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*—**

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 moved up to 4% in 2010. [**?**]. At present, of the 5 billion mobile phones in the world, only a fifth are smartphones [**?**]. The user base of smartphone is expected to expand by about 42% a year, and with that grows the mobile web traffic [**?**]. 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 [**?**]. Globally, the ubiquitous appearance of mobile devices with the rise of cheap smartphones and tablets in developing countries such as Africa and India is creating a demand for available and affortable 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.

Browsing over mobile wireless networks is still a relatively new area and we currently face four overarching challenges: high cost, high latency, low bandwidth and unreliability. The inherent inefficiencies that cause these issues are connection overhead, redundant transmission and verbose protocol that make communication through mobile networks expesnive in terms of both time and money. In this paper, we focus on addressing the bandwidth constraint problem by identifying and eliminating data redundancy in content viewed through a mobile web browser. Specific pages and sites have the tendency to change little over time. Thus, when making a new request for some specific web content, the response will often contain a small number of modifications, but for the most part will be very similar to a previously requested version of the webpage. There has been some previous work in analyzing redundant desktop browser data [**?**], and we find out how thi redundancy compares with mobile browser data.

We present a technique which leverages these data redundancies to improve the bandwith utilization efficiency of mobile browsers. We use data deduplication techniques 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. We integrate our technique 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.

Our technique allows us to analyze redundancies across and within websites and to calculate the amount of potential bandwidth savings that can be obtained through data deduplication. As part of our analysis, we test various data chunk sizes with the aim of finding the optimal parameter settings for our technique. We also study various implementations of cache eviction algorithms to determine the optimal choice for our technique.

In summary, our main contributions are the following:

- The design of a system and protocol which integrate into the current mobile browsing framework.
- The implementation of two simulators, one for experimental purposes, one as a proof-of-concept prototype.
- The experimental evaluation of our system and protocol, with which we demonstrate significant improvements to current mobile network bandwidth requirements.

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

II. Previous Work

Some work has been done in trying to reduce bandwidth in wide-area networks. Ihm *et al.* [**?**] present a system for reducing bandwidth in wide-area networks for efficient Internet usage in developing countries. Their technique is similar to the one used in this work, *e.g.* chunking of the stream of data and Rabin fingerprinting of the resulting chunks. However, they are assuming large and small

Ihm and Pai [**?**] perform a thorough and deep analysis of Web traffic logs that have been collected over five-years period. They find many interesting facts about connection

speed of today's Web users, NAT usage, content type of the traffic and the traffic share of different types of Web sites. Moreover, they find that there is a fair amount of redundancy in the Web traffic and discuss different approaches for reducing the amount of this redundancy.

Qian *et al.* [**?**]

III. System Design

Our system is comprised of we add the following three steps: (1) Data Chunking which partitions incoming web content into fixed-size data chunks, (2) Fingerprinting which uses fingerprinting techniques to create a unique encoding of each unique data chunk, and (3) Caching which adds a layer on top of the web cache to only store unique chunks of data. We use a proxy server to perform the computationally intensive steps (1) and (2) to minimize the additional strain on the limited computational resources of mobile devices.

A. Data Chunking

B. Data Fingerprinting

C. Caching

D. Bandwidth Reduction Protocol

The second 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 Section **??**. Every time a new page is requested by the mobile browser, the following protocol is performed:

1) Mobile device 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 device 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 device reconstructs the entire requested page from its cache contents and the received list of needed chunks.

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 [**?**], and uses the `RabinHashFunction32` implementation of Rabin's fingerprinting method creating 32-bit fingerprints of a given data chunk [**?**].

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 **??**. The networked simulator uses the networked counterparts of these two components.

A. Offline Simulator

B. Networked Simulator

We implemented the networked simulator in its entirety in over 1100 sloc of Java (including all helper classes and interfaces, not including the rabinhash library). The networked simulator consists of the proxy server (`ProxyServerNet`) and the mobile client simulator (`SimulatorV3`), which is a wrapper for the networked mobile 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 a web browser (see Figure **??** for an example of the user interface of our mobile client). The simulator architecture can be seen in Figure **??**.

In more detail, the networked simulator implements our reduction protocol as follows. First, the mobile client opens a socket to the proxy server, which is listening for connections at a user-specified port. The mobile client sends a simplified HTTP request of the format *GET <url> HTTP/1.1* to the proxy server. It then formulates a proper HTTP request,

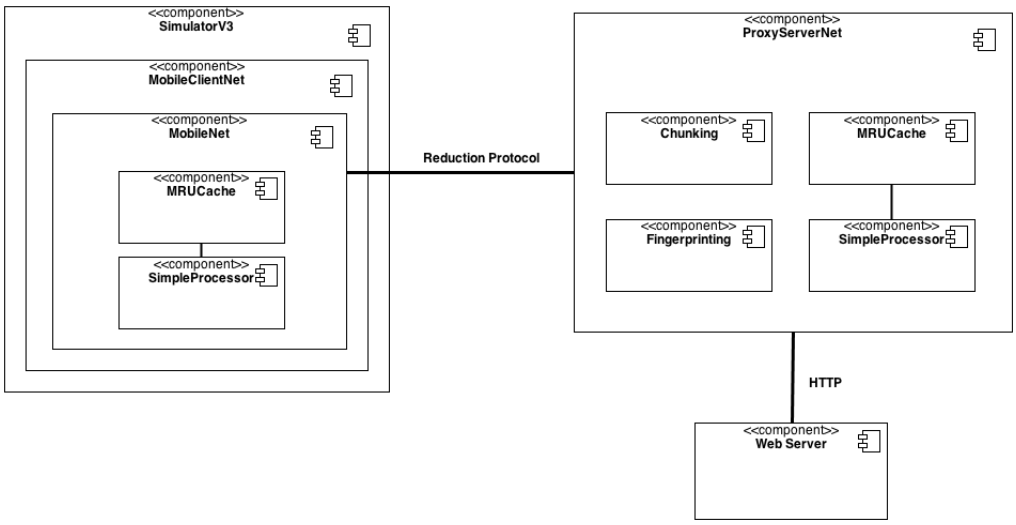

Figure 1: Networked Simulator Runtime Interactions.

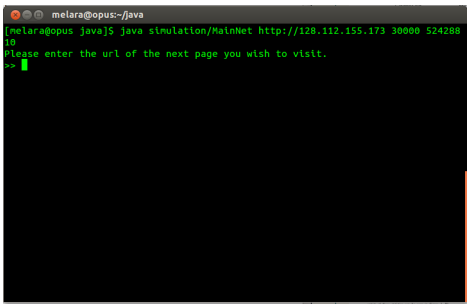

Figure 2: The User Interface of our Mobile Client Simulator.

including the User-Agent string of a mobile device[1] to ensure that it receives the mobile version of the requested web page from the hosting web server. Once the proxy has received the content of the requested page from the appropriate web server, it engages in the reduction protocol[2] with the mobile client, exchanging the relevant information via the network. At the end, the mobile 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.

During each round of the protocol, both the proxy server and the mobile client simulator display three statistics to the user: (1) The number of chunks (and hence fingerprints) processed, (2) The remaining cache capacity after processing a web page, and (3) The cache missrate for the last web page processed. Moving towards our ultimate goal of reducing the required bandwidth for mobile phones to save data plan usage, we can use these statistics to calculate the average mobile cache missrate for one series of protocol simulations, as well as the average number of bytes transmitted between the proxy server and the mobile client. A sample simulator output of these statistics can be seen in Figure **??**. We elaborate further on these calculations in Section **??**.

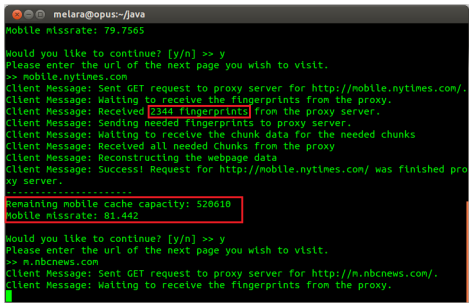

Figure 3: Sample Output of our Mobile Client Simulator.

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 address some caveats of our networked simulator. First, as our mobile client does not have the capabilities of a web browser and it receives pre-processed 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 $40X$ since the web server returns HTML content with

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

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

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. Using Different Caching Algorithms

V. Experimental Evaluation

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.

Figure 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 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.

The use of chunk size of 10 means that

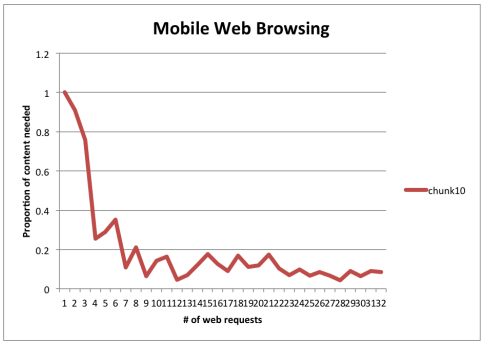

Figure 4: Mobile Web Browsing.

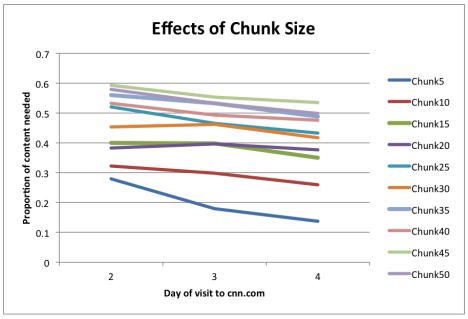

Figure 5: Effects of Chunk Size on portion of content needed.

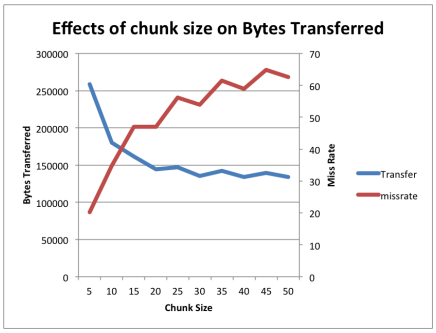

Figure 6: Effects of chunk size on Bytes Transferred.

VI. Discussion

Other techniques that have been used to reduce bandwidth are data compression [**?**] and partial responses [**?**]. For our

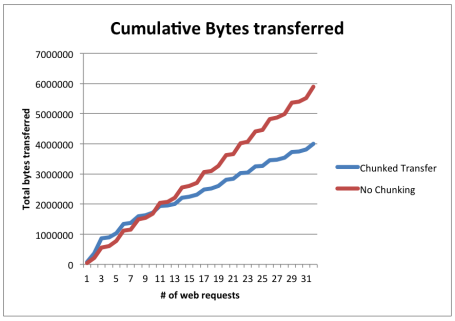

Figure 7: Cumulative bytes transferred during browsing.

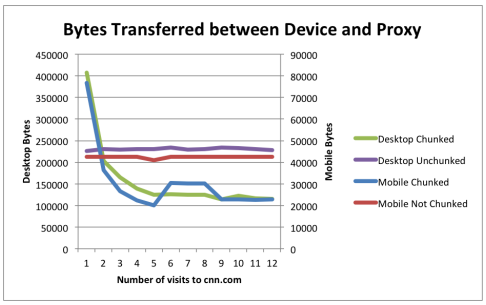

Figure 8: Desktop vs Mobile Browser page differences.

purposes, we use data deduplication because it has better efficiency with smaller data chunks whereas compression is better with larger data sizes. Partial responses reduce bandwidth by allowing the client to specify only the data it ; this was not relevant for us since we want to load the entire webpage requested by the browser. [But aren't our responses essentially partial? After all, the proxy does only send back the data our client specified as needing. Really need to point out the difference here]

VII. Conclusion and Future Work

We have built a simulator to perform these analyses using use two datasets, one obtained by capturing HTTP response packets through the packet analysis tool Wireshark and the other obtained through telnet requests. Additionally, we have implemented a proof-of-concept networked mobile client simulator and basic proxy server showing that our technique does not require changes to web server configurations, and does not alter the mobile browsing experience, making this a viable enhancement to mobile browsers benefitting mobile users. [**?**]

Acknowledgments

The authors would like to thank...