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 [**?**]. In 2009, this number was just 1%, moved up to 4% in 2010, and is expected to grow exponentially over the next X years [**?**]. 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 is that mobile networks in North America are already running at 80% of capacity and 36% of base stations are facing capacity constraints **[what does "capacity constraints" mean]** [**?**]. 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 the 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. One property of web data pertaining especially to content viewed through a web browser is of particular use for increasing the efficiency of mobile bandwidth utilization: Data redundancy. 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 will, for the most part, be identical to a previously requested version of this content. While there has been previous work in this area analyzing redundant desktop browser data [**?**], there has been minimal study of data redundancy in the area of mobile browsing.

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. More specifically, 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.

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 and cache sizes with the aim of finding the optimal parameter settings for our technique. 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. We also study various implementations of cache eviction algorithms to determine the optimal choice for our technique. 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.

II. System Design

A. Data Chunking

B. Data Fingerprinting

C. Caching

D. Proxy Server

III. Reduction Protocol Design

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 [3], and uses the `RabinHashFunction32` implementation of Rabin's fingerprinting method creating 32-bit fingerprints of a given data chunk [2].

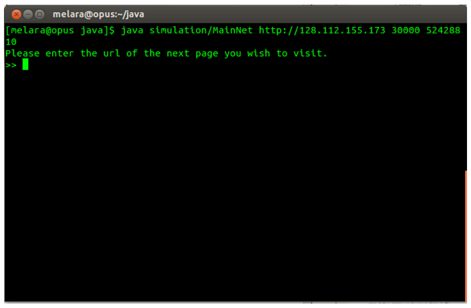
Finally, we created the `ISimulator` interface to build different kinds of simulators. Our offline version uses one or more `Mobile` devices and a `ProxyServer` to implement the simulation of our reduction protocol described in Section III. The networked simulator uses the networked counterparts of these 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 2 for an example of the user interface of our mobile client). The simulator architecture can be seen in Figure 1.

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



```
melara@opus:~/java
[melara@opus java]$ java simulation/MainNet http://128.112.155.173 30000 524288
10
Please enter the url of the next page you wish to visit.
>> 
```

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

HTTP request of the format *GET <url> HTTP/1.1* to the proxy server. It then formulates a proper HTTP request, 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 3. We elaborate further on these calculations in Section V.

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-

[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.

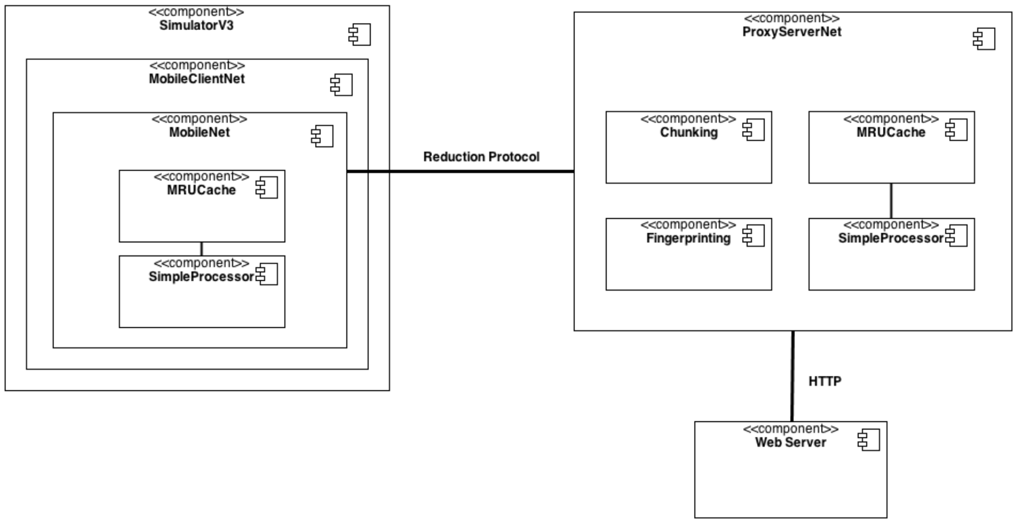
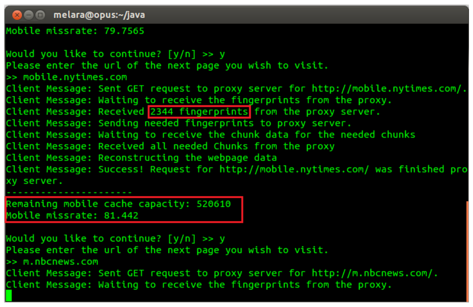


Figure 1: Networked Simulator Runtime Interactions.



```
melara@opus:~/java
Mobile missrate: 79.7565

Would you like to continue? [y/n] >> y
Please enter the url of the next page you wish to visit.
>> mobile.nytimes.com
Client Message: Sent GET request to proxy server for http://mobile.nytimes.com/.
Client Message: Waiting to receive the fingerprints from the proxy.
Client Message: Received 2344 fingerprints from the proxy server.
Client Message: Sending needed fingerprints to proxy server.
Client Message: Waiting to receive the chunk data for the needed chunks
Client Message: Received all needed Chunks from the proxy
Client Message: Reconstructing the webpage data
Client Message: Success! Request for http://mobile.nytimes.com/ was finished pro
xy server.
----------------------
Remaining mobile cache capacity: 520610
Mobile missrate: 81.442

Would you like to continue? [y/n] >> y
Please enter the url of the next page you wish to visit.
>> m.nbcnews.com
Client Message: Sent GET request to proxy server for http://m.nbcnews.com/.
Client Message: Waiting to receive the fingerprints from the proxy.
```

Figure 3: Sample Output of our Mobile Client Simulator.

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 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

VI. Discussion

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

[1]

Acknowledgments

The authors would like to thank...

References

[1] Udi 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.

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

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