to drastically reduce the set of documents considered and the feature set used. We use standard information retrieval techniques to extract the commonly occurring features (1-gram or 2-gram keywords) across different documents. Clustering the reduced set of features is straightforward and can be performed using any standard clustering algorithm. An example of a content topic tree is shown in Figure 2(c).

4.4 Presentation and User Interface

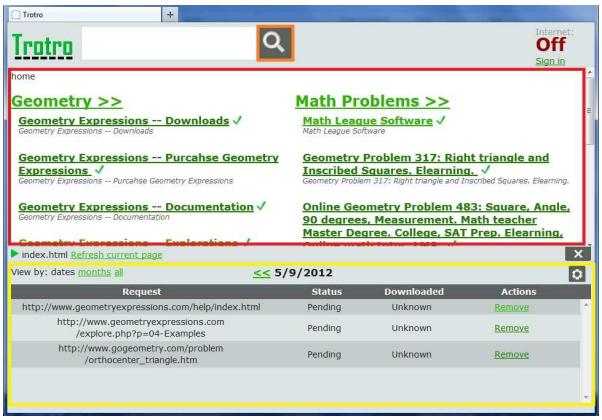


Figure 3: Screenshot of the web browser interacting with the intermittent proxy. The main browsing pane on the homepage (red) contains topics and is augmented with a search box at the top (orange) and a request queue at the bottom (yellow). The top right corner indicates the Internet connection status (currently off).

Interactive caching techniques may be combined and are synergistic with recently developed asynchronous web browsing systems [19, 46]. We borrow liberally from previous works on asynchronous browsing interfaces and layouts. In such systems, requests by clients are served directly by the proxy. These asynchronous browsing proxies already expose the cache contents to users via two simple mechanisms: offline search and browsing. Offline search is when the user decides to search through the cached web pages when the network is too slow. The system will return a list of results and the user will then click on links to navigate

through the cached pages. It is also possible for the user to directly specify URLs to request those pages directly.

In interactive caching, we increase the level of interaction to the cache and expose more of the cached contents by exporting an organized view of the cache to the presentation layer. In real-time, the cached pages are gathered into topics and presented to the user as links for navigation. Figure 3 shows a screenshot of an example asynchronous browsing interface [37]. The search box and request queue are respectively outlined orange and yellow. The main browsing pane (outlined in red) is typically pre-populated with static cached content categories, the system's homepage, or tools (e.g. dictionary/wikipedia/etc.). Rather than pre-populating the cached contents statically, we use this space to present the interactive cache's real-time trending topics. We defer a more detailed study into the design of user interfaces and integration of interactive caching to future work.

5. OPTIMIZING FOR LATENCY

There are many ways to make latency the focus of web optimizations. We briefly introduce two of these possibilities at the caching level that can make a big difference in web performance.

5.1 Latency-sensitive Caching

To make web cache policy decisions we need to place a cost or value on each page or object as a measure of how costly it is to store or how valuable it is to users. Conventionally, cost and value are typically considered synonymous and Least Recently Used (LRU) or similar policies that may take into account object size are commonly used in web cache eviction policies. In interactive caching we not only have to perform eviction, but also presentation. We observe that object size is not a particularly good measure of how valuable an object is to the user. In light of these considerations, we explicitly decouple the eviction policy from presentation and the cost from value. In other words, a conventional cost is associated with all web objects stored in the cache and these costs are used in the cache eviction policy, e.g. size adjusted LRU. Separately, the value function, latency adjusted LRU, is used in the presentation policy of the cache.

Since the emphasis of presentation (interest) is different from cache eviction (storage) we create a new measure for making decisions about prioritizing pages for presentation. The conventional measure of the cost of an object in the cache is the number of times it was requested or the size of the object. In a high bandwidth network scenario, this reflects the optimal behavior: smaller objects are downloaded faster and therefore the time spent transferring on a cache miss would be minimized. However, for constrained networks, the relationship between object size and transfer time fluctuates significantly between different network conditions as observed earlier. For example, a single large object may download faster than several smaller objects due to connection setup latency, latency to particular servers, or extreme bandwidth contention. Across these disparate network scenarios, object size is no longer a consistent and accurate measure of the time spent transferring. instead, the time spent downloading an object is better implicit measure of the user's valuation of the object, because the user was willing to wait for it to download.

Measuring the value of objects is straightforward - we simply record the amount of time spent fetching a resource; this stored 'time to fetch' is the value for the resource. This value function explicitly takes into account the reality of the web browsing experience for users in high-latency environments where the value of fetching N bytes may vary radically. When objects are present in the cache the value function is nearly zero. Therefore, the value of most objects is the sum of the initial time to download the object plus many tiny access times caused by subsequent requests resulting in cache hits.

In normal situations where request latencies are relatively stable, caching decisions made based on object size reflect accurately the resources used for retrieving and storing the object. However, in constrained situations where this is often not true, the estimated value of an object will be extremely inaccurate. We use the scenario in a previous work, Event Logger for Firefox (ELF) [14], to illustrate this problem. ELF is a Firefox extension that optimizes web page load times using aggressive caching (of somewhat stale contents) and prefetching while logging all requests. The ELF trace data was collected from a shared school network in peri-urban India where, relative to intermittent or high latency networks, the network was "good". The authors found that by aggressively optimizing for performance, page load times could be reduced by a factor of 2.8x. We used the same traces to generate Figure 4, which shows the variation in user-perceived latency for downloaded (cached and uncached) objects of a given size. We observe that for this slightly constrained shared network connection, the variation for a given object size is up to 2 orders of magnitude for uncached objects around 100KB in size; in other words, basing caching decisions on object size would fail to account for nearly 2 orders of magnitude in user-perceived latency similar sized objects! This disparity is further magnified for web pages that consist of around 30 separate objects. This is surprising given that ELF was able to achieve significant speedups already, and suggests that further gains are possible by taking latency into account.

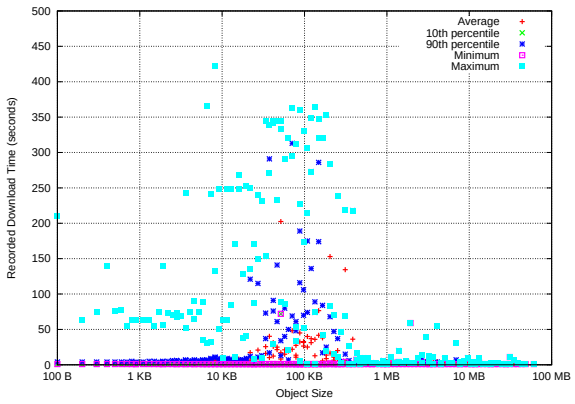


Figure 4: Per-object download time from ELF log bucketed by size showing values for average, min, max, 10th percentile, and 90th percentile.

5.2 DNS Caching

As we observed in Section 2, DNS query lookups can dominate the overall web page request time, and adding a DNS cache could help address this problem. To assess how much potential benefit there is in deploying DNS caching we perform some cursory experiments from behind a broadband Wifi connection in Ghana. In August 2012 we downloaded Alexa's top 100 web pages for Ghana using Firefox v12 over a 10Mbps broadband connection while recording the traffic in the HTTP Archive (HAR) format [29]. The HAR file format does not preserve dependencies among web page assets so it is not possible to automatically calculate how well DNS caching would perform. To address this, we manually calculated the dependency tree for the first few bottleneck assets of a webpage requested to render the complete page. This is a con-

servative estimate since further dependencies are possible, but as a lower bound we found that in our dataset that the DNS request time was responsible for up to 72.86% of the page load time. This means that as long as the DNS mappings for a page are cached, subsequent requests for the page would load nearly 4 times faster in this environment.

Despite our conservative accounting method, DNS caching has obvious potential for dramatically improving page load time on even broadband networks where latencies are relatively high. As a part of our interactive cache we would simply maintain a DNS cache that stores the DNS to IP mappings for all sites accessed. The entries in this cache are associated with a relatively long lifetime to prevent premature expiration. Further optimizations such as DNS entry eviction policy, DNS prefetching, request scheduling based on object dependencies are possible, but are not the focus of this work.

6. UNALIASING CONTENT

In a well-connected network when a web page is requested that results in a cache miss the browser automatically fetches the page from the content server on the Internet; since the connection is fast, this process is transparent and seamless. Behind a slow network a cache miss is more frustrating to the user since it takes a longer time to fetch the missing page and associated resources. If the connection is completely down, then the browser returns error since it cannot download the missing page and the user has no way to make progress. The key idea of unaliasing content is that when a user requests a web page, the *information* contained in the page could be found in a *different* page in the cache. Therefore, in such scenarios when the connection is down or even just very slow, requests that would result in cache misses are instead diverted to existing pages in the cache that contain the same information.

As with the idea of latency aware optimizations, there are many different ways to unalias content. In fact, existing asynchronous systems already employ one method for unaliasing content by allowing users to search the cache contents. This allows users to find the information that they are focused on rather than basing the cache functionality on an entirely binary URL-centric concept of cache hits or misses. Here, we suggest a substantially more aggressive mechanism to unalias content: missing link suggestions.

Missing link suggestions are just what the name would suggest; when links are missing from the cache, the user is presented with other pages that could potentially satisfy the same information needs as the linked page rather than waiting for the network to be available. Again, missing link suggestions could be implemented in different ways, but with two basic requirements. First, at the information retrieval level, the algorithm used to suggest hyperlinks should be accurate. Second, at the user interface level, how the links are suggested to the user is critical to the user experience. Here, we briefly discuss a proof of concept of the algorithmic aspect of the information retrieval problem and leave the other aspects for future work.

To show that missing link retrieval is possible, we crawl the web for 120 search topics for a total of approximately 60,000 pages. We crawled our corpus of pages during May, 2013 using some of the topics from previous work on offline web portals [16]. Using this corpus, we conducted some simple retrieval experiments to see the accuracy with which we could retrieve pages in the corpus based solely on the information from the referrer page. Our experiment is as follows: First, we index all 60,000 pages in our corpus. Then, we find all of the hyperlinks where the linked page exists in our total corpus (210 links). Using the referrer page of these hyperlinks we check to see whether we are able to retrieve the linked page *without* using the URL to directly look up the page. We experiment in this manner because if the actual page were able to be retrieved with 100% accuracy, then our retrieval model has perfect performance; therefore, for retrieval of pages that are actually missing, our retrieval model would retrieve highly relevant pages.

Table 1: Simple retrieval model performance on missing link suggestions.

Retrieval Model	P@1	P@10
TF-IDF	0.33	0.53
Language Model	0.40	0.60
BM25	0.38	0.58
GA	0.63	0.67

We use three standard retrieval models in this first experiment: TF-IDF, LM, and BM25. Due to space constraints we do not include the formulas for these retrieval models, more information about these models can be found in [26]. Our results for 210 queries (generated with the hyperlink anchor text on the referrer page) for the three retrieval models are shown in Table 1. In the Table, P@1 and P@10 indicate the percentage of queries for which the correct page appears at the first and within the top ten ranked positions, respectively. We found that by using only the link anchor text and URL on the referrer page for the search query, the effectiveness was low, but still promising.

Following this first experiment, we then combined different features (e.g. Anchor Text, URL, context, contents of document containing missing link) and used a using genetic programming algorithm for analyzing how far feature combination could improve the performance. Out of 210 queries, we first randomly picked 50 as training used the rest for testing. We ran the genetic programming algorithm up to 100 generations with 50 individuals per generation. After training, we tested the effectiveness on evolved retrieval model on test queries. We found that this algorithm achieved accuracy of 63% for P@1 and 67% for P@10. We believe that even more refined algorithms and features will yield better results, but these numbers imply that approximately 2/3 of the time our missing link suggestion algorithm would suggest a useful link if it exists in the cache.

7. IMPLEMENTATION

From the system organization perspective, interactive caching may either completely subsume conventional caching or be applied as a layer on top of conventional caching. In our implementation, we chose the latter design for a cleaner separation between cache organization and object-level management so as to leave cache eviction largely unchanged.

For this work, we implemented each of the caching mechanisms designed in the previous sections. Our interactive cache is implemented in C# on top of an existing asynchronous browsing system [19]. We chose this system as a platform because we had previous experience with it and it installs easily on Windows machines. Our additions to the code primarily were to include the DNS caching functionality, missing link suggestions, and topic identification and presentation using our new value metric.

Adding DNS caching was relatively simple since Windows already has a DNS resolver the .NET frameworks use. We modified the installer to change the Windows registry to increase the size of the cache and extend the default expiration time. We opted for this most basic implementation since it was easy to make these changes without delving too deeply into the resolver implementation itself.

We expect that adding prefetching and more intelligent expiration of the cache would also be possible, but more involved.

For missing link suggestions we run our genetic programming algorithm described in Section 3 implemented in 3000 lines of C code including query expansion algorithms to boost performance. Identifying missing links to make suggestions for is trivial since the cache can query itself for all URLs. Our missing link suggestion algorithm is fully integrated into our interactive cache using Javascript injection into the served pages. We considered an alternative implementation using a browser extension that queries our cache for the suggestions, but this would necessitate installation of the extension onto all client browsers.

Our cache organization and presentation layer was implemented using in C and C# with dependencies on several open source libraries. The domain and search topic identification routines are approximately 2000 lines of C# code. The content topic extraction was implemented by first collecting all files with text/html MIME type and removing HTML tags using the C# HTMLAgilityPack package [28]. Then, the doc2mat [22] utility was used to convert tokens into the cluto format. Cluto [20], a clustering toolkit, was used to cluster documents into K = 10 clusters for display. For a small cache size of 66MB with 1366 text files. Clustering takes about 40s the first time and about 10s for consecutive calls. In our implementation clustering takes around 2 hours for large caches (150GB) and we are working to improve the clustering performance. We run clustering currently once per hour by default, but this parameter may be tuned for a desirable tradeoff between system performance and responsiveness to changes in the cache. Finally, we modified the user interface with hooks that request our intermittent cache for topics to display the most high value topics using a basic two column grid layout (Figure 3).

8. EVALUATION

In this section, we evaluate the effectiveness of our interactive caching layer using real-world access patterns of web browsing requests from four developing region deployments. We emphasize evaluation of functionality across different contexts to show how our ideas can improve caching. Our system, by design, significantly alters user behavior and access patterns. We therefore focus on independent evaluation of the individual mechanisms of our system rather than a full system evaluation.

Traditionally, cache hit rate has naturally been used as the standard system level metric for evaluating web cache performance. As web content becomes more fragmented and dynamic, cache hit rate declines in its explanatory value and other metrics may be more useful to show system performance (e.g. load time). Interactive caching also expands functionality, changes user behavior, and relaxes the binary concept of a cache hit. Since the goal of organizing the cache is topic identification, we specifically use topic level rather than system level metrics in our evaluation.

We define a new metric "topic hit rate" to capture the idea of having a page in the cache that satisfies a request to *some* degree. A request results in a topic hit if there already exists a topic identified and cached by our system. For example, if a user looks for information about "Steve Jobs" and finds a link to his biography that is not cached, but then searches for or is offered a similar page with the desired information and views that page instead, then the request would be a topic hit for the search topic "Steve Jobs". While the information in the cache may not be exactly what the user wanted, if it was returned by the local search it should be at least somewhat relevant and facilitate further browsing. By design and by definition, this metric is more relaxed than cache hit rate.

8.1 Web Traces

Table 2 outlines the properties of the datasets, and the environments from which they were gathered. The datasets are varied and have many interesting properties beyond what we describe here for comparison. We only highlight the relevant differences as they relate to our results.

8.2 Identifying Useful Topics

To evaluate whether our system is able to find useful topics we use a web trace gathered from a computer lab at a secondary school near Nairobi, Kenya using a CIP system [16] via a mechanical backhaul. Overall, this trace contains over 100000 requests and over 1400 search queries gathered over a period of four months. The usage of the cache is bursty due to multiple scheduling and classroom constraints, only 41 days out of 160 have any activity. Days with no activity are omitted. Since this trace does not contain the actual object payloads we are not able to generate the content topics. On most days only a handful of "big" topics dominate the set of requests. Table 3 shows an example of the topics our system identified and their value on an arbitrary day. Given that the computer lab was being used to teach Information Technology courses, these topics appear to be relevant to the topics covered in class.

8.3 Topic Recurrence and Stability

We next attempt to understand the utility of topics over time: "how long does it take to detect a topic, and how useful is the topic over time?" Figure 5 shows a semi-log plot of the cumulative number of requests over time (Note: time here is in hours after the first request belonging to the topic) for a few of the top topics by topic coverage. The threshold $T = 3$ is shown for reference (in yellow). Again, we make some observations. First, we observe that most of the domain topics are identified within the first 24 hours. Second, the identified topics recur regularly over time. This figure also shows that despite the requests being bursty over short timescales of hours, topic stability is fairly regular across days. The long quiet periods are due to intermittent use.

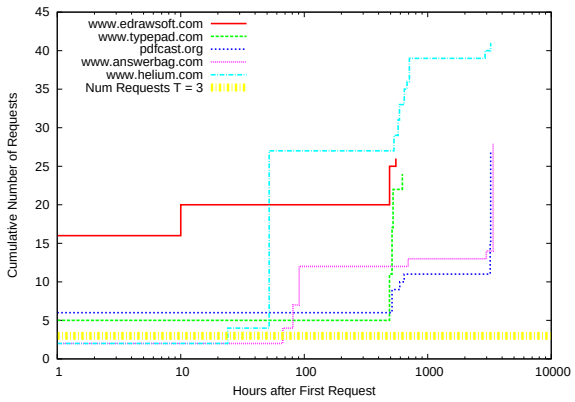


Figure 5: Semi-log plot of domain topic trends over time. Note that the x-axis is the number of hours after the first request for a topic and therefore not all topics extend to the end of the graph.

8.4 Topic Coverage and Value Function

Figure 6 illustrates the topic hit rate of user requests as the number of domain topics varies. The top N topics (x-axis) are chosen by the highest value. We constructed these topics from 1432 search queries, and find that after 100 search topics the topic hit rate is

Table 2: Descriptions of web traces used for Figure 8

Dataset	# of Requests	Time Taken	Context
ELF	263,979	17 day sample, 2010	Peri-urban elementary school in Bangalore, India, students and teachers
CIP	110,493	Oct 2010 - Feb 2011	Rural secondary school near Nairobi Kenya, mechanical backhaul, students and teachers
Kelsa+	169,819	1 year, 2009	Office building Bangalore India, used by unskilled office workers, good connection
TEK	64,786	7 year period, 2002 - 2009	Solomon Islands, slow/intermittent connection, local farmers

Table 3: Example topics along with their values according to our value function.

Topic	Value
system development methodology	19559
operating system development	14867
computer types	4139
www.network-tutorial.com	201
msdn.microsoft.com	190
www.cisco.com	183
www.electronicsreviewsnow.com	114
imgs.xkcd.com	73
justin bieber	9
hoax virus	5

greater than 66%. Also, as few as 500 domain topics are sufficient to cover 53% of requests over a period of 4 months.

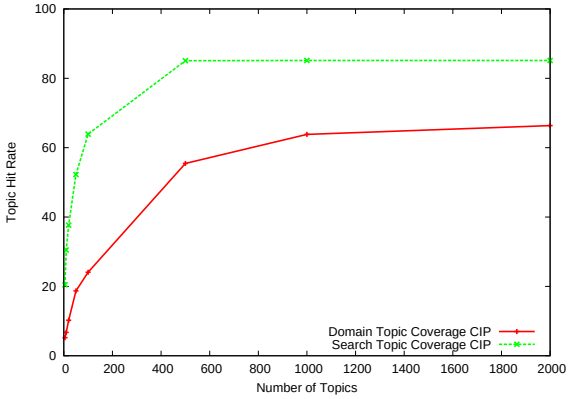


Figure 6: Topic hit rate of domain and search topics across all requested objects.

8.5 Caching Sensitivity Analysis

The previous topic coverage rates were measured using web requests for all objects in our trace. However, as we discussed in Section 2, modern web pages generally require resources from many different domains to render. Since the CIP trace contains information that allows us to differentiate between the user requested page and all dependent pages (i.e. embedded objects, scripted requests, and active content), we filter these out to simulate the performance if all caching standards are ignored and our cache aggressively caches all objects. An alternate way of thinking about this experiment is that it gives an idea of the cache performance per page rather than per object. We also conduct a sensitivity analysis on some of the parameters in our implementation to show that our system does not require significant tuning.

Figure 7 shows the results of this experiment using the CIP trace. We make three observations: First, we observe that our system achieves significantly higher topic coverage than counting individual objects separately and strictly obeying HTTP/1.1 caching standard as compared to Figure 6. Second, we require only few page topics to achieve this high page topic coverage. Approximately 40 completely cached pages will satisfy 65% of requests over a 4 month period. Finally, the resultant topic hit rate is insensitive to choice of the threshold T other than the relatively low gain tail of the graph is truncated.

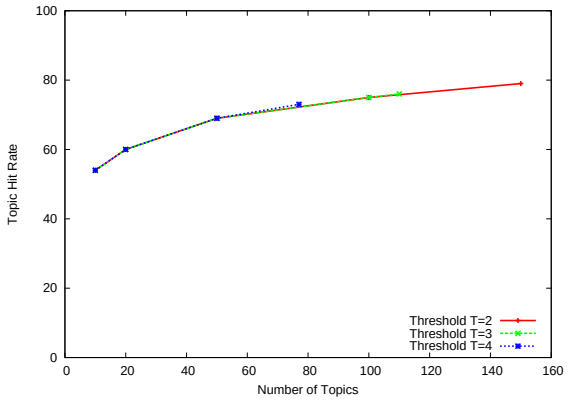


Figure 7: Sensitivity analysis of parameter T (number of user requests before accepting a topic) of topic hit rate of domain topics across total *user* requests.

We also experimented with the sensitivity of search topics on the threshold, S, which is the maximum time between the last search query within which a request will be considered as a part of that query's search session. We found that between 15 and 60 minutes, there was very little increase in search topic coverage rate. We found experimentally that the duration of 15 minutes is reasonable for capturing most real user search sessions.

8.6 Generalizability to Other Contexts

It is promising to show that topics are potentially beneficial for one specific context, but it is significantly more compelling if our system works across contexts. We evaluate our system using each of the web traces from the separate pilot deployments (Table 2) in developing regions across three very different geographical and situational contexts to see whether our results generalize. In this

comparative evaluation, we only compare domain topics here because the page content itself was not available, and not all logs contained enough state information to extract search topics from them. Furthermore, there were very few search queries in the TEK and Kelsa+ logs with which to do a comparison.

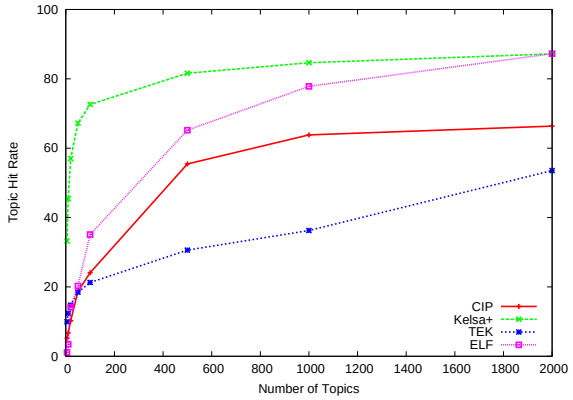


Figure 8: Per-object topic hit rate across four datasets: CIP, Kelsa+, TEK, ELF.

Figure 8 shows the request coverage for a given number of topics across the four datasets. We observe that we actually attain better coverage per topic for the ELF and Kelsa+ traces than our results with the CIP trace. It is important to note that these results are far from a best-case scenario; ideally, the epoch length, when topics are extracted and topic hit rate measured would match the network characteristics. For example, in the case of a low bandwidth connection, topics should be extracted at least once per hour. For mechanical backhauls, topic extraction should occur at at least once per round trip. This would lead to an increase in the short-term topic hit rates. We also found that the request recurrence of the domain topics of each of the traces is qualitatively similar to the CIP trace results in Figure 5 and the time between topic recurrence generally scales with the rate of requests over time.

9. RELATED WORK

There is a large body of literature about web caching in general. In regards to constrained networks and improving the availability of content, Coda introduced idea of disconnected hoarding of often used content for offline use [35]. After the observation that web requests may be modeled as a Zipf-like distribution of independent requests, much of the early work on caching emphasized prediction schemes for exploiting the temporal locality of page requests [10]. This included the design of caching hierarchies and models [47]. More recently, distributed or collaborative web caching systems have been popular areas of research [1, 36]. The unaliasing aspect of our work is the similar to value-based caching [44], which allows matching of data chunks at sub-page granularities. Interactive caching does matching at the per-page level granularity for missing link suggestions, and the purpose is to redirect users rather than to save bandwidth on redundant data.

As with hierarchical caching and collaborative caching, prefetching systems may be implemented in different places in a network. Prefetching is complementary to, but independent from, interactive caching. Web prefetching methods typically emphasize server-side or server-assisted systems for increasing web server performance or client latency [21, 39, 40]. These systems employ various techniques such as probabilistic user modeling or popularity-based prediction by partial match [39] to enhance prefetching accuracy. Other works suggest prefetching between low-bandwidth clients and proxies [24].

There are only a few works on caching for developing regions, but they are otherwise unrelated to interactive caching both in terms of goals and ideas. The C-LINK collaborative caching system improves caching within a village by pooling resources across a set of weak nodes and sharing resources [32, 33]. Recent work extends collaborative caching to support mobile devices [34]. HashCache introduced techniques for scaling up web caching for cheap commodity laptops with limited memory [9]. In contrast to these approaches, interactive caching is user centric and emphasizes reductions in user-perceived latency, unaliasing content, and organizing existing cached contents for presentation. Other developing region specific web optimizations besides caching focus on improving bandwidth utilization through various techniques [4, 15, 31].

The organization of topics in interactive caching is reminiscent of automatically building an ontology. Ontologies of topics have been in active use by Yahoo and AOL for many years, but slowly lost popularity as search engines improved and Internet speeds increased. More recently, the rising interest in semantic web has revitalized ontology based searching, particularly for domain specific web portals or vertical search engines [41]. Interestingly, topics are also popularly used in SMS-based search engines, another type of constrained search [2, 8, 18].

Time Equals Knowledge (TEK) [46] and RuralCafe [19] are web browsing systems for challenged networks that enable web access over poor network connections. These systems use an asynchronous browsing model and allow for local search through the cache. The cache organization and presentation aspects of interactive caching are dependent on the asynchronous browsing model, but the idea of interactive caching is not tied to a particular implementation.

10. CONCLUSION

In this paper we proposed a novel approach to caching in developing regions with slow networks, which we call interactive caching. In our interactive caching model we maximize the utility of web caches in three general ways: organize the cache into topics and present these topics, optimize for latency, and unalias content. We designed and implemented one possible instantiation of these ideas and found several interesting results.

We showed how cache may be organized based on different kinds of primary features (domains, search terms, text contents) and showed how this organization can then be integrated into the presentation of an asynchronous browsing system. Since we found that DNS resolution times were a huge contributor to page load times our system's basic DNS caching could reduce page loads by up to 72.86%. We observed that object size as a cost metric for cache eviction could be replaced in some situations with a latency aware value metric. We decoupled cost of eviction from value in presentation and used our latency values to weight the cached contents for presentation; replacing this metric may more accurately reflect user interest.

To unalias content beyond basic search and sifting through results, we implemented a proof of concept genetic algorithm for missing link suggestions to assist with browsing while the network was unusable. For our sample dataset our prototype algorithm achieves up to 63% link suggestion accuracy. From our implementation we showed the potential of topics as a basis for presentation through trace-based analysis from four intermittent or slow network settings. Using trace-based analysis from diverse settings in developing regions we found that our interactive caching concepts are applicable to a variety of contexts.

11. REFERENCES

[1] Akamai: State of the internet. http://www.akamai.com/stateoftheinternet/.

[2] Google sms. http://www.google.com/sms.

[3] Let's make the web faster - google code. http://code.google.com/speed/articles/web-metrics.html.

[4] Loband. http://www.loband.org.

[5] Spdy. http://www.chromium.org/spdy/spdy-whitepaper.

[6] Websiteoptimization.com. http://www.websiteoptimization.com.

[7] The world in 2010: Ict facts and figures. http://www.itu.int/ITU-D/ict/material/FactsFigures2010.pdf.

[8] Yahoo one search. http://mobile.yahoo.com/onesearch.

[9] A. Badam, K. Park, V. Pai, and L. Peterson. Hashcache: Cache storage for the next billion. In *Proceedings of the Sixth USENIX symposium on Networked Systems Design and Implementation*. USENIX Association, 2009.

[10] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 1999.

[11] E. Brewer, M. Demmer, M. Ho, R. Honicky, J. Pal, M. Plauche, and S. Surana. The challenges of technology research for developing regions. *Pervasive Computing, IEEE*, 5(2):15–23, 2006.

[12] J. Charzinski. Traffic Properties, Client Side Cachability and CDN Usage of Popular Web Sites. *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, pages 136–150, 2010.

[13] J. Chen, S. Amershi, A. Dhananjay, and L. Subramanian. Comparing web interaction models in developing regions. *Proceedings of the First ACM Symposium on Computing for Development*, 2010.

[14] J. Chen, D. Hutchful, W. Thies, and L. Subramanian. Analyzing and accelerating web access in a school in peri-urban india. *Proceedings of the 20th International Conference companion on World Wide Web*, 2011.

[15] J. Chen, J. Iyengar, L. Subramanian, and B. Ford. Tcp behavior in sub-packet regimes. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, 2011.

[16] J. Chen, T. Karthik, and L. Subramanian. Contextual information portals. *Proceedings of AAAI Spring Symposium*, 2010.

[17] J. Chen, R. Power, L. Subramanian, and J. Ledlie. Design and implementation of contextual information portals. *Proceedings of the 20th International Conference companion on World Wide Web*, 2011.

[18] J. Chen, L. Subramanian, and E. Brewer. Sms-based web search for low-end mobile devices. *Proceedings of the International Conference on Mobile Computing and Networking*, 2010.

[19] J. Chen, L. Subramanian, and J. Li. Ruralcafe: web search in the rural developing world. *Proceedings of the 18th International Conference on World Wide Web*, 2009.

[20] Cluto. http://www.cluto.com.

[21] E. Cohen, B. Krishnamurthy, and J. Rexford. Improving end-to-end performance of the web using server volumes and proxy filters. *ACM SIGCOMM Computer Communication Review*, 28(4):241–253, 1998.

[22] doc2mat utility. http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download.

[23] J. Domčnech, A. Pont, J. Sahuquillo, and J. Gil. A user-focused evaluation of web prefetching algorithms. *Computer Communications Review*, 30(10):2213–2224, 2007.

[24] L. Fan, P. Cao, W. Lin, and Q. Jacobson. Web prefetching between low-bandwidth clients and proxies: potential and performance. *Proceedings of the ACM SIGMETRICS International Conference on Measurement and modeling of computer systems*, 1999.

[25] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: Hypertext transfer protocol–HTTP/1.1, June 1999. *Status: Standards Track*.

[26] K. Ganesan and C. Zhai. Opinion-based entity ranking. *Information Retrieval*, 15(2):116–150, 2012.

[27] A. Ghodsi, T. Koponen, B. Raghavan, S. Shenker, A. Singla, and J. Wilcox. Information-centric networking. In *Proceedings of HotNets*, 2011.

[28] HTMLAgilityPack. http://htmlagilitypack.codeplex.com/.

[29] HTTP Archive Format Specification. https://dvcs.w3.org/hg/webperf/raw-file/tip/specs/HAR/Overview.html.

[30] S. Ihm and V. Pai. Towards understanding modern web traffic. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 295–312. ACM, 2011.

[31] S. Ihm, K. Park, and V. Pai. Wide-area network acceleration for the developing world. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, 2010.

[32] S. Isaacman and M. Martonosi. Potential for collaborative caching and prefetching in largely-disconnected villages. *Proceedings of the ACM Workshop on Wireless Networks and Systems for Developing Regions*, 2008.

[33] S. Isaacman and M. Martonosi. The C-LINK System for Collaborative Web Usage: A Real-World Deployment in Rural Nicaragua. *Proceedings of the ACM Workshop on Networked Systems for Developing Regions*, 2008.

[34] S. Isaacman and M. Martonosi. Low Infrastructure Methods to Improve Internet Access for Mobile Users in Emerging Regions. *Proceedings of the 20th International Conference on World Wide Web*, 2011.

[35] J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Transactions on Computer Systems (TOCS)*, 10(1):3–25, 1992.

[36] R. Lancellotti, B. Ciciani, and M. Colajanni. A scalable architecture for cooperative web caching. *Web Engineering and Peer-to-Peer Computing*, pages 29–41, 2002.

[37] L. Li and J. Chen. Trotro: Web browsing and user interfaces in rural ghana. In *Proceedings of ICTD*, 2013.

[38] S. Mubaraq, J. Hwang, D. Filippini, R. Moazzami, L. Subramanian, and T. Du. Economic analysis of networking technologies for rural developing regions. *Workshop on Internet Economics*, 2005.

[39] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. A data mining algorithm for generalized web prefetching. *IEEE Transactions on Knowledge and Data Engineering*, 15(5):1155–1169, 2003.

[40] T. Palpanas and A. Mendelzon. Web prefetching using partial match prediction. In *Proceedings of WCW*, 1999.

[41] C. Patel, K. Supekar, Y. Lee, and E. Park. Ontokhoj: a semantic web portal for ontology searching, ranking and classification. In *Proceedings of the 5th ACM international workshop on Web information and data management*, 2003.

[42] R. Patra, S. Nedevschi, S. Surana, A. Sheth, L. Subramanian, and E. Brewer. Wildnet: Design and implementation of high performance wifi based long distance networks. In *Proceedings of the Fifth USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2007.

[43] A. Ratan, S. Satpathy, L. Zia, K. Toyama, S. Blagsvedt, U. Pawar, T. Subramaniam, and A. Ratan. Kelsa+: Digital Literacy for Low-Income Office Workers. *Proceedings of International Conference on Information and Communication Technologies and Development*, 2009.

[44] S. Rhea, K. Liang, and E. Brewer. Value-based web caching. *Proceedings of the 12th International Conference on World Wide Web*, 2003.

[45] S. Surana, R. Patra, S. Nedevschi, M. Ramos, L. Subramanian, Y. Ben-David, and E. Brewer. Beyond pilots: keeping rural wireless networks alive. In *Proceedings of the Fifth USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2008.

[46] W. Thies, J. Prevost, T. Mahtab, G. Cuevas, S. Shakhshir, A. Artola, B. Vo, Y. Litvak, S. Chan, S. Henderson, et al. Searching the world Wide Web in low-connectivity communities. *Proceedings of the 11th International Conference on World Wide Web*, 2002.

[47] J. Wang. A survey of web caching schemes for the internet. *ACM SIGCOMM Computer Communication Review*, 29(5):36–46, 1999.