Towards Minimizing the Required Bandwidth for Mobile Web Browsing

Madhuvanthi Jayakumar, Marcela Melara, and Nayden Nedev
Princeton University
{*jmadhu, melara, nnedev*}*@cs.princeton.edu*

***Abstract*—The number of smartphones has been increasing at a very fast pace over the last several years. At this rate, it is quite likely that mobile web traffic will take a serious portion of the global Internet traffic in the very near future. However, the improvement of mobile web access technology has been known to lag behind the current growth: we face issues such as limited bandwidth and computational power, and high latency. Additionally, users have financial incentives to use their limited data plan effectively. These facts motivate new ways of designing and building mobile web access technologies and networks. Specifically, new methods for reducing mobile web traffic and for utilizing the mobile computational and network resources more efficiently should be found.**

In this paper, we propose a novel technique for reducing the needed bandwidth for mobile web browsing. It automatically identifies redundancies in mobile website data to reduce the number of transmitted bytes. We leverage the standard deduplication techniques of partitioning website data into chunks and fingerprinting them for redundancy detection and efficient retrieval from a cache. The only additions to the current mobile browsing framework that our method requires is a dedicated proxy server and the installation of a mobile proxy-client. We implement our techniques in a sophisticated simulator and evaluate their effectiveness on a number of popular websites. Our empirical results show that, with an appropriate tuning of all system components, the needed bandwidth can be consistently reduced to less than 20% in most cases.

I. Introduction

As of the beginning of 2013, global mobile traffic represented roughly 13% of Internet traffic. This number was just 1% in 2009 and increased to 4% in 2010. At present, of the 5 billion mobile phones in the world, only a fifth are smartphones. The user base of smartphones is expected to expand by about 42% a year, and with that grows the mobile web traffic [8]. The issue we face in North America is that mobile networks are already running at 80% of capacity and 36% of base stations are facing capacity constraints [1]. Globally, the ubiquitous appearance of mobile devices with the rise of cheap smartphones and tablets in developing countries in Africa and India is creating a demand for available and affordable bandwidth as well. In addition to computational barriers, the data limits posed on mobile carrier data plans and data overage charges are an incentive for users to utilize their available data effectively.

The growth in demand of mobile network bandwidth in conjunction with the financial incentives of smartphone users motivates new and innovative techniques to reduce mobile network traffic, and to use mobile bandwidth more efficiently. Prior work has found that one possible solution to the bandwidth constraint problem is to identify and reduce data redundancy in website data [15]. However, this work has mainly been in the area of desktop browsing, while data redundancy in content viewed through a mobile browsers still remains widely unexplored.

We present a technique which leverages these data redundancies to improve the bandwidth utilization efficiency of mobile browsers. We use the standard data deduplication techniques of chunking [5], [15], [6] and Rabin fingerprinting [11] to find the content a requesting mobile client actually needs, avoiding the transfer of redundant data. By focusing purely on redundancies in web page content, our mechanism helps reduce the number of bytes sent across the network, thereby reducing the required bandwidth. Our technique allows us to analyze the redundancies across and within websites and to calculate the amount of potential bandwidth savings that can be obtained through data deduplication. This paper makes the following major contributions:

- Design of a system and a protocol for effective reduction of required bandwidth for mobile browsing.
- Complete implementation of two simulators - one for experimental purposes and another that serves as a proof-of-concept prototype.
- Thorough experimental evaluation of our prototype system on a large number of different criteria, including various data chunk sizes, different cache eviction scheme algorighms and transferred data metrics.

The rest of this paper is organized as follows. We discuss the previous work done on the problem in Section II. We describe the basic design of the system that we built in Section III. Details about our implementation are presented in Section IV and results of experimental evaluation in Section V. We discuss the results of the evaluation that we performed in Section VI, give some directions for potential future work and conclude in Section VII.

II. Previous Work

Two notable works reduce the required bandwidth in different contexts. They both introduce similar concepts that have inspired our work. Ihm *et al.* [4] design an efficient WAN accelerator for developing countries by leveraging chunking, Rabin fingerprinting and caching. The authors propose efficient chunking and caching schemes to handle

the challenges caused by the equipment in developing countries such as limited RAM and poor hard disk performance. Muthitacharoen *et al.* [6] present a network file system designed for low-bandwidth networks (LBFS). This system leverages file similarities using sliding window chunking and Rabin fingerprinting. They experimentally show that this approach uses an order of magnitude lower bandwidth - a fact that make this approach promising for the case of mobile Web browsing.

A different line of works measure the amount of redundancy in today's web traffic and to identify its cause by analyzing web traffic logs and datasets. Ihm and Pai [3] perform such an analysis with data collected over a five-year period. Along with proposing a new page analysis algorithm that is better suited for the modern web, the authors find many interesting facts about connection speed of today's web users, NAT usage, traffic content type and share of different types of web sites. They also investigate the amount of redundancy in the collected data and the impact of caching. More specifically, they compare content-based with object-based caching approaches and quantify the major sources of redundancy. Qian *et al.* [10] measure the amount of redundant data transfers caused by inefficient web caching particularly for smartphones. They conclude that redundant transfers contribute up to 20% of the total HTTP traffic on two different datasets. Moreover, the reason for this redundancy is usually developers' faults in cache implementations, *e.g.* by not fully supporting and following the used network protocol or by not fully utilizing the caching support provided by the libraries. The authors also explore the impact of other factors such as limited cache size and caches whose data does not survive a process restart or a device reboot.

Lastly, the standard deduplication techniques used in our work were first developed quite a long time ago. The idea of chunking files to find similarities between them was originally presented by Manber in 1994 [5]. The notion of "fingerprinting" files dates back to 1981 when Rabin first presented a novel hashing technique that uses random polynomials [11].

III. System Design

At the core of our technique lies the ad hoc system and protocol we have designed to integrate into the existing mobile browsing framework in place today. This system is comprised of a specially configured proxy server, a per-device mobile proxy-client, each of which manages specific tasks in our deduplication mechanism. Our protocol specifies the data exchanged between these two components. We integrate our system into the data processing phase of browsing, *i.e.* the phase between the initial request for a page and the rendering of the requested page.

In our system, the proxy server and mobile proxy-clients are logically located between the mobile browser and the Web, managing requests for web pages coming from the browser and engaging in the bandwidth reduction protocol. Most of the computation for our deduplication mechanism occurs on the proxy server so as to minimize the additional strain on the limited computational resources of mobile devices, while the mobile proxy-client supplies the proxy server with relevant information it needs to find data redundancies.

For our deduplication mechanism, we use several standard techniques useful for identifying similarities between files and data streams. In particular, we use the following two:

1) Chunking: Partition incoming web content into data chunks.
2) Fingerprinting: Create a unique encoding for each unique data chunk.

We explore two chunking techniques and use the implementation for Rabin fingerprinting given by Broder [2].

In addition, our mechanism requires special caches, one on the proxy server, and one on the mobile proxy client to store unique chunks of data. The client cache replaces the built-in browser web cache. We assume multiple mobile devices connecting to the proxy server and making distinct, un-correlated requests such that our mechanism does not attempt to synchronize the proxy server and client caches.

A. Chunking

1) Fix-Sized Chunking: The first way of partitioning the data that we explore use fix-sized chunks. If the size of the chunk is set to be n bytes, then the first chunk is produced by taking the first n bytes from the stream. The second is constructed from the second portion of n bytes and so on, until the end of the stream is reached. If there is not a sufficient number of bytes to construct the last chunk, it is padded.

2) Sliding Window Chunking: This is a technique originally presented by Manber in order to find similarities between different files [5]. As opposed to partitioning files and streams of data into fix-sized chunks, this method allows us to adapt chunk sizes based on the actual contents of the file or packet stream. This is done by choosing a fix-sized window which we slide across the entire contents of a web page, and fingerprinting each region until some number of low-order bits of the fingerprint are all 0. Once this occurs, we have found a breakpoint and the chunk boundary is set to the end of this special window.

We compute our sliding window chunks based on the equations and parameters presented in [15] and combine them with the enhancements used in the LBFS content-based breakpoint chunking scheme [6]. Thus, not only do we specify a window size β and breakpoint fingerprint value with γ zeros in the low-order bits, but we also specify a minimum and maximum chunk size.

The purpose of choosing this scheme over the fix-sized chunking scheme is rather straight-forward. Since the chunks

are chosen based on content rather than on position in the web page, minor changes will not affect surrounding chunks. In contrast, any minor change to a web page will probably shift a large number fix-sized chunks by some amount causing changes to a large number of fingerprints, which in turn leads to an overall decrease in redundancy detection. Unlike Manber, and Spring and Wetherall, who use this chunking scheme to fingerprint every possible region in a file or packet and then choose a specific subset of these fingerprints to find redundant data [5], [15], we use the same approach as LBFS, *i.e.* to use these scheme merely for finer-grained chunking of the data and determine redundancies using additional hashing or fingerprinting once the chunks have been found.

B. Data Fingerprinting

Data fingerprinting is the second deduplication technique we use in our bandwidth reduction mechanism. In general, a fingerprinting function is similar to a hash function in that it is also a one-way function which maps an arbitrarily large input to a fixed-size number, and each unique input has a unique fingerprint value. However, unlike an conventional hash function, this fingerprinting function can be decomposed for the incremental computation of a fingerprint. This is done by representing the input as a polynomial modulo a pre-determined irreducible polynomial [11], [6].

Our use of fingerprinting is two-fold: (1) To generate the sliding window chunks, and (2) to generate fingerprints as identifiers for unique chunks. We exploit the fact that any size chunk maps to a fixed-size fingerprint, which is much more efficient to transfer across the network than the entire data chunk. As we discuss in more detail in Section III-D, our mechanism requires the transmission of two rounds of fingerprints for data redundancy detection but the total bandwidth savings of our system outweigh these added overhead and bandwidth requirements of these two additional transmissions (see Section V).

C. Caching

Another main component of our system are two caches - one in the proxy server and one in the mobile client. Each of them maintains a mapping between fingerprints and the chunks that they represent. The fingerprint of each chunk is computed to retrieve the corresponding chunk from either one the caches. There are no specific restrictions to the internal representation of the caches and their size.

D. Bandwidth Reduction Protocol

The final major part of our technique is the reduction protocol between the mobile device and the proxy server. It brings together all the components described in the previous sections. Each time a new page is requested by the mobile browser, the following protocol is performed (Figure 1):

1) Mobile proxy-client sends an HTTP request to the proxy server.
2) Proxy server relays this request to the proper web server.
3) Proxy server performs chunking and fingerprinting of the chunks for all the received web content. It sends all the fingerprints to the mobile device.
4) Mobile proxy-client checks its cache for the fingerprints, and creates a list of those it needs. It sends this list to the proxy server.
5) Proxy server creates a list of the needed chunks according to the received needed fingerprints. It sends this list back to the mobile device.
6) Mobile proxy-client reconstructs the entire requested page from its cache contents and the received list of needed chunks.

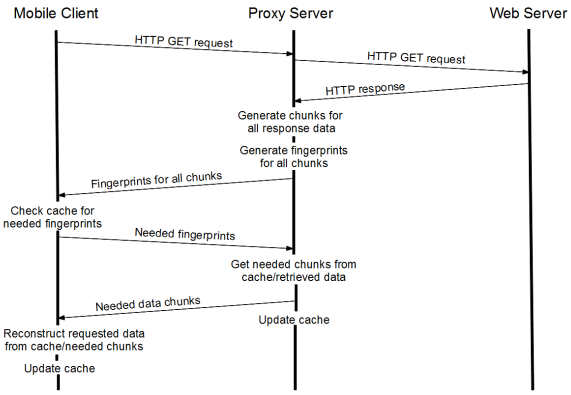

Figure 1. The Bandwidth Reduction Protocol as Part of the Whole System.

Our protocol uses standard HTTP GET requests and does not require changes to web server configurations. It also does not alter the mobile browser front-end and simply requires the mobile proxy-client as the interface to the proxy server, leaving our technique transparent to users. Furthermore, our protocol does not aim to synchronize the mobile proxy-client caches with the proxy server cache because we do not make any assumptions about similarity of requests coming from different users' mobile devices.

IV. Simulator Implementation

We implemented two versions of a simulator of our system:

1) An offline simulator, which uses data collected and stored during a mobile browsing session, and input into the simulator offline, and
2) A networked simulator, which simulates a basic incarnation of our system in real-time.

Both simulators are written in Java, and use five helper interfaces and classes each one representing a component of the system, in addition to the proxy server and mobile device classes. In particular, we use two helper interfaces: `ICache` and `IProcessor`, which allow for different implementations of caches supporting various eviction algorithms, and of what we call cache processors, respectively. A cache processor is an entity which acts as an interface between a device and its web cache, with its most important task to process incoming web content based on the device's cache contents. While both of our simulators use a simple implementation of `IProcessor` called `SimpleProcessor`, which manages web content caching, and measures the cache hitrate and missrate, we have multiple implementations of `ICache`, which we address later in this section.

The three helper functions we use are `Chunk`, `Chunking` and `Fingerprinting`. The `Chunk` class defines a fixed-size data chunk with a given size in number of bytes and the data. `Chunking` is the facility which generates all the data chunks for a given input, either an input file containing web page data or a data stream of online web data. The `Fingerprinting` class is a wrapper for the Java *rabinhash* library written by Sean Owen [13], and uses the `RabinHashFunction32` implementation of Rabin's fingerprinting method creating 32-bit fingerprints of a given data chunk [12].

Finally, we created the `ISimulator` interface to build different kinds of simulators. Our offline version uses one or more `Mobile` devices and a `ProxyServer` to implement the simulation of our reduction protocol described in Section III-D. The networked simulator uses the networked counterparts of these two components.

Figure 2 summarizes our implementation software in its entirety[1].

A. Offline Simulator

Our implementation of the offline simulator consists of an umbrella simulator (`SimulatorV1`) which instantiates a proxy server and a mobile device (`ProxyServer` and `Mobile`). Since the main goal of this simulator is to analyze mobile web content redundancy, we built it such that it accepts multiple input files containing stored web page data. The chunk size as well as proxy server and mobile cache sizes are parameters to the simulator.

The simulator implements our protocol by passing the appropriate values as variables between the two device instances via specific methods and functions. In order to simulate multiple rounds of our protocol, it then iterates over the specified set of files gathering the following three statistics: (1) The number of chunks (and hence fingerprints) processed, (2) The remaining cache capacity after processing a web page, and (3) The cache miss-rate for the last web page processed. We use these statistics to calculate the average mobile cache miss-rate for one series of protocol simulations, as well as the average number of bytes transmitted between the proxy server and the mobile client. We elaborate further on these calculations in Section V.

There is one detail about our offline simulator that is worth noting. After running tests with fixed-size chunks, we began an implementation of this simulator (`SimulatorV2`) using sliding window chunking in order to measure differences in number of bytes transferred and saved bandwidth. During our preliminary tests we found that our implementation still contains a few bugs and does not accurately perform sliding window chunking for all input web page data. Nevertheless, the results we obtained from the early experiments that came to a successful completion with this simulator, show promising bandwidth reduction rates indicating that this chunking method indeed offers further improvements to our bandwidth reduction system.

B. Networked Simulator

The networked simulator consists of the proxy server (`ProxyServerNet`) and the mobile client simulator (`SimulatorV3`), which is a wrapper for the networked mobile proxy-client (`MobileClientNet`) and is capable of performing several rounds of requests to the proxy server in real-time. It does so by prompting the user to manually enter the next web page URL she wishes to visit, simulating web browser interactions. The simulator architecture can be seen in Figure 3.

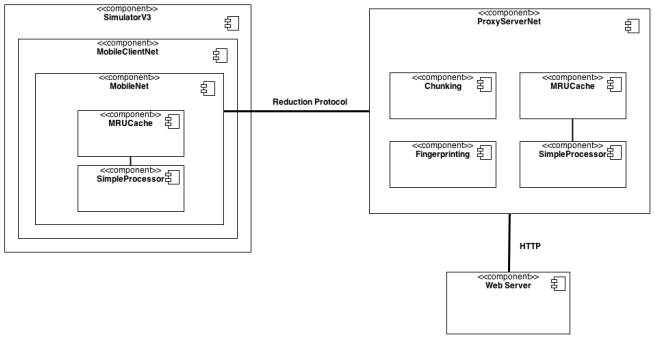

Figure 3. Networked Simulator Runtime Interactions.

In more detail, the networked simulator implements our reduction protocol as follows. First, the mobile proxy-client opens a socket to the proxy server, which is listening for connections at a user- specified port. The mobile proxy-client sends an HTTP request to the proxy server, which in turn sends it on to the hosting web server, including the User-Agent string of a mobile device[2] to ensure that it receives the mobile version of the requested web page. Once the proxy

[1]Blue lines denote aggregation with the cache and are merely for legibility purposes.

[2]We use the Samsung Galaxy SII as our model mobile device across all our implementations and experiments.

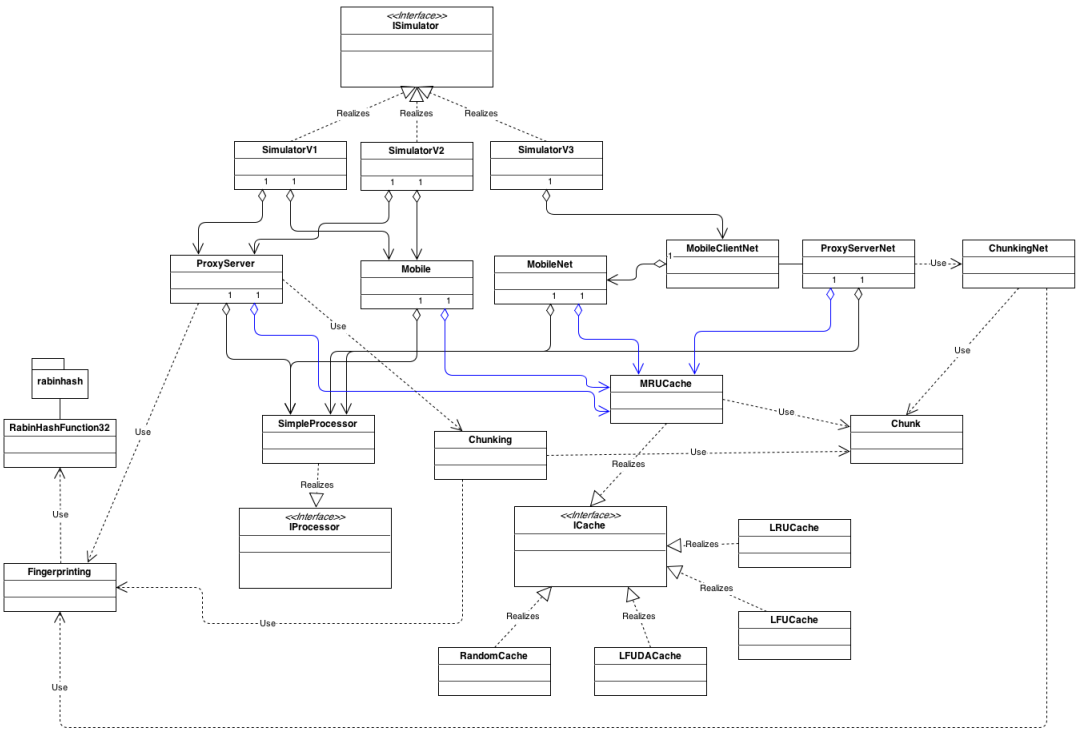

Figure 2. Implementation Software Class Hierarchy.

has received the response from the appropriate web server, it engages in our protocol[3] with the mobile proxy-client, exchanging the relevant information via the network. At the end, the mobile proxy-client reconstructs the content data of the received web page into an HTML file, which can then be viewed in any web browser. After the proxy has served the mobile client's request, the client is able to make a new request and repeat this process.

We found that, while simulating mobile browsing, for one not using an actual mobile phone, and for another not using a web browser program of any sort, is not realistic, our networked simulator is a good first proof-of-concept prototype showing that our reduction protocol is viable. In our experimental evaluation, we argue that it reduces the required bandwidth compared to the currently required mobile browsing bandwidth.

In the end, we would like to discuss some caveats of our networked simulator. First, as our mobile client simulator does not have the capabilities of a web browser and it receives preprocessed data from the proxy server, it cannot automatically handle HTTP responses indicating page redirects (i.e. server response code 301, for example). Thus, our proxy server handles HTTP responses with the server response codes 200 and $40X$[4] since the web server returns HTML content with these responses. An additional consequence of the fact that our mobile client is not browser-like is that it must create a local copy of each retrieved web page. Although the content of the web page is reconstructed correctly, since many links within web pages point to relative paths, the retrieved web pages are often rendered incorrectly in the local web browser as it cannot find cascading stylesheets (CSS) and embedded images on the local disk. We propose how to solve these issues in our discussion on future work.

C. Different Caching Algorithms

A substantial number of different web cache eviction schemes have been discussed in the literature [16]. Although most of our experiments are done with the MRU scheme, we have implemented several others:

- *Least Recently Used (LRU)*. This replacement policy could be viewed as the opposite of the *MRU* one. It removes the item that has been got or put in the earliest moment of time compared to all other items in the cache.
- *Uniformly Random*. This eviction scheme randomly picks an item to be replaced. All the items have

[3]Minor changes were made to the chunking facility as well as the mobile device definition to support networking.

[4]200 means granted normally, $40X$ response codes indicate a form of client-side error such as malformed requests but return HTML-format content with this information.

equal probability to be picked, *i.e.* they are uniformly distributed. It could be considered a scheme where a different distribution is used. However, it our case there is no clear distinction between the fingerprints, so we chose each one to be picked with equal probability.

- *Least Frequently Used (LFU)*. Here the item that has been got or put the smallest number of times compared to the all other items in cache is replaced.
- *LFU with dynamic aging (LFU-DA)* [9]. A well-known issue with the *LFU* scheme is the cache pollution problem. That is, when an item has accumulated a very high frequency count and then becomes unpopular, it remains for a huge amount of time in the cache before being removed. *LFU-DA* solves this problem by adding a constant to the frequency count of an item when it is accessed. This way, the recently popular items have a bigger frequency count that the older ones. It prevents, previously popular items to pollute the cache and accelerates their removal.

V. Experimental Evaluation

A. Obtainining Web Data

We obtained results through a process of collecting offline data, and modifying our simulator to output information about the data being processed. Mainly, we observed how changes in parameters affected the miss rate as well as the number of bytes transferred between the proxy and mobile device.

In order to run our experiments, we first collected offline data. Over the course of four days, we issued telnet GET requests to various webpages (both desktop and mobile versions) in the morning, afternoon and evening. The frequency with which we made these GET requests were for the purpose of reflecting browsing patterns, and it would give us information about the change in the content of a webpage over the course of a day and over the course of multiple days. We stored each response in a different file and then processed the data to obtain the byte stream version of the html pages. Using this byte stream, we ran several experiments that gave us insight into data redundancy within webpages.

B. Mobile vs. Desktop Browser Content

Figure 6 shows the distinctions between mobile web content and desktop web content. Many web servers today structure their webpages differently depending on the user-agent they're serving to increase the speed with which the webpages load, to provide better service with respect to UI and various other reasons. Therefore, mobile pages are inherently different from desktop browsers and thereby require its own analysis. Figure 6 shows that the mobile version of cnn.com is only about a fifth of the size of the desktop version. The bytes transferred for the unchunked protocol shows that the size of the webpage remains relatively constant, and that the entire webpage has to be reloaded from the server for each request since the content is no longer "fresh".

The bytes tranferred with the chunked protocol shows that the amount of redundancy that is eliminated in both mobile and desktop websites is proportional to the size of the web page. It also provides insight into exactly where our protocol performs well, and where the overhead of the protocol takes away from the benefits achieved from chunking. We see that on the first visit, the amount of bytes that needs to be transferred is almost twice the size of the actual content. This inefficiency comes from the fact that we're using chunk size of ten bytes. During the first visit to cnn.com, when there is no base copy of the webpage, the fingerprints representing the entire webpage need to be sent back and forth creating an inefficiency. However, once there is a base copy in the cache, the overhead decreases substantially. We can see from the graph that by the 12th visit, we are only transferring half the number of bytes as we would need to reload the entire webpage.

C. Effects of Chunk Size

The use of chunk size of 10 bytes means that each redundant chunk saves 6 bytes because of the 4 bytes of fingerprint needed to represent that chunk. This led us to explore different chunksizes to find the ideal chunk size that takes into consideration the tradeoff between having a low cache miss rate and having a fingerprint map to a bigger chunk. Figure 7 shows the relationship between % of web content that is needed (based on cache miss rate) and chunk size based on a series of visits to cnn.com. We obtained the data through visits to cnn once a day for four days. Day 1 is not shown since the cache is empty so the contents of the entire webpage needs to be transferred. The first visit is not shown since the cache is empty and so 100% of the content needs to be transferred for all chunk sizes. It is clear from this graph that if we use smaller chunk sizes, the percent of content that needs to be sent decreases. The steeper line for chunk 5 when compared to chunk 45 shows that as the number of visits increase, the overlap of smaller chunk sizes increases faster. However, it means that each fingerprint maps to a smaller chunk and so more fingerprints are needed to represent the small amount of data that needs to be transferred and fewer fingerprints are needed to represent a large amount of data.

Figure 8 takes into account the effects of the extra bytes that need to be transferred to account for the fingerprints that needs to be transferred to represent redndant chunks. In this graph we can see that as the chunk size increased, the miss rate also increased as expected, but the bytes transferred actually decreased. This is because if the chunk size is small it gets expensive for the mobile device to communicate which chunks it needs. At this point, the ideal chunk size

depends on the size of content that needs to be transferred as opposed to percentage.

D. Bandwidth Reduction while Web Browsing.

The next two graphs show what happens when we visisted three websites three times a day for four days to simulate ”mobile browsing”. The data was gathered by visiting cnn.com, nytimes.com and economist.com in an alternating basis three times a day over four days. This graph shows that if the ’base’ content of each webpage is in the cache, then less than 20% of the content is generally new. The first three requests show that there is some, but not a lot of redundancy between webpages. Figure 9 shows that on the the first visit, the cache is empty and 100% of the traffic needs to be transferred. For the second website, almost all of it needs to be transferred (90%) because of lack of overlap with the first website. On the third, the proportion decreases further but a majority of the page still needs to be transferred. After this point, we have the base page for all three websites in our cache and only the differences need to be transferred from the proxy, so the proportion of content that needs to be transferred stays below 20% by the sixth url request. Figure 9 calculates the proportion of content that needs to be transferred based on the cache miss rate but does not take into account the additional bytes that have to be transferred due to fingerprints.

Figure 10 takes this into account and shows the total number of bytes that were transferred for the 32 requests. This shows the bandwidth savings obtained from chunking. We assume that no response is identical to a previous response. This means that without chunking, the full webpage has to be reloaded for each request, leading to the linearly increasing number of bytes we see in the graph. With chunking however, we see that past the first few requests in which the effects of the overhead are heavy, the number of total bytes transferred rises gradually, and the gap between the bytes transferred grows with the number of requests.

E. Effect of Different Eviction Schemes

We evaluated the missrate of all cache eviction algorithm schemes described in Section IV-C. We simulated mobile web browsing of three webpages, opening them three times a day in four consecutive days. Then we measured the missrate the the mobile cache has. The results are shown on Figure 9.

The first thing that we can notice is the poor performance of the random cache eviction scheme. It does not achieve good results because it “blindly” evicts items from the cache without any consideration of their importance. Probably it could be improved by using a non-uniform distribution for the probability with which each items is evicted. However, we have not found any differentiating factor between the chunks that can serve as a basis of such distribution.

LFU and LFU-DA scheme does not perform best in our study. We explain this with the fact that recency is more important than frequency. Since the data from the most recent visit to a webpage is needed when requesting information from the cache, it is not so relevant how frequent this information is in all the visited websites. Another thing that can be notices is that LFU-DA scheme performs even worse than ordinary LFU. The reason for this is that it is unlikely that cache pollution appears in a relatively small number of accesses, as in this experiment.We also performed the same experiment with LFU-DA cache with different values of the aging parameter (*e.g.* 2, 10, 50, 250). However, we omitted the results of these experiments from the plot because they are almost the same as for 5 (which is shown). We belief that this eviction scheme will be useful for much longer sequence of web requests where cache pollution actually occurs. It can be seen that it has very good performance at the end of the shown curve. Likely, it will have better performance for longer sequences.

As it can also be seen from the figure, the MRU scheme achieves the lowest missrate. We explain this with the fact that the cache size is small and when a new website is visited the data from the previous has been evicted. This in turn improves the missrate for the new website and the overall shown on the graph. The same motivation applies for the LRU scheme. LRU is actually the opposite of MRU and it is impossible both of them to perform well.

Another factor that is in favor of the MRU scheme is implementation complexity and speed. It is one of the easiest to implement. It does not have to maintain information in efficient data structures as the LRU or LFU one, for example. It is also quite hard to achieve the same speed for LRU, LFU or LFU-DA as for the MRU.

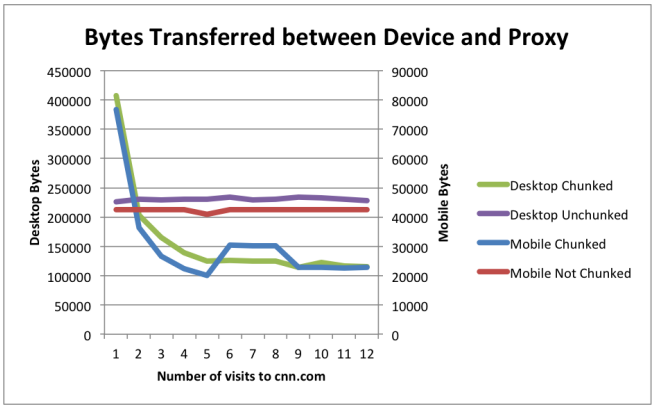

Figure 4. Desktop vs Mobile Browser page differences.

VI. Discussion

The process of getting our results provided insight into the redundancies that exist within mobile web pages. We first observed that pages that are designed for mobile browsers are only a fraction of the size of pages designed for desktop browsers. In addition to size, we also know that the way

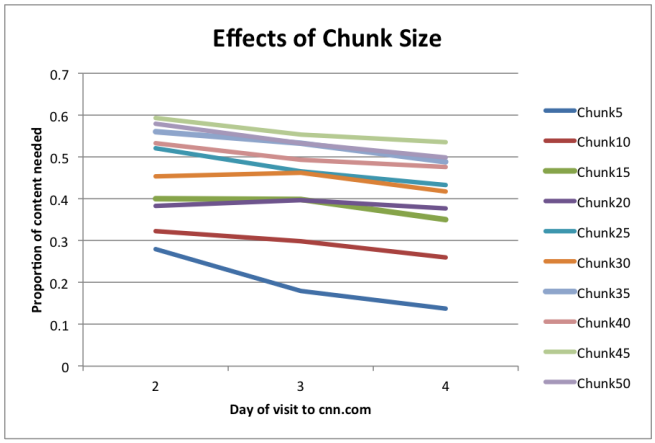

Figure 5. Effects of Chunk Size on portion of content needed

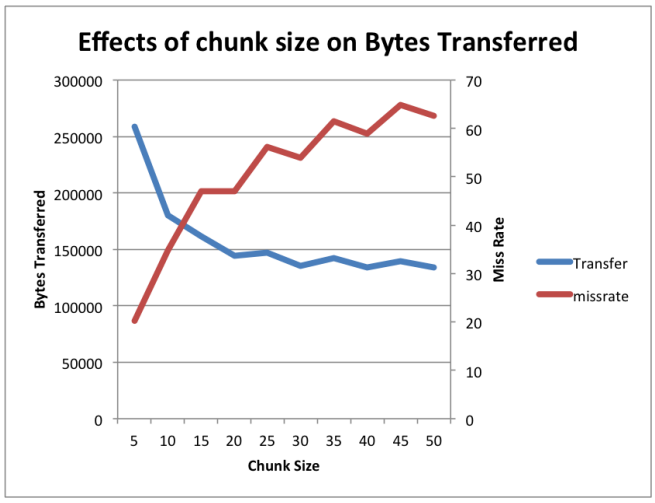

Figure 6. Effects of chunk size on Bytes Transferred. .

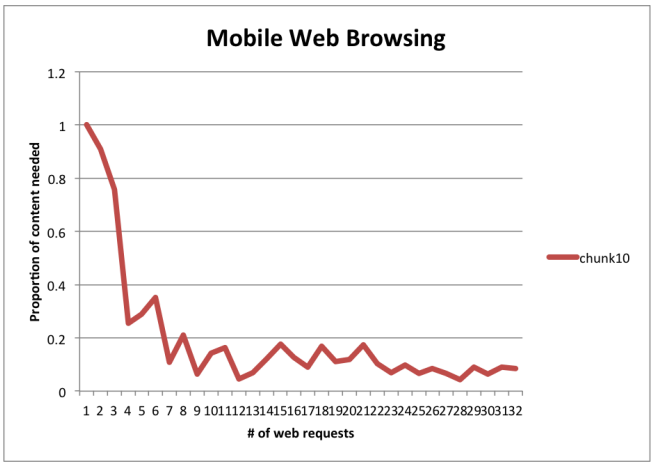

Figure 7. Mobile Web Browsing.

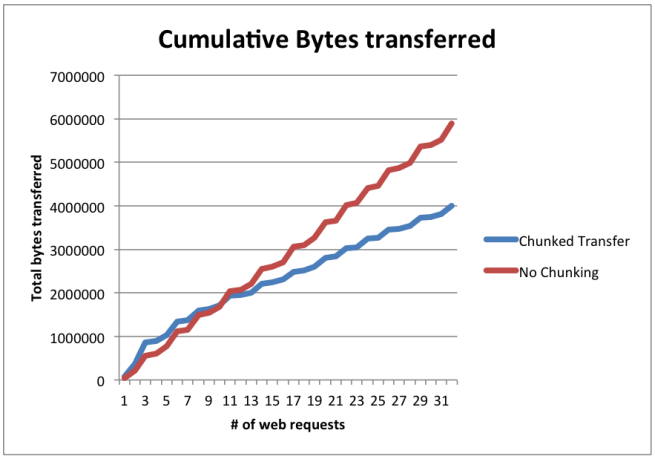

Figure 8. Cumulative bytes transferred during browsing.

that mobile sites are structured are inherently different through observation. This shows that it is worth exploring the types of redundancies that exist within mobile sites. This information can be used in the future to examine how these redundancies are similar to or different from desktop browser site redundancies. It can also be used to design caches that will take advantage of the redundancy patterns that we find.

As for chunk size, we found that a chunk size close to 10 bytes was optimal in the trade-off between obtaining a low miss-rate and reducing the number of bytes of fingerprints transferred back and forth between the proxy and mobile device. Based on the feedback we received after our presentation, we also implemented the sliding window chunking methodology so that we wouldn't be constrained to a fixed chunk size and so that we would find a larger percent of deduplicated data.

Lastly, we simulated patterns of mobile web browsing to obtain an approximate result of how much we can expect to reduce bandwidth. We found that if the cache contains an older version of the page, and data from several similar pages, then (with a chunk size of 10 bytes) only about 20% of the content is new and needs to be retreived from the proxy.

There were multiple design decisions that we considered before constructing the protocol. We designed our protocol such that fingerprint computation only occurs at the proxy. This is because mobile devices tend to have limited computational capacity and if the heavy lifting is deferred to the proxy, then the mobile device simply has to do look-ups to determine cache hits and misses. However, the drawback in this approach is that we incur the overhead of passing fingerprints across the network, leading to possible latency and additional power consumption. In addition, it also reduces the benefits of bandwidth reductions obtained by chunking.

We build a secondary protocol implementation to address the following few additional concerns. First, There is an

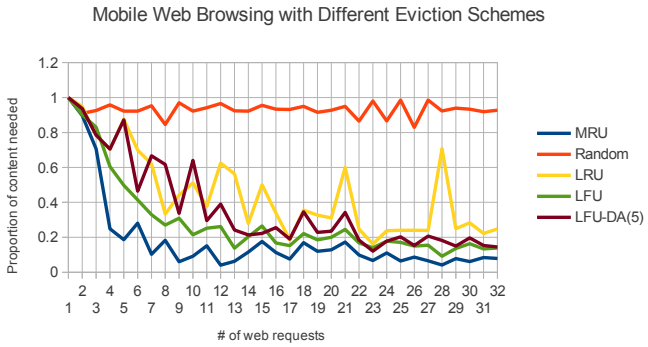

Figure 9. Missrate of Different Caches

inefficiency that comes from the additional communication that occurs between the device and proxy to send fingerprints. Second, using the MRU eviction algorithm for our tests could have possibly restricted by how much we can reduce bandwidth. To consider tradeoffs in communication protocols and eviction schemes, we implemented the protocol with a url-cache as proof of concept and for testing purposes. The higher level approach is shown in Figures 12 and 13.

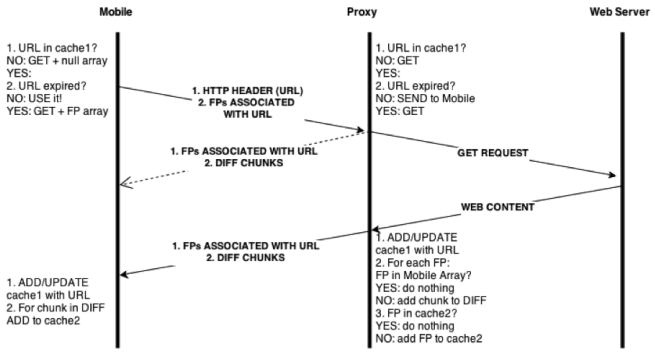

Figure 10. Protocol.

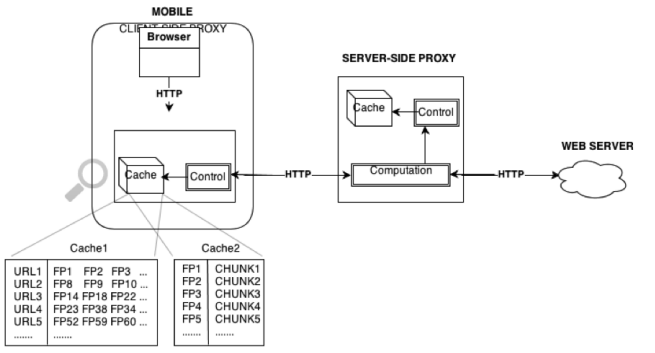

Figure 11. Url-Cache.

The modified implementation works in the following way. In addition to the previous implementation we added a url-cache, shown in Figure 13, which maps each url request to the set of fingerprints that represent it. If the mobile device contains an older version, it sends the list of fingerprints to the proxy along with the requested url. The proxy then sends back the diff between the mobile's fingerprints and fingerprints of fresh content which it either obtains from its own cache or from the web server.

Figure 12 shows that it may be possible to reduce latency with this approach due to the decreased amount of communication that needs to occur between the proxy and device. In addition, preliminary testing during implementation showed that in some cases (dependent on the size of the page, whether or not an older version of it is in the cache, and pattern of 'browsing') bandwidth was reduced. The fingerprint overhead remains on the same order because the fingerprint of the entire page still needs to be sent. The bandwidth reduction for some cases came from a lower missrate afforded by the url-cache which was used for the eviction scheme. The Least Recently Added url was evicted from the url cache and the fingerprints that the url mapped to were evicted from the other cache as long as the fingerprint didn't overlap with the other webpages.

We also made several assumptions while implementing the protocol and testing. We assume that each time we made a request that the content in our cache was not fresh, and we did not implement a check for expiration-date or use the 'If-Modified-Since' protocol to check for freshness. We also attempted to collect data from various other websites but ran into several issues. Some websites don't have a different version for mobile devices at all. Some others had a different version, but did not use it as its response based on the User-Agent, and instead specifically required a request of the mobile url. Redirects were also not handled in the way we obtained offline data. Furthermore, we did not check to see if the data was "cacheable" and assumed that all the websites we were working with had cacheable content. Lastly, the results of our tests did not incorporate checks for content overlap between different chunks. We do however, now have an implementation of a sliding window for chunking and fingerprinting in place which is passed along the content stream to catch redundancies between different chunks that would not have been caught with fixed-sized chunks.

A last, but minor point is the consideration of how integrity of data is affected by possible collision due to fingerprinting. We assume that we do not get collision because the size of our content is small enough that the risk of collision is negligible.

We have thoroughly tested our implementation in various settings and with a wide range of parameter values. We have evaluated the results of chunking by using data from non-chunked protocol as control. We've done preliminary analysis on using mobile web pages by collecting offline data for desktop browsers and using it as a control. We have

experimented with various chunk sizes as well as different caching algorithms. We have evaluated the effectiveness of our protocol both in terms of miss-rate as well as how that translates to the actual number of bytes transferred which gives us better insight into the amount of bandwidth that can be saved.

VII. Conclusion and Future Work

Our results show that our simulated system and protocol significantly reduce bandwidth requirements of mobile browsing. The next step would be an actual implementation of our mobile proxy-client to make it compatible with commodity smartphones, as well as to modify a mobile browser back-end to integrate our proxy-client. On the server side, we would like to use a publicly available server for our proxy server so that we can test our technique outside of our departmental boundaries. With an actual implementation in place, it would be possible to make actual latency and computational overhead measurements for our system.

Other enhancements to our current model include using one of our better cache implementations for both the proxy server and mobile proxy-client caches, and allowing the proxy server to support all kinds of HTTP response codes to fully integrate it into the browsing framework. Moreover, we would like to study how the addition of a third deduplication technique, namely delta encoding as used in [14], would affect our bandwidth reduction rates.

Another important area that needs more exploration is data redundancy due to HTTP traffic from mobile apps. Smartphone users are increasingly accessing specific websites and web services via dedicated mobile apps such that the use of mobile browsers is slowly declining. According to the app analytics firm Flurry, the average smartphone user spent 94 minutes using mobile applications and only 72 minutes browsing the web by the end of 2011 [7]. Thus, an important future direction motivated by our project is to study mobile app data redundancy and design a similar system to our bandwidth reduction technique for mobile app traffic.

In conclusion, we have developed a technique that leverages data redundancy in mobile web pages viewed through mobile web browsers to more efficiently make use of the limited mobile bandwidth. We built a simulator to analyze these data redundancies and found that our system reduces the number of bytes transmitted to a mobile device to a consistent 20% of the total in most cases by only sending uncached data. Furthermore, we have extensively studied various cache eviction algorithms in the context of our system with the hope that our insights will inform future work on mobile bandwidth reduction. Lastly, we have implemented a proof-of-concept networked mobile client simulator and dedicated proxy server showing that our technique does not require changes to web server configurations, and would not alter the mobile browsing experience, making our technique a viable enhancement to mobile browsers benefiting mobile users.

References

[1] H. Baldwin. Wireless bandwidth: Are we running out of room? *ComputerWorld*, pages 1–4.

[2] A. Z. Broder. Some applications of rabin's fingerprinting method. In *Sequences II: Methods in Communications, Security, and Computer Science*, pages 143–152. Springer-Verlag, 1993.

[3] S. Ihm and V. S. Pai. Towards understanding modern web traffic. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '11, pages 143–144, New York, NY, USA, 2011. ACM.

[4] S. Ihm, K. Park, and V. S. Pai. Wide-area network acceleration for the developing world. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, pages 18–18, Berkeley, CA, USA, 2010. USENIX Association.

[5] U. Manber. Finding similar files in a large file system. In *Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference*, WTEC'94, pages 2–2, Berkeley, CA, USA, 1994. USENIX Association.

[6] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, pages 174–187, New York, NY, USA, 2001. ACM.

[7] Newark-French, Charles. Mobile App Usage Further Dominates Web, Spurred by Facebook. In *Flurry Blog*, Jan 2012.

[8] P. Olson. 5 eye-opening stats that show the world is going mobile. *Forbes*, Dec. 2003.

[9] S. Podlipnig and L. Böszörmenyi. A survey of web cache replacement strategies. *ACM Comput. Surv.*, 35(4):374–398, Dec. 2003.

[10] F. Qian, K. S. Quah, J. Huang, J. Erman, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Web caching on smartphones: ideal vs. reality. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12, pages 127–140, New York, NY, USA, 2012. ACM.

[11] M. O. Rabin. Fingerprinting by Random Polynomials. Technical report, Center for Research in Computing Technology, Harvard University., 1981.

[12] Sean Owen. Package com.planetj.math.rabinhash. http://rabinhash.sourceforge.net.

[13] Sean Owen. Rabin Hash Function. http://sourceforge.net/projects/rabinhash.

[14] P. Shilane, M. Huang, G. Wallace, and W. Hsu. Wan optimized replication of backup datasets using stream-informed delta compression. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, FAST'12, pages 5–5, Berkeley, CA, USA, 2012. USENIX Association.

[15] N. T. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '00, pages 87–95, New York, NY, USA, 2000. ACM.

[16] K.-Y. Wong. Web cache replacement policies: a pragmatic approach. *Netwrk. Mag. of Global Internetwkg.*, 20(1):28–34, Jan. 2006.