Efficient and Transparent Dynamic Content Updates for Mobile Clients

by

Trevor Armstrong

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

Your file Votre référence
ISBN: 978-0-494-21117-5
Our file Notre référence
ISBN: 978-0-494-21117-5

Abstract

Efficient and Transparent Dynamic Content Updates for Mobile Clients

Trevor Armstrong

Master of Applied Science

Graduate Department of Electrical and Computer Engineering

University of Toronto

2006

We introduce a novel infrastructure supporting automatic updates for dynamic content browsing on mobile devices. We employ a pair of proxies, located on the mobile client and on a fully-connected edge server, to minimize the battery consumption caused by wireless data transfers to and from the mobile device. The client specifies her interest in changes to specific parts of pages by highlighting portions of already loaded web pages in her browser. The edge proxy aggregates updates as one batch to be sent to the client. Our approach is fully implemented using two wireless networking technologies, 802.11 and GPRS. Furthermore, we leverage SMS to implement and evaluate a hybrid approach. Our results show that our proxy system saves data transfers to and from the mobile device by an order of magnitude and battery consumption by up to a factor of 4.5, compared to the common client-initiated continuous polling approach.

Acknowledgements

First and foremost, I would like to thank my research supervisor, Professor Cristiana Amza. Without her support, patience, and guidance none of this work would be possible. She has allowed me to pursue my research interests, and for that, I am grateful.

Secondly I would like to thank Professor Eyal de Lara. His knowledge and experience in the field of mobile computing has been invaluable over the past two years.

I would also like to thank my labmate Olivier Trescases, whose hardware knowledge was crucial in getting over a number of the hurdles we encountered during the experimental evaluation.

Finally, I would like to thank my thesis defense committee members, Cristiana Amza, Eyal de Lara, Michael Stumm, and Shahrokh Valaee, for their thought provoking questions and suggestions which have improved the quality of this document.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this thesis, we introduce an automated and efficient approach for browsing HTML pages with dynamically changing content on mobile devices. Following the fluctuations of the favorite currency, stock value, or auction currently requires the user to reload all the pages in order to capture any changes to the data. Figure 1.1 illustrates just such a scenario. The costs of these data transfers to the user come in many forms, including slow data access, excessive battery consumption on the device and inconvenience due to the user's active involvement in constant data reload.

We observe that, while web pages may change frequently due to systemic reasons, such as updating the time of day or an add banner, the content relevant to the user does not change much. Based on this key observation, we introduce a general purpose infrastructure that allows the browsed HTML pages to be seamlessly updated only when content of interest to the user changes. Our approach greatly reduces the costs of updates by: i) allowing the users to mark the parts of each page that are of interest to them, ii) offloading the task of determining when those parts have changed to a resource-rich proxy and iii) leveraging the proxy for batching those updates and sending them to the user's device periodically.

We expect that our system will be useful in two kinds of browsing situations: Our first

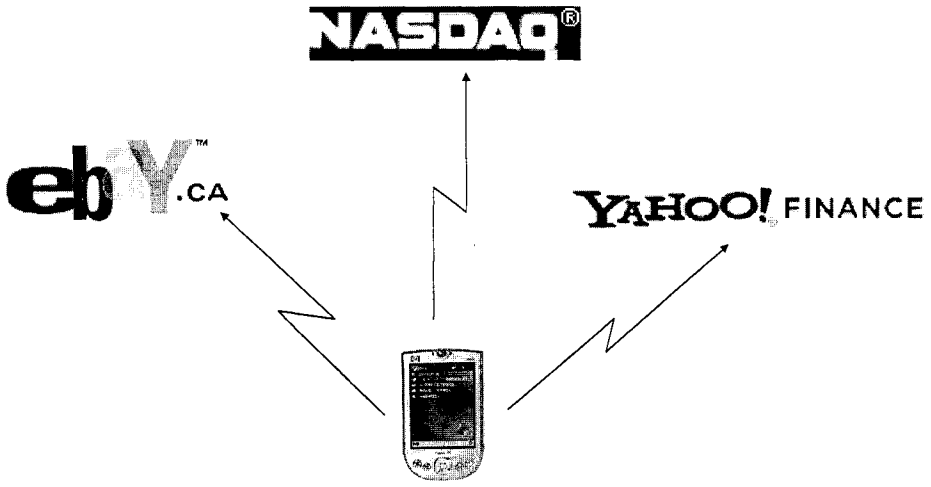

Figure 1.1: Mobile user interaction with dynamic content web sites

target is providing seamless low-cost content updates during active client web browsing. Imagine a user browsing dynamic content on her PDA during her daily commute or at an airport terminal waiting for her flight. We leverage our resource-rich proxy to save data transfers for both the case where the user wants to keep up to date with rapidly changing content for her favorite pages as well as for the case of browsing to random pages.

Our second target scenario is automatic periodic content refresh for the user's favorite content, for subsequent browsing while disconnected. This scenario corresponds to a user carrying a handheld device in her pocket, and having her preferred content (news, weather, stocks, etc) automatically updated whenever her device has the opportunity to network and significant changes to the content have occurred.

In our system, dynamic content is cached both within a local mobile client proxy and on an edge proxy with an always-on high-bandwidth low-latency connection to the Internet. The client registers her interest with the edge proxy. The edge proxy keeps track of the web pages cached at each client and polls the web servers involved for changes. We allow the mobile client to specify their interest in regions within each web page, using a very simple interface, by highlighting portions of already loaded web pages.

We also refine our interface by allowing the user to specify sensitivity thresholds for changes to numerical values. This specified interest allows the edge proxy to selectively accumulate content updates for all pages in the client's profile, and propagate updates as a single batch only when changes of interest to the client occur. Even when browsing previously unseen web sites, our proxy system provides savings to the user by aggressively prefetching all the embedded objects in a requested HTML page. This prefetching allows the user to bypass much of the latency associated with wireless communication.

In the first scenario above, our proxy system would reduce wireless communication by only reloading the dynamic content when a change, that is of interest to the user, has occurred. While in the second scenario the edge proxy would update the client automatically, with a batched set of updates each time the user had network access. This would allow the latest version of the content to be served locally the next time the user decides to load the page.

We implemented the system on a mobile PDA running Windows and have experimentally evaluated our systems benefits using traces from real web sites. We deployed an actual edge proxy in our lab from which our mobile device can connect using two alternative wireless networking capabilities: 802.11 and cellular communication over GPRS. Each of these networking capabilities offer different trade-offs in terms of data download costs. More specifically, access to content over cellular networks is ubiquitous and low power, but is relatively slow. On the other hand, transfers over WiFi (802.11) are fast, but have high energy costs. Indeed, an 802.11 card can reduce the battery lifetime of a PDA by up to a factor of six when in continuously active mode and by a factor of nearly two when in power saving mode [46].

In our experiments, we measure the data transfer and energy savings for several dynamic content refresh schemes, with our most advanced proxy system saving an order of magnitude in the amount of data transferred and reducing energy consumption by up to a factor of 4.5. Specifically, we implement and compare a poll-based scheme,

where the mobile proxy periodically polls the edge proxy for updates, and a push-based approach, where the edge proxy pushes updates to the device based on a schedule. We implement and measure the poll-based and push-based proxies and compare against a baseline without proxies on both WiFi and GPRS. Finally, we leverage lightweight SMS text messaging to signal when new updates have been created. The edge proxy creates a text message for the client, on which it piggybacks information such as the number of updates, size of updates, and pages that have changed. Thus, the mobile proxy can use this information to make an intelligent decision on whether to use its WiFi or cellular connection to download the updates. Specifically, the energy expenditure of transferring a short message on GPRS is expected to be dwarfed by the energy overhead of turning on the WiFi card. Conversely, for large messages, the superior download speed of WiFi translates into overall energy savings even if transfers occur at a higher power level.

We present results from experiments conducted by replaying four 3-hour URL traces collected from the following live sites: eBay.ca [17], the CNN Toronto weather page [15], the currency web site XE.com [55], and the financial site Yahoo! Finance [58]. We select parts of these web pages that the user would normally be interested in, such as, a particular auction's current bid, the current temperature, the value of a particular currency, and the value of a particular bond, respectively. These sites proved to be extremely dynamic, with page changes occurring at nearly every access, while, in some cases, the content of interest was far less dynamic. For example, of the 732 sample points in the eBay trace, there were 377 page changes, while the auction's current bid only changed 6 times.

Our results show that, in all cases, combining the update filtering with the batching edge proxy results in significant cost reductions. The client can learn in a single message exchange that there are no updates for any of the pages of interest. Conversely, when significant changes occur, transferring all relevant updates in a single batch bypasses much of the latency involved in Wireless Wide Area Network (WWAN) web browsing,

hence conserving battery power. We show that our proxy-based infrastructure reduces data transfers and energy consumption when using either of the two communication capabilities, i.e. GPRS and WiFi, alone. Furthermore, we have determined the threshold message size, for which using WiFi is more energy efficient than GPRS. This information is leveraged in our hybrid approach which uses both wireless connections in conjunction with SMS messages and allows for even further energy savings. Overall, we save data transfers to and from the mobile device by an order of magnitude and battery consumption by up to a factor of 4.5, compared to the client-initiated proxyless approach. Our results also show that even when the client is browsing new, previously unvisited, web pages, our proxy system has a beneficial prefetching effect by batching downloads for the page and all its embedded objects into one data transfer. As a result, in this case our proxy reduces energy expenditure by 69% when browsing over GPRS, and 15% when browsing over WiFi.

In summary, the contributions of our work are two-fold:

1. We provide a simple interface to allow users to automatically receive notifications on changes to web based dynamic content.

2. Our system significantly reduces the amount of wireless communication required to keep up to date with dynamic content, which in turn reduces battery consumption and extends the battery life of the mobile device.

The outline of the rest of the thesis document is as follows. Chapter 2 presents the background in mobile devices necessary to place our techniques in context. Chapter 3 introduces our proxy based approach. Chapter 4 presents our experimental platform and methodology. Chapter 5 presents our results. We provide a summary of our results, and a discussion of our contributions in Chapter 6. Chapter 7 discusses related work. Chapter 8 concludes the thesis.

Chapter 2

Background

A mobile device refers to any computing device that is normally used away from a fixed location and has been manufactured specifically to be portable and functional while being moved. This is a very general definition which encompasses everything from the most feature rich laptop computer to the simplest day planner to the smallest of sensors. For this work we will further require that the device be able to download, render, and display web pages. This restriction limits us to laptops, network enabled Personal Digital Assistants(PDAs), and smart phones (a PDA with built in cellular capabilities).

This chapter will begin with a general overview of the field of mobile computing, before delving into the current state of web browsing on mobile platforms. We will then discuss several wireless communication technologies, including GSM, GPRS, SMS, and WiFi.

2.1 Mobile Computing

Mobile Computing research is becoming more and more important as mobile devices are increasingly a part of everyday activities. The distinguishing characteristics of mobile devices have led to many research directions within the Mobile Computing research area. These research directions range from improving the usability of small devices,

to networking in mobile platforms.

For example, a downside of the increased portability is a greatly reduced screen size. In some of today's smart phones the screen can be as small as 2 inches diagonally. This makes browsing web pages and documents that were designed for large desktop monitors tedious and time consuming. To combat this problem, researchers have investigated transcoding techniques and content adaptation [11, 32, 38, 31] for the mobile platform. This work ranges from adapting the layout of the content in such a way to optimize the screen real estate and minimize scrolling, to altering the content itself in order to make it more acceptable for the target platform. Such transcoding techniques include reducing the size, fidelity, and/or changing the file type of the objects before they are delivered to the mobile device. This modification can be based on user preference, community profiles, and/or past history.

Another problem introduced by the mobility of the device is providing network access to the device as its physical location changes over time. The IP address of the mobile device will change as the network access point changes. As a result current routing techniques would not work. Researchers have proposed protocols to work around this problem [27, 26, 42, 34, 54]; along the lines of using a static third party node on the network to route and tunnel traffic through. Since this third node does not change address, it can always be used to contact the mobile device.

Arguably one of the most pressing issues facing mobile computing today is battery life. The trend in today's mobile electronics industry is to create more feature rich devices in a smaller form. As the devices become smaller, the size of the included battery is also decreasing. This trend which includes new and improved hardware (network interface, faster processors, larger & brighter screens) is placing a growing strain on the battery life of the device. This fact is best illustrated with the following example.

In April of 2005, Bill Gates (Microsoft's chief software architect) detailed his company's plan to bring an Ultra Mobile Personal Computer (UMPC) to the marketplace.

His goals were simple, a small tablet PC with a 7" screen, always connected network access, and all day battery life. This device would be ideal for the mobile professional and student alike. In the summer of 2006 the first series of UMPCs will be released. They will be small 7" tablet PCs, with built in WiFi, just as Gates had envisioned. However the advertised battery life is only 2.5 hours. This failure to reach the target battery life has marketplace experts such as CNN Money [16] predicting failure for the product line before it has even hit store shelves.

Many researchers have tackled the problem of reducing energy consumption for mobile devices, attacking the problem from many different angles. To reduce the energy consumption of the processor, researchers have proposed dynamic voltage scaling [47, 40, 41]. This technique allows the device to adaptively change both the processor speed and voltage depending on the processing needs of the applications currently running. Other researchers have looked at adapting the applications and operating systems that run on the mobile devices [32, 59, 53, 23] to better utilize the battery. Yet other researchers have investigated reducing the energy consumption of the network interface, which is considered one of the most power hungry components of the mobile device. Thus, conserving energy by intelligently using the network interface of a device [24, 45, 46, 30, 10] has received much attention in the research community. Work in this area has included using proxies to reduce wireless traffic [24, 10], modifying the send and receive patterns of the device to allow for aggressive sleep patterns [45, 30], and using alternate low power hardware to wake up the network interface when there is incoming network traffic [46].

2.2 Web Browsing on a PDA

The evolution of web browsing on the PDA can be closely tied to the evolution of Pocket Internet Explorer (PIE) for the Windows Mobile platform. It was one of the first web browsers written specifically for the mobile platform, and its evolution is an excellent

illustration of how far we've come.

PIE 1.0 was released with Windows CE 1.0 in November 1996. It was a basic web browser in every sense of the word. It lacked support for cookies, HTTPS, tables, frames, GIFs and much more. As the years went on each new version saw the inclusion of more and more features that help make PIE a valid option when browsing the web on the go. PIE 1.1 brought with it cookies, HTTPS and SSL. Version 2.0 added many new features: offline browsing, resizing images to fit the screen, and richer HTML support, including framesets and tables. By 1998, PIE 3.0 had added support for java script, and in version 4.0 java applets and advanced HTML features had been added. The current iteration, available in Windows Mobile 2005, is a nearly complete and very feature rich browser, supporting nearly all the functionality of its desktop counterpart.

As with the desktop, PIE isn't the only option when browsing the web on a PDA. In 2004 Opera released a version of its popular desktop browser which they had optimized for the small screen of the mobile device while retaining most of the features of its desktop counterpart. The benefit of this browser is that it does not require a Windows Mobile based device and can run on Symbian and various flavors of Linux, as well as Windows Mobile.

Although the software has matured over the years, the method with which the user interacts with the web has remained the same. It is still a client-server model, where users must make individual requests for the content of interest. RSS feeds [56] have helped improve this interface by aggregating large amounts of data and displaying it at once. However, as many of the other features have evolved to better suit the mobile environment, so too must the way we retrieve data from the internet.

2.3 GSM and GPRS

The Global System for Mobile Communications (GSM) is the most popular standard for cellular phones in the world. As of mid-March 2006 GSM service is used by over 1.7 billion people across more than 210 countries [21]. This represents approximately 77% of the world's cell phone users. In fact the European Union has mandated that their countries only use GSM cellular technology, in an attempt to maximize interoperability. The reason for GSM's popularity lie in it's interoperability with most service providers throughout the world, allowing for hassle free international roaming. This interoperability is a result of the fact that the GSM specification gives detailed requirements to all hardware vendors, which ensures that one manufacturers product will work seamlessly with a competitors product.

GSM was designed with digital networks in mind. It was the first cellular technology to use digital channels for both its signaling and speech channels. This design choice meant a higher voice quality but also that data communication was built into the system from the beginning, and as a result GSM was the first technology to support the Short Message Service (SMS).

The data communication inherent in the GSM specification eventually lead to the development of the General Packet Radio Service (GPRS). This service uses unused TDMA channels in the GSM network to provide the mobile user with moderate speed data transfers. Theoretically GPRS is capable of 64 Kbps data transfer [21]. However thanks to the shared medium nature of GPRS (multiple users share the same transmission channel) and the fact that voice is prioritized higher than data in a GSM network, the actual throughput can be much lower and extremely variable, with peak round trip times of over 1 second [9].

Browsing the internet on your GPRS enabled device is not only a time consuming task, but it may also be a very expensive activity, depending on your service provider. Below is a sample of some of the data rates being charged:

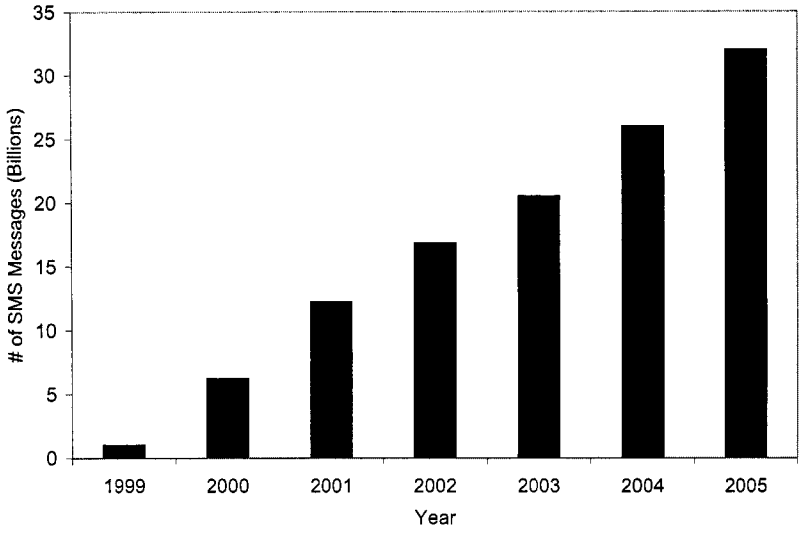

Figure 2.1: Total number of SMS messages sent by year [52]

- Rogers Wireless (Canada): $15/month for 1.5 MB after which it's $21/MB [44]
- Cingular (United States): $19.99/month for 5 MB after which it's $8/MB [14]
- O2 (United Kingdom): 8 Euros/month for 5 MB after which it's 1 Euro/MB [33]

2.4 SMS

The Short Message Service (SMS) is a service that started with GSM phones and is now available on most digital phones around the world. SMS allows users to send and receive short text messages to other phones, computers and even landline telephones.

Message delivery is best effort, using a store and forward mechanism at multiple hops throughout the cellular network. As a result, text messages are often susceptible to delay, especially when traversing from one operators network to another. However, as text messaging grows in popularity, service providers are allocating more resources to improve the SMS experience.

Since it's inception in 1995 text messaging has exploded into it's very own phe-

nomenon. Figure 2.1 illustrates the number of text messages sent each year around the world since 1998. That number has dramatically increased each year, and in the 6 years since these stats have been recorded, the number of text messages being sent has increased approximately 3200%.

The text messaging industry is big business for service providers:

- T-Mobile in the US charges $5/month for 400 messages/month, with additional messages costing $0.10, and $15/month for unlimited text messaging [51]
- Rogers in Canada charges $2/month for 25 messages/month with additional messages costing $0.15 [44]

There are also numerous vendors that provide services to users through SMS messages. For example, Jamster [28] will send you jokes, dating tips, sports news, and horoscopes for $5.99/month for each service. Zingo taxi [60] will give you location based information on taxi availability. Paypal [36] allows it's customers to send money securely to another person simply by texting Paypal the phone number of the recipient and the amount of money to transfer.

One of the biggest and most encompassing SMS services is available through Google SMS [20]. With this service users can text information requests to GOOGL (46645, only in the US) and receive the response by text message. Users can get driving directions, movie show times, language translation, currency conversion, local listings and a whole host of other information.

2.5 WiFi

WiFi is the brand given to any products that pass testing demonstrating they adhere to a set of product compatibility standards based on the IEEE 802.11 specifications. These standards were laid out to ensure that equipment from different vendors could be used

in the same Wireless Local Area Network (WLAN). Although WiFi is the brand given to the equipment that is based on the 802.11 specification, these two terms are used interchangeably when discussing the equipment and network (eg. WiFi network card & 802.11 network card).

The typical WiFi setup usually contains one or more Access Points (APs) and any number of clients. Clients connect to the access point of choice and route any traffic through it. Any place that has access to a WiFi AP is called a "hotspot". Free hotspots are becoming more and more common in public places such as coffee shops, book stores, hotels etc. However there is still a market for charging users for WiFi access on the go. Several service providers own hotspots in airports, hotels, and coffee shops throughout North America, and they charge for access to them. For example, in Canada, Bell charges $25/month for unlimited access to any of their hotspots, likewise, in the US, T-Mobile charges $20/month for access to theirs.

Since we've already discussed communication over cellular GPRS, a comparison of the characteristics of WiFi & GPRS may help illuminate the trade offs inherent in both types of communication.

Throughput The two most common flavors of 802.11 are 802.11b and 802.11g with throughputs of 11Mbit/s & 54 Mbit/s respectively. However throughput will be limited by the connection the AP has to the internet. GPRS has a theoretical limit of 64 Kbit/s.

Coverage WiFi networks have a limited range of between 45 to 90 meters from the AP. Although free access points are common throughout many cities, coverage is still no where near complete. Whereas GPRS has nearly ubiquitous coverage since it uses cellular towers for communication.

Energy Consumption For smart phones that have a cellular connection, communication over GPRS adds very little overhead, since it uses the existing hardware.

However to communicate over WiFi the device requires a secondary WiFi interface which has been shown to be power hungry [46].

The WiFi standard leaves features such as connection criteria, authentication and roaming totally up to the vendor of the wireless adapter. As a result, one wireless adapter may perform substantially better and with totally different characteristics than another adapter does.

It seems unlikely that WiFi will directly compete against cellular data communication; not only is the limited range of WiFi a problem, but the current 802.11 standards fail to address the issues above, mainly roaming and authentication, which are key to ubiquitous coverage. While the next iteration of this standard may make for an ideal, always connected, high speed mobile network, this thesis addresses the current challenges and trade offs between the two technologies.

Chapter 3

Our Proxy Framework

We begin this chapter by giving a general overview of our system in Section 3.1. This is followed by a detailed description of each component in Section 3.2.We describe the interface that we offer the mobile client and the way we keep track of the client's interest in Section 3.3. Section 3.4 details the operation of the system. Building on this basic framework, Section 3.5 introduces our proxy enhancements for further communication and energy savings. We conclude this chapter by giving a brief overview of the implementation details of the system in Section 3.6.

3.1 System Overview

Figure 3.1 illustrates a high level view of how our system works. There is a mobile proxy, which resides on the mobile device, and a edge server proxy, which is a regular computer with no power limitations, sufficient memory and a high capacity link to the internet.

All HTTP requests from the device's web browser are routed through the mobile proxy. The mobile proxy contains a cache of previously requested files. If the current request is in the cache then it is served locally and no wireless communication occurs. However, if the item isn't found, then the mobile proxy creates a connection to the edge server proxy and forwards the request to it, where it will either be serviced from that

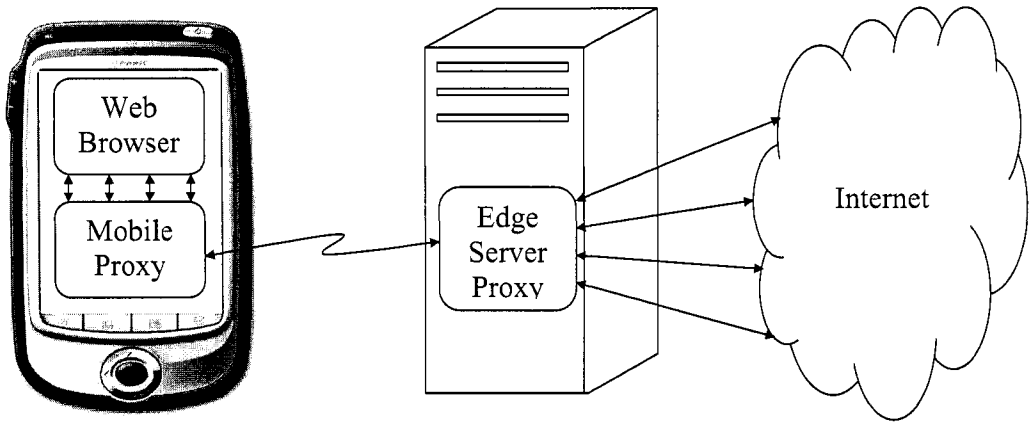

Figure 3.1: System Overview

cache or from the requested web server. At any given time there is only one active wireless connection; multiple concurrent requests are all forwarded over this single connection. This removes the overhead of having to constantly create and tear down wireless connections as new requests are made. It also reduces DNS lookup overhead as the device only needs to get the address of the edge proxy. In other words, the edge proxy acts as a single point of contact for all communication from the mobile device.

We assume a scenario where a mobile user is interested in browsing dynamic content from various web servers. The client interaction with many of today's web servers is repetitive in nature, such as, constantly polling an eBay auction to check the status of a bid, or refreshing a page that contains stock quotes to track the changing values of a stock. While browser caches support "get if modified since" mechanisms, this typically fails to save any data transfers due to frequent updates to parts of the page that are largely irrelevant to the user. These changes include ad banners or the time of day, and although the user may not be interested in them, they usually result in the page being reloaded almost every time.

To address this problem, we provide a simple mechanism that allows users to specify the information they want to keep track of in general HTML pages. Based on the user

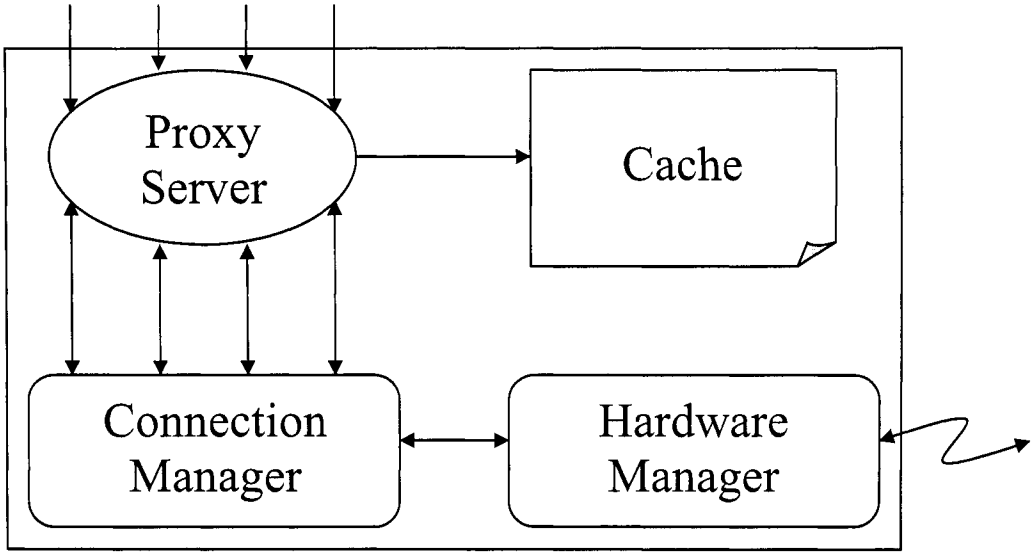

Figure 3.2: Mobile Proxy Architecture

interest, we offload the polling work from the mobile device to the edge proxy. The edge proxy performs high-frequency polling actions for detecting updates to the content of interest to the mobile clients. When changes occur, the updated cached entries are batched together and passed to the mobile proxy.

3.2 System Components

Figure 3.2 illustrates the components of the mobile proxy, which resides on the device. The mobile proxy's main job is to handle any user requests and communicate with the edge server proxy to process any cache updates. It consists of a proxy server, a cache, a connection manager, and a hardware manager which controls the state of the wireless connections available on the device.

The proxy server listens for HTTP requests from the device's web browser. If the response is in the cache, it services the request locally, otherwise it forwards the request to the edge proxy through the connection manager.

The connection manager's responsibility is to multiplex all the proxy servers sockets over the single wireless connection. This is accomplished by assigning a virtual socket to each of the local connections. This virtual socket is used when data is returned over the wireless connection to determine which local connection that data was intended for.

The hardware manager on the mobile device is responsible for determining which wireless interface the inter-proxy communication should use. The hardware manager makes its decision based on user defined preferences. The user can choose to prefer GPRS-only, WiFi-only or an adaptive GPRS/WiFi hybrid with the goal of optimizing energy consumption automatically. For example, in an interactive scenario of actual user browsing, where download speed is important or if avoiding monetary costs is paramount to the user, the user would set a preference for the 802.11 connection. In this case, the hardware manager first checks for the availability of an 802.11 access point before falling back to the GPRS connection if one is not available. In the hybrid case, the hardware manager bases its decision on which interface to use on the size of the data to be transferred. A long download of a large update on GPRS may consume more energy overall than the equivalent transfer over 802.11, even if the GPRS connection uses relatively less power.

The edge server proxy is illustrated in Figure 3.3. This proxy can be placed on any well connected computer. The edge server proxy consists of five components: proxy server process, connection manager, cache manager, cache, and update manager.

The connection manager demultiplexes the multiple connections coming over the wireless link and passes the individual connections to the proxy server for processing.

The proxy server process is an event driven server which interacts with multiple clients and serves their requests either from the cache or by forwarding the requests to the web server(s) in question.

The cache manager consists of an interface to the cache and a thread pool. The cache manager's responsibility is to keep the cache up to date. Each thread periodically polls

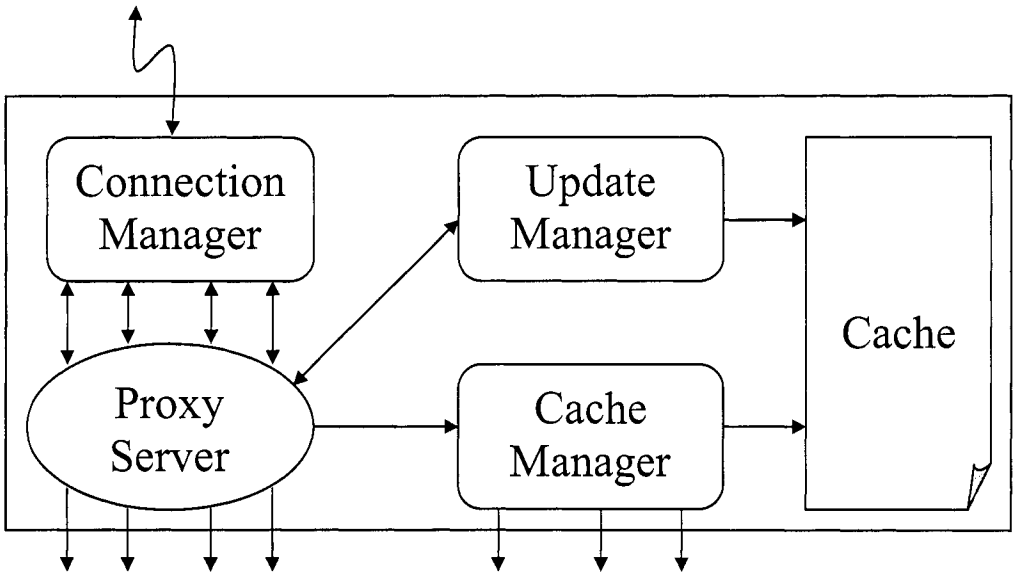

Figure 3.3: Edge Proxy Architecture

the web servers that a particular cache entry references, checking for any changes. The cache stores the interest profiles for all the mobile devices that registered their interest with the edge proxy.

When a cached page is changed, the update manager adds a reference to the changed content to the update batch of each mobile device that has registered interest in that particular page.

3.3 Client Interface

The user specifies her interest in changes to specific parts of each page by highlighting portions of the web page on her device screen, as illustrated in Figure 3.4. The end points of a highlighted region serve as the start and end points of an annotation that the system captures.

To keep track of the mobile client's interest in specific page regions even while the content changes, we use a well-documented tree technique for maintaining robust HTML

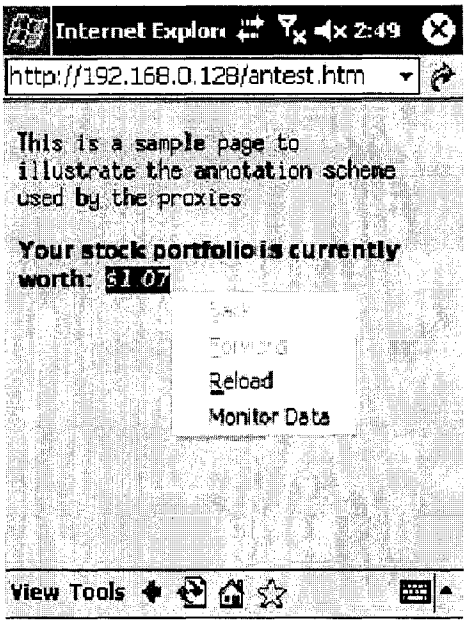

Figure 3.4: Client Interface Screen Shot

document locations [39]. This technique has been shown to robustly keep track of a location within a web document, in the face of typical value changes to dynamic content and even in the case of structural changes to the document, such as paragraph reordering or deletion.

Tree walks are the central component of robust HTML locations. A tree walk describes the path from the root of the document, through internal structural nodes, to a point with plain content at a leaf. Since tree walks incrementally refine the structural position in the document as the walk proceeds from root to leaf, they are robust for dynamic content page changes, such as with stock quotes, where the content itself changes while the structural position remains constant. In many cases, even when structural changes have occurred, it is possible to find the content of interest by visiting sibling nodes in the tree and using additional context information.

The annotations map the primary structures of HTML documents onto a general tree structural model. To create the general tree structural model, we follow the HTML

parse tree, a tree node is created for each structural markup tag in the document. For example, the HTML code for the web page in Figure 3.4 is presented in Figure 3.5. The highlighted text in the web browser screen shot creates an annotation corresponding to the following tree structure:

- 0-html
- 1-body
- 1-p
- Data: 50-54

Following the correspondence between this tree and the highlighted data is simple:

1. We enter the first node, in this case the `<html>` node.
2. We then enter the second node within `<html>`, `<body>`, skipping over the first child node `<header>`.
3. We enter the second child node of `<body>`, `<p>`
4. The data to be monitored stretches from the 50th item in this node to the 54th.

We can see that even if the highlighted value changes, the annotation describing it remains the same. The captured annotations describing the location of the highlighted portion in the document are transmitted from the mobile client to the edge proxy to help with identifying the portions of the page that the client is interested in. Dynamic content web pages may contain Java scripts embedded in them. The edge proxy needs to go through the process of rendering each page that it obtains from a web server in order to identify the portion of the page that the client is interested in.

Finally, our interface also allows the client to specify thresholds of meaningful change for numerical values.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Untitled Document</title>
<style type="text/css">
<!--
body {
background-color: #CCCCCC;
}
-->
</style></head>

<body>
<p>This is a sample page to illustrate the annotation scheme used by the proxies</p>
<p><b>Your stock portfolio is currently worth:</b>
<em>\$1.07</em></p>
</body>
</html>
```

Figure 3.5: HTML Code for Illustrating Annotations

3.4 Operation

When a mobile device first joins the system, it registers with the edge server proxy. The edge server proxy assigns each device a unique id so that it can subsequently differentiate between devices in the system. Differentiating based on IP address is not a sufficient means, since a mobile device may change IP addresses several times each day.

When a request is issued by the web browser on the device, the mobile proxy checks its cache. If the cache contains the corresponding response (local cache hit), the response is returned immediately to the web browser and no wireless communication occurs. If the response is not found in the cache (local cache miss), the mobile proxy forwards the request to the edge server proxy through it's connection manager. If a wireless connection hasn't yet been established the connection manager creates one through the hardware manager. It then assigns this local socket a virtual socket id and sends the request over the wireless connection.

The edge server proxy, in turn, checks its cache for the response and returns it from its cache if it is there. Otherwise the request is forwarded to the actual web server. If the response is a HTML page, the edge server proxy prefetches all the embedded objects within that page and batches them with the response to be delivered to the mobile client in one transmission. The edge server proxy then checks with the update manager to see if there are any pending updates for that client, if so, those cache updates are also included in the batch transfer.

Upon receiving the response from the edge server, the connection manager uses the virtual socket id to map the communication back to a local connection and returns the data to the mobile proxy. The proxy then caches the response and updates its cache with any other additional files included in the transfer. The response is then returned to the web browser. The client proxy acknowledges the receipt of any updates, such that the edge server proxy can remove those updates from the update manager's list for that device.

In our system, the mobile proxy learns that cache updates are available through three alternative means; polling the edge server, receiving pushed updates from the edge server, and receiving a SMS message from the edge server.

3.4.1 Polling for Updates

In the polling based scheme, the mobile proxy periodically polls the edge server proxy asking whether any updates are available. The edge proxy simply checks with the update manager, returning any available updates to the mobile proxy, or returning a "No Updates Available" message if no updates exist.

This periodic content refresh occurs automatically during active browsing sessions in order to keep the local client cache up to date, and in turn to minimize client perceived staleness and waiting time. During idle time, the client has the option to enable automatic periodic content refresh as well. This option is useful when preparing for browsing while disconnected and/or for implementing user notifications about changes in her content of interest, such as, a significant change in a stock price or currency.

3.4.2 Receiving Pushed Updates

Alternatively, for the push based approach, the mobile proxy listens on a particular port for incoming updates initiated by the edge server proxy. The update manager on the edge server keeps a schedule for each device in the system. Updates are sent to each device according to that client's schedule. This allows each client to specify different update periods, for example client 1 might be happy with updates every hour, while client 2 wants them as soon as a change has occurred. There is an obvious trade off here in the number of updates that are sent versus the staleness of the content on the mobile device.

When pushing updates to the client, the edge server proxy requires a valid IP address for the client. Hence, when the device's IP address changes, the mobile proxy is required to contact the edge server proxy to give it its new address.

3.4.3 Notification by SMS

The previous two methods require the user to have an active data connection in order to learn of updates. However, there may be times when the user is in a disconnected state[1] and would still like to receive notifications about changes to pages of interest. By harnessing the existing Short Message Service (SMS) infrastructure that most wireless carriers provide, our proxy system is able to provide this functionality.

In this scenario, when there are updates for the mobile proxy, the edge server proxy constructs and sends a single SMS message that is divided into two parts. The first part consists of control information for the mobile proxy, including the number of updates and the size of the download. The second part is an update summary, which is intended for the user. This summary includes a list of the pages that have changed, and if particular values were being monitored, the changes that occurred[2].

The mobile proxy intercepts any incoming SMS messages from the edge proxy. It strips off the control portion of the message and passes the remaining user portion back to the SMS handler for delivery to the user. As a result, both the user and the mobile proxy have information describing the updates. The mobile proxy uses this information toward making an intelligent decision on the appropriate connection to use for acquiring the updates. As a positive side effect, the SMS notification may enable the user to avoid using the browser altogether. For example, if the user is interested in a stock quote, then the SMS message containing the new value may convey all the information required. A similar SMS notification feature is provided by some vendors for certain proprietary data. However our system allows the user to use this functionality on virtually any data on the Internet.

[1] We assume that there is no valid data connection, but the device's cell phone is still on

[2] The size of the user section is restricted by the maximum total size of the message (160 characters for Rogers Wireless)

3.5 Infrastructure Enhancements for Communication and Energy Savings

The edge proxy keeps track of each client's interest in portions of HTML pages, expressed through the interface described in Section 3.3, as a client profile. The edge proxy polls the respective web servers for changes to the web pages in each client's profile. If the pages have been modified, the proxy determines whether these changes are meaningful for the respective client. Specifically, the edge proxy determines whether the differences between the old and new versions of a page are indeed located within portions of the page of interest to the client.

Furthermore, the edge proxy determines whether changes to numerical values that have a client-specified change threshold, exceed the recorded threshold. If a meaningful change occurs, the edge proxy accumulates the new version of the page as an update to be sent to the respective mobile client. As before, the edge proxy aggregates all updates to be sent to each client as a single update batch. Since each update overwrites the whole page, the number of updates in the update batch never exceeds the number of pages that the user has registered interest in.

3.6 Implementation Details

The edge proxy was implemented on a Linux based desktop PC in approximately 4300 lines of C++ code. It is an event driven server, using a single thread in conjunction with the select statement to handle all client requests. The cache manager consists of a pthread pool whose sole responsibility is to constantly pole the web sites contained in each user's cache, checking for updates to the pages as specified in that client's profile. These cache updates happen in parallel with the main threads actions, so to prevent any inconsistencies with the data, each client cache is protected by a lock. The main thread

is given priority in its cache interactions; as a result, if a request comes in from a client, and one of the cache manager threads is currently updating that client's cache, it will abort its action and wait until the main thread has completed its tasks before resuming.

The mobile proxy was originally implemented in C++ for a Linux based PDA. As a result it shared much of the code of its edge server counterpart, totaling around 2000 lines of code in all. It was designed as an event driven server in the same way the edge proxy was, but was simpler in design, since it only had to handle requests from one client, the mobile device's web browser.

We eventually moved to a newer device, one which did not support the Linux operating system, and as a result we had to port the existing code to the Windows Mobile 2003 environment. We choose the C# language for its feature rich library functions, and ease of implementation. The new mobile proxy contained many of the same ideas but the implementation details were different. For example, instead of using a select statement, the mobile proxy now used C#s asynchronous sockets to handle all network communication. C# also had the ability to more closely monitor the devices hardware and network connections. Using the OpenNetCF Smart Device Framework library [35] we were easily able to track the state of all the device's network hardware, and create any required network connections on demand. A task which was much more difficult and complex on the Linux based PDA. We also created a simple windows based user interface for the mobile proxy. This allowed the user to specify the cache update method (polling, push based, or SMS based), the polling period (if necessary), and the preferred wireless communication method (GPRS, WiFi, or a hybrid of the two). In total the C# implementation of the mobile proxy contains 3200 lines of code.

Although we have two versions of the mobile proxy, a C++ based version for Linux and a C# based version for Windows Mobile, the C# version is much more complete. After switching platforms, no further updates were made to the Linux version. As a result, a large amount of work would be required to bring the C++ version up to date

with its C# counterpart.

In order to allow our mobile proxy to intercept and parse SMS messages we needed a library which would tie into the Windows Mobile Mail API (MAPI). We used a modified version of the Mapirule code provided by an article on the Microsoft Developers Networks [49]. This code, written in C++ and totaling 1700 lines, allowed us to intercept SMS messages from the edge proxy, and ignore messages from other sources. The library then stripped the control portion out of the message and passed the user portion to the regular message handler, for delivery to the user's inbox. This control portion was then passed to the mobile proxy for processing. On startup the mobile proxy would load this dll and register for the callbacks that would signal it that a SMS message had been received.

To allow the user to specify their area's of interest within a web page loaded in their browser, we used a C# wrapper around Pocket Internet Explorer. Using the Intelliprog WindowsBrowserCE library [25] we were easily able to retrieve the user's highlighted area of interest. We then popped up a window asking if the user wanted to set a threshold value of change for this data, before transferring the area of interest to the edge proxy through the mobile proxy. This simple wrapper was written in 300 lines of C# code.

Chapter 4

Experimental Methodology

For our evaluation, we use a desktop PC running Redhat 9 as our edge proxy. This system contains dual Athlon 2600+ processors, 512 MB RAM, and a 100 Mbit/s Ethernet network connection. In addition, we equip this system with a second network interface card that interfaces the edge server proxy to a wireless access point.

The mobile device we use in the experiments is a HP iPAQ 6325. It runs the Windows Mobile 2003 operating system and comes equipped with a 168 MHz processor, 64 MB of ROM, and 64 MB of RAM. This device has built in 802.11b and GSM/GPRS interfaces as well.

In order to facilitate repeatable experiments, we took 3 hour traces from 4 real web sites and we compare all our configurations by replaying these traces in real time (i.e., each experiment is a 3 hour experiment). These traces are served by an Apache web server running on the same desktop machine as the edge proxy.

4.1 Real World Traces

For the traces, we used four popular sites; eBay.ca [17](we choose an auction that would be ending near the end of the trace period), the CNN Weather page for Toronto [15], the currency site XE.com [55], and the Yahoo! Finance page [58].

	Pages	Total Changes	Relevant Changes	Threshold Changes
XE.com	733	732	365	23
CNN Weather	733	732	3	3
Yahoo! Finance	732	731	211	19
eBay Auction	732	377	6	6

Table 4.1: Summary of the web site traces. Relevant changes are changes to the parts of the page that our "user" is interested in, described in Section 4.3. Threshold changes are the number of changes to those values when we provide a minimum threshold level for the change, as described in Section 3.5

To gather the traces, we created a Firefox browser extension that repeatedly visited each of the sites in the trace and saved each downloaded web page to disk. Each iteration was roughly 20 seconds after the previous one, resulting in approximately 733 copies of each site in the trace. Table 4.1 provides summary of the changes in the web pages over the three hour period of the trace.

The content of the traced pages proved to be very dynamic. Changes in content were either due to the volatile nature of the information being presented (stock quotes or currency values), or due to the inclusion of random advertisements. The eBay auction is a good example of frequent minor changes to the page, while the main bid-related information itself changes relatively infrequently. The auction that was chosen had 18 bids at the start of the trace. By the end of the trace, 24 bids had been placed. However, by analyzing the trace files, we were able to determine that the auction page had changed 377 times. This is due to the fact that the page contains a value showing how much time is left in the auction. This value was reported at the granularity of seconds during the last hour of the auction, and minutes prior to that.

The other sites in the trace were even more volatile, changing upon every access. The

Yahoo financial site contains a large amount of rapidly changing information, including the Dow, NASDAQ and NYSE, as well as several currency values. In addition, the site has several random advertisements. All these factors combine to cause the constant change. We monitored the value of the 10 year bond. This value changed 211 times over the 3 hour trace.

Similarly, the volatile nature of the currency values reported on XE.com resulted in a change to this page every time as well. We monitored the value of the Euro, which over the course of the 3 hour trace changed 365 times. This high rate of change is mainly due to the fact that XE.com reports the value to the nearest $0.00001.

The large number of random ads, along with a time of day value, on the CNN weather page were the main culprits for the numerous changes recorded for this page. This site changed on every access as well, while the actual temperature value reported changed only 3 times over the course of the trace.

4.2 Proxy-based and Standalone Configurations Used for Comparison

In the following section, we describe in detail the various proxy-based and standalone configurations we use for comparison with our main approach. By gradually introducing some of the features of our main proxy approach, we are able to demonstrate what aspects contribute to the overall wireless communication savings.

4.2.1 Baseline Configuration without Proxy

In our baseline configuration, the browser running on the mobile device polls all web sites periodically for the pages opened by the client for any change in the content. No proxies are used in this configuration. However, the browser's cache is fully functional.

4.2.2 Simple Proxy

In this configuration, we run the two proxies, the mobile device proxy and the edge proxy, and we use the edge proxy to poll for any changes to the data occurring at each separate data source. The proxy schedules an update to be sent to the client when there is *any* change to a web page. The edge proxy aggregates all updates to be sent to the user as described in our main algorithm. The mobile proxy pulls updates both upon a cache miss and periodically with the same interval as that of polling in the baseline configuration.

4.2.3 Intelligent Proxy

The intelligent proxy configuration is our proxy-based approach which filters out any updates to the mobile device if the parts of the page that the client is interested in have not changed. The client specifies interest by highlighting page regions through the interface described in section 3.3.

4.2.4 Thresholds Proxy

The thresholds proxy is an enhanced intelligent proxy where the client specifies her regions of interest within a web page, but can also specify a threshold of significant change for each numerical value. All updates for numerical value changes that are below the significant change threshold are filtered out by the edge proxy.

We use both a polling based and push-based thresholds proxy in our experiments. One drawback of our experimental setup is that our edge server is operating outside the Rogers GPRS network. As a result, our edge server is unable to create a connection to the device over GPRS as all incoming communication from an external source is blocked by the Rogers firewall. In order to facilitate push-based experiments over GPRS, our mobile proxy creates a persistent TCP connection with the edge server. Updates are then pushed to the mobile device over this connection.

4.3 Parameters used in Each Configuration

We use Pocket Internet Explorer (PIE) as our web browser on the mobile device. However, we use a simple wrapper around it to mimic the user and drive the experiments. All communication uses HTTP/1.1. In our baseline configuration, IE is running alone on the mobile device. The web browser contains a cache of its own, and as a result, after the first round of communication, the majority of the requests consist of if-modified-since requests from the browser for validating the cached items.

The browser is set up to visit the four sites in the trace, once every 4 minutes. This means that over the 3 hour experiment, each of the 4 websites in the trace is loaded 45 times. The period with which the pages are loaded is irrelevant, except for allowing the experiment to complete in a reasonable amount of time and to allow for full download of the respective pages over WiFi or GPRS.

In our experiments, we select the data of interest as follows: the current bid for the E-Bay auction, the current temperature for the weather site, the value of the 10 year bond for the financial site, and the current value of the Euro in US dollars for the currency site.

As reported in section 4.1, the values of the Euro and the 10 year bond are very volatile. This is mainly due to the resolution with which these sites present the values, the nearest \$0.00001 and 0.001%, respectively. For our threshold proxy configuration, we set conservative thresholds for each of these values in an attempt to observe the impact of threshold filtering on the results. We set the thresholds to a \$0.001 change for the Euro and a 0.005% change for the 10 year bond. We believe these settings give us a conservative estimate in our evaluation for the potential communication savings in most real scenarios.

4.4 Experimental Setup for Power Measurements

The simplest and most commonly used method for automated measurement of power dissipation in a mobile device uses a precision ammeter. In this traditional method, the device is powered by a low-noise constant voltage source. The precision ammeter, equipped with a serial communication interface, is placed in series with the device's power delivery path. Energy is computed as a function of the measured current and supply voltage. This approach can result in very high accuracy, low bandwidth current measurements, but it is not practical for today's low-voltage devices which typically operate from a single Lithium-Ion cell. During startup, the high in-rush current, I_{in} causes the device's voltage supply, V_{in} to drop due to the relatively large internal ammeter sensing resistance (5 Ω) and its parasitic inductance. In many cases, this drop causes the internal power management protection circuit included in newer devices to suspend startup. Hence, the traditional power measurement technique becomes infeasible, as we experienced first-hand with our transition from an older device to a more modern version.

Instead, we use an improved, non-intrusive power measurement technique, suitable for modern low-voltage devices, which uses a high-bandwidth current-sensor probe clamped around the power supply wire. The current probe is used to measure the battery current based on the magnetic hall-effect. The PDA battery is used instead of the voltage source, since the power management circuitry disables the PDA if the additional battery wires are left unconnected. These wires may be used either for digital communication in advanced "Smart Battery Packs", or simple local analog sensing functions. We connect the current probe to a two-channel oscilloscope through a calibrated amplifier. The battery voltage, V_{in} is also sampled by the oscilloscope using a voltage probe. The sampled current and voltage data is transferred to a PC using a serial interface. The total energy consumed during N samples of the oscilloscope data is calculated using formula (4.1), where f_s is the oscilloscope sampling frequency.

$$E_{tot} = \frac{1}{f_s} \sum_{i=1}^{N} V_{in}[i] \cdot I_{in}[i] \tag{4.1}$$

We are currently using a Tektronics TCP312 current probe, a Tektronics TCPA300 amplifier and a Tektronics TDS3032 oscilloscope to record the measurements.

Chapter 5

Experimental Results

We begin this section by showing preliminary energy measurements for our device in a variety of communication scenarios using GPRS and WiFi. Then, we present wireless communication and energy measurements performed while running our 3 hour traces in a scenario illustrating periodic content refresh. We compare several proxy-based approaches including push and poll-based approaches and the baseline proxyless approach in the two communication scenarios: using WiFi or GPRS alone for all wireless communication. We also investigate the use of SMS messages to signal the device that updates have occurred during periods of disconnection. By including important information in the message, such as the number and size of pending updates, we allow the mobile proxy to adaptively use either WiFi and GPRS to fetch the update batch.

5.1 Preliminary Energy Measurements

In this section, we characterize the energy consumption of our mobile device hardware. Specifically, we measure the current being drawn by the device as it utilizes each hardware component. In each case, all other non-essential hardware on the device is disabled. For example, when testing the 802.11 connection, we disable the GSM radio and the backlight of the device. We also characterize the cost of downloading data of various sizes over each

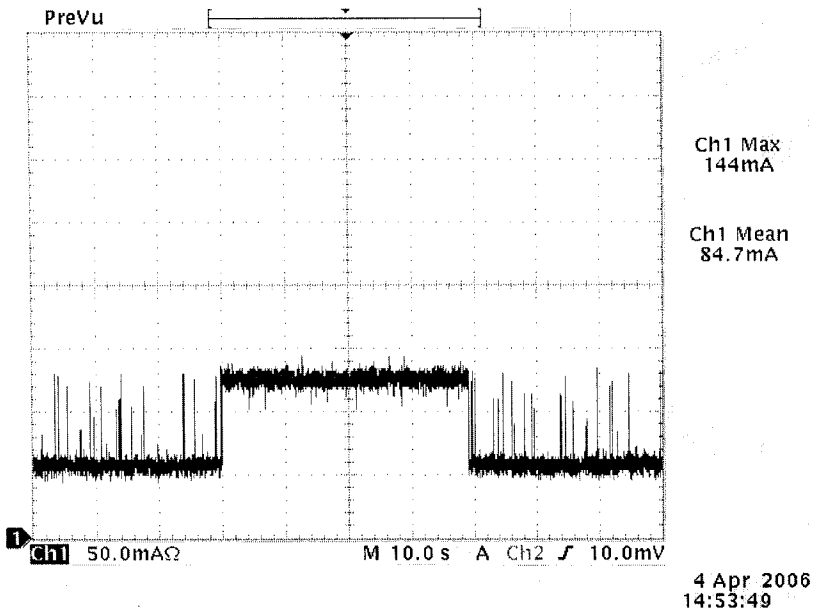

Figure 5.1: Processor impact on energy consumption *Scale 50 mA/div, 10 s/div*

of the wireless connections. These preliminary power measurements guide our design for power savings.

Figure 5.1 shows the current drawn by the device's processor. The processor transitions from an idle state to an active state, where it continuously processes data for 40 seconds before returning to the idle state. We can see from this screen shot that even while idle, the processor frequently becomes active for short periods of time, probably to handle interrupts and timers. During its active phase, the processor draws an additional 65 mA of current on top of its baseline 65 mA.

In the experiment depicted in Figure 5.2, we vary the brightness of the backlight while recording the current being drawn by the device. We transition the backlight from a completely off state, to the medium brightness level, and then to maximum brightness. At medium brightness the screen draws, on average, an additional 55 mA, while at maximum brightness it draws an additional 80 mA.

Figure 5.3 shows the current drawn by the device as the cellular GSM radio transitions

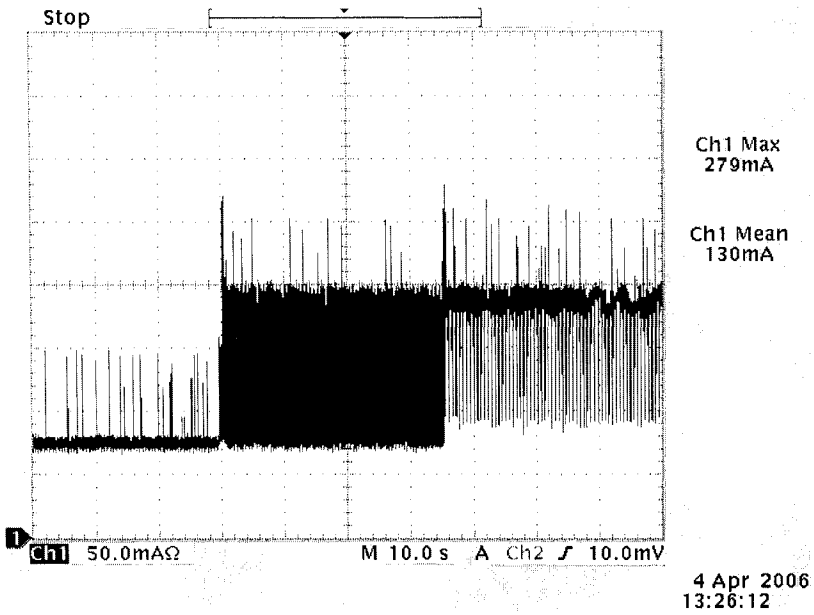

Figure 5.2: LCD backlight brightness impact on energy consumption *Scale 50 mA/div, 10 s/div*

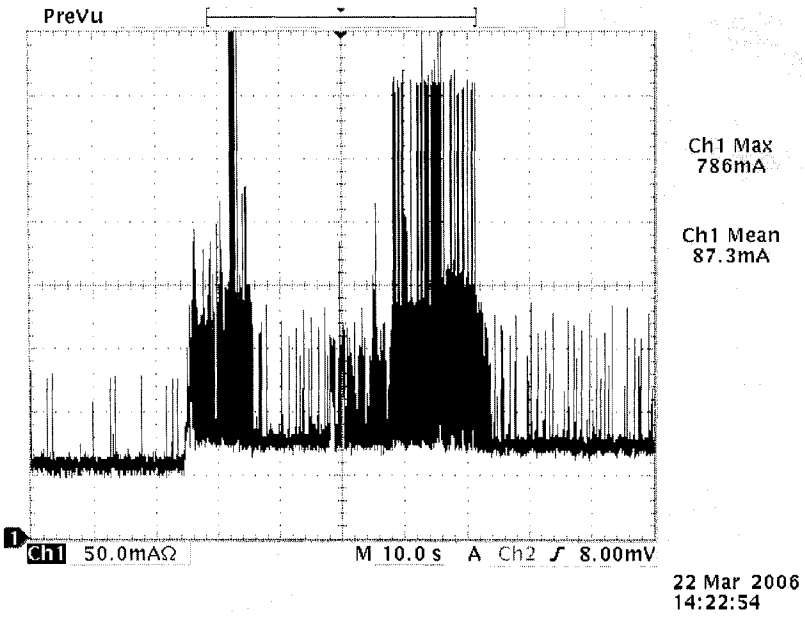

Figure 5.3: GSM radio state impact on energy consumption *Scale 50 mA/div, 10 s/div*

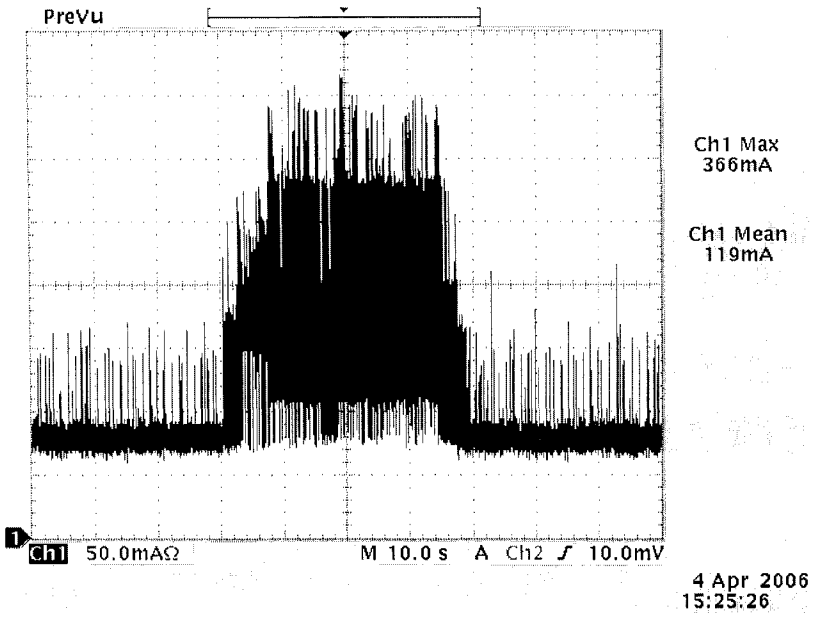

Figure 5.4: Effects of downloading a 160 KB file over a GPRS connection *Scale 50 mA/div, 10 s/div*

through a number of states. In the beginning, the radio is off, hence the phone connection is disabled. At about the 25 second mark of the experiment, the radio is turned on, resulting in a visible fluctuation in the current drawn by the device as the hardware is powered up and the drivers are loaded. This corresponds to a steady state with an additional current of 20 mA being drawn. At the 50 second mark, we dial the GPRS connection and a data connection is established. There are more fluctuations as hand shaking occurs to establish the connection, before the device settles back to the same level of power consumption as when there was no connection. This experiment shows that little or no energy can be saved by having the cellular radio turned off. Likewise, disabling the GPRS connection results in no energy savings, while there are penalties for connection establishment.

Figure 5.4 illustrates the current being drawn by the device while a 160 KB file is downloaded over the GPRS connection. The current varies considerably over the course

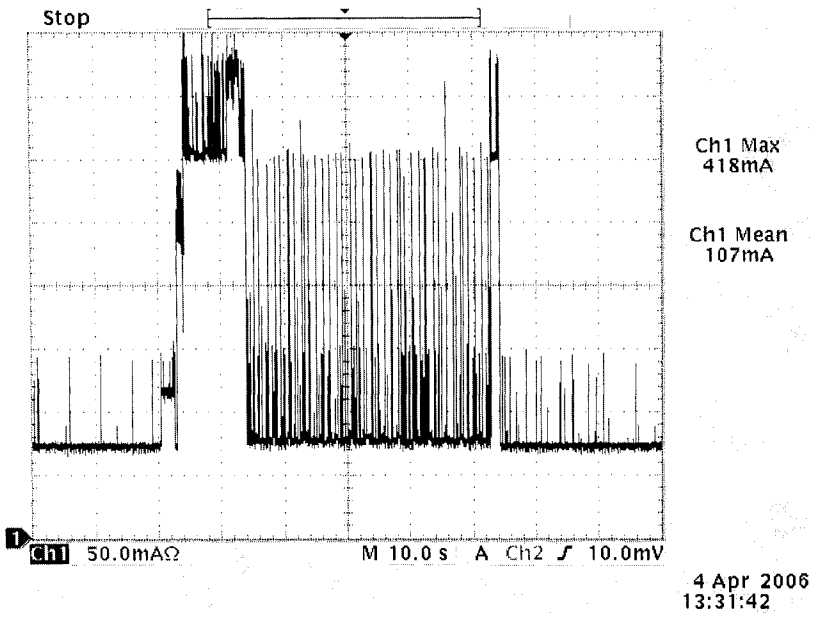

Figure 5.5: 802.11 radio state impact on energy consumption *Scale 50 mA/div, 10 s/div*

of the download, as packets are being sent and received, and as the processor becomes active to process the packets. On average, over the course of the download, the current increases by 100 mA (an 125% increase over the idle state consumption).

Figure 5.5 plots the current being drawn by the device when the 802.11 radio is switched on and becomes associated with an access point. This causes the current to spike to over 400 mA. After associating with the access point, the network card periodically communicates with the access point to check for incoming data. These periodic checks only require an additional 10 mA on average. The current spikes again just after the 70 second mark as the network card is disabled. This plot illustrates that even though the 802.11 radio is power hungry, aggressively switching it on and off may consume more power in the long run.

Figure 5.6 represents the current being drawn by the device as a 10 MB file is downloaded over the 802.11 wireless connection. During the download, the average current being drawn by the device reaches 375 mA, an increase of over 300 mA (nearly a 450%

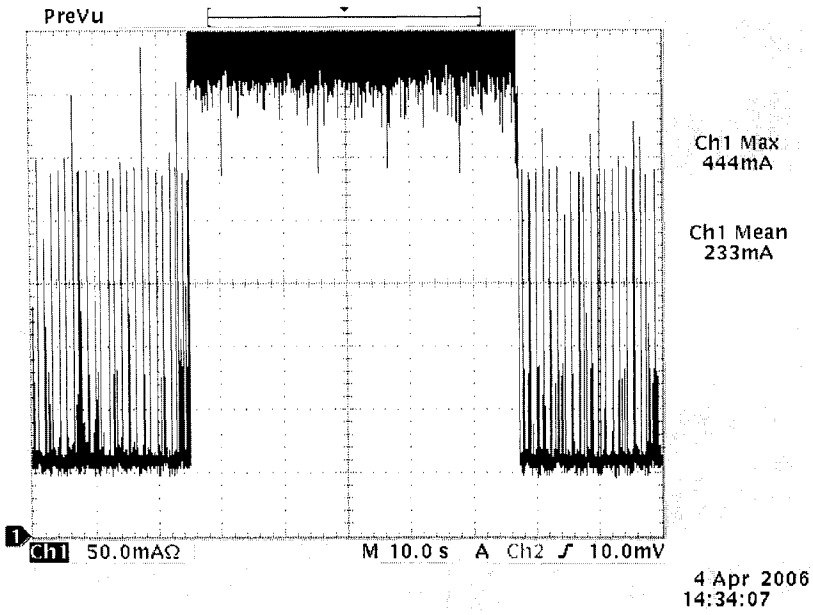

Figure 5.6: Effects of downloading a 10 MB file over the 802.11 connection *Scale 50 mA/div, 10 s/div*

increase) compared to the idle state.

Figure 5.7 shows the current being drawn by the device as it receives an incoming SMS message. The current spikes a couple of times over a 4 second period, with an average increase of 45 mA. We can see from this plot that receiving a SMS message is relatively low cost in terms of energy consumption.

In order to determine the message size for which the energy cost of downloading over GPRS is greater than that of WiFi, we downloaded several file sizes over both connections. In this experiment the device already has an active GPRS connection, since, as shown in Figure 5.3, toggling this connection has little to no energy savings. However we leave the WiFi connection disabled until it is needed. So the cost of downloading over the GPRS connection is just the cost of receiving the file, whereas the cost of downloading it over the WiFi connection also includes the penalty of activating that interface. Figure 5.8 shows the energy expenditure as a function of file size for WiFi and GPRS. We can see that the

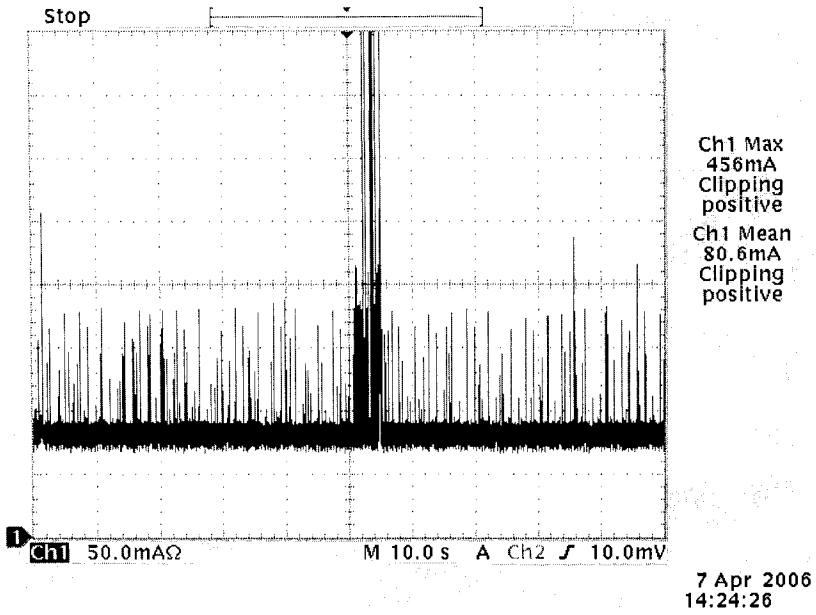

Figure 5.7: Effects of receiving an SMS message *Scale 50 mA/div, 10 s/div*

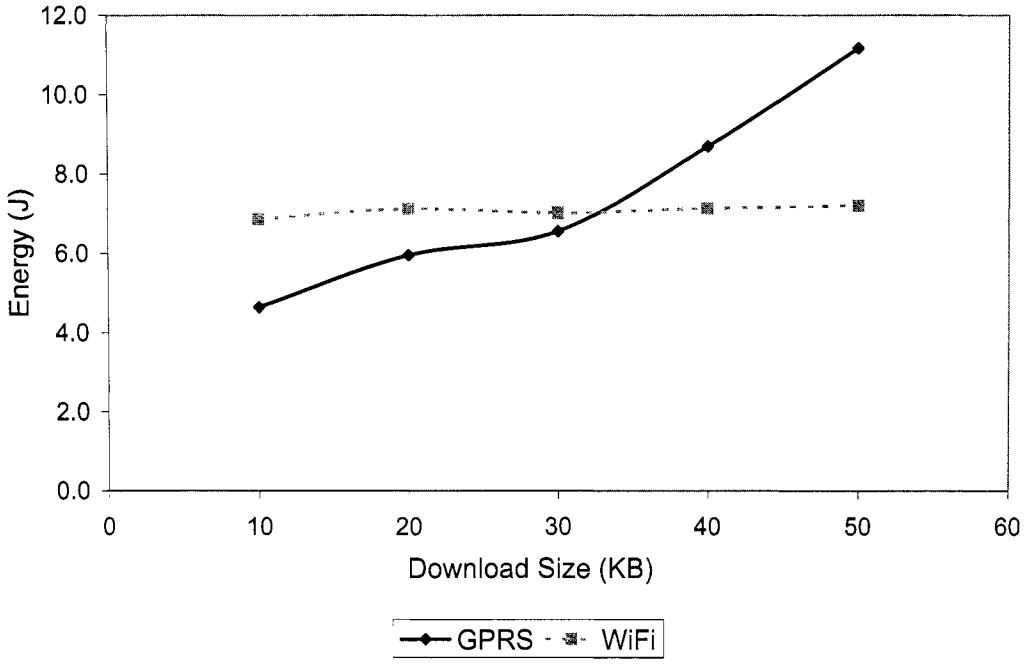

Figure 5.8: Energy cost of downloading data of various sizes over WiFi and GPRS.

cross over point is roughly 30 KB. This experiment tells us that any communication under 30 KB should use the GPRS connection, while for communication over this threshold, the WiFi connection is more energy-efficient.

5.1.1 Preliminary Energy Summary

There are several important factors that the previous energy measurements have pointed out:

- Disabling the GPRS connection has no energy benefit, and there are energy penalties for reconnecting.
- Downloading over GPRS increases current draw by 125%.
- Downloading over WiFi increases current draw by nearly 450%.
- There is a crossover point, in terms of download size, at 30 KB where it is cheaper, in terms of energy consumption, to download over the high capacity WiFi connection than the low capacity GPRS connection.
- Receiving an SMS message has very little impact on energy consumption.

5.2 Wireless Communication

In order to compare the proxy-based and proxyless configurations in terms of data communication, we ran all configurations introduced in section 4.2, i.e., without a proxy, simple proxy (polling), intelligent proxy (polling), and both a polling and push-based thresholds proxy, on the 3 hour traces described in section 4.1. In each configuration, the mobile client browser requests the URLs encountered in the trace over the 802.11 connection, while the Apache web server returns each saved page from the trace corresponding to the URL. We do not show the results of the experiment for both GPRS

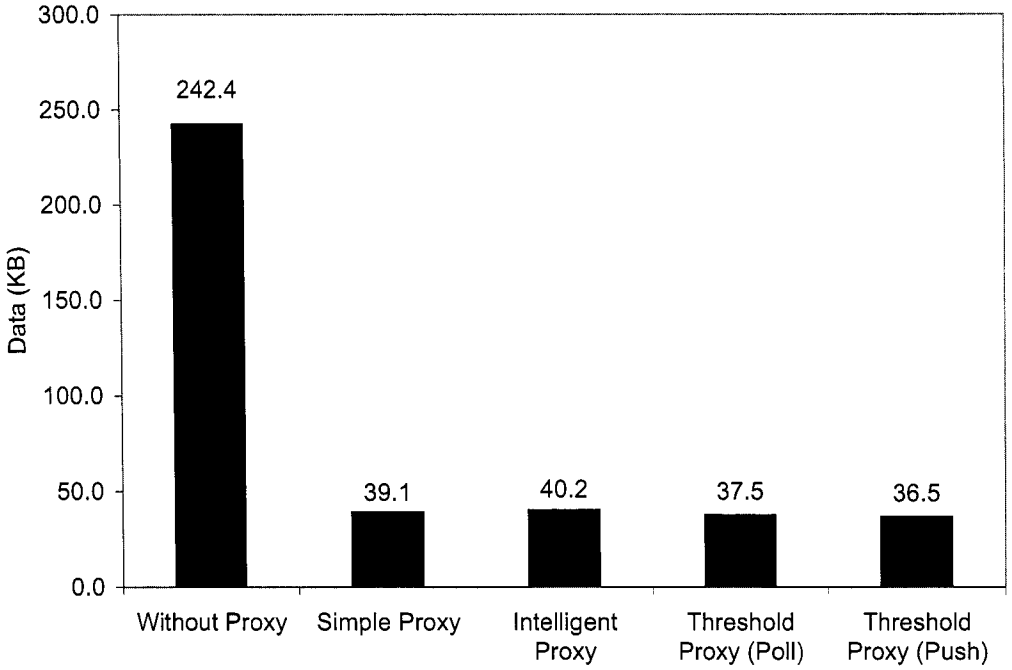

Figure 5.9: Transmitted data for the 45 accesses of 4 real web sites

and 802.11 because the results are virtually identical. We are measuring the amount of data transferred at the application level, so any minor differences between the control messages transmitted by these two wireless technologies are negligible.

Each of the proxy configurations was set to either poll for updates, or receive pushed updates, periodically with a 4 minute interval. The frequency with which the edge server polls each of the data sources was set to 2 minutes.

During each three hour experiment, the data is viewed a total of 45 times. The resulting total amount of wireless communication for each configuration is presented in Figures 5.9 and 5.10. Table 5.1 shows the total data transmitted and received in conjunction with the total number of updates and cache hit statistics of the mobile proxy cache.

Figure 5.9 illustrates the amount of wireless data that was transmitted from the PDA over the 45 accesses. We can clearly see that the amount of data that the PDA is

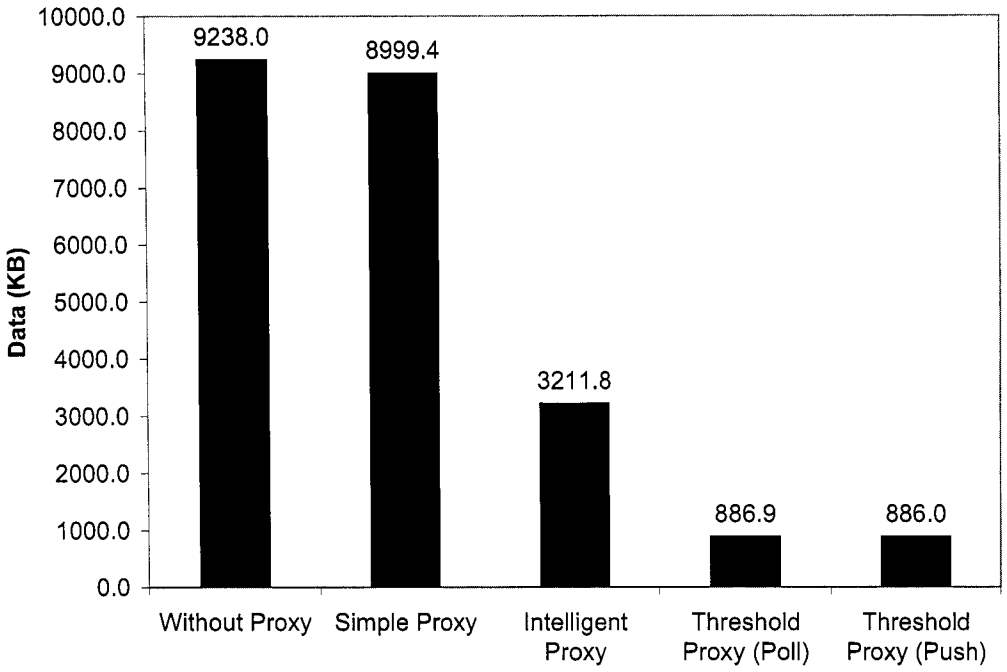

Figure 5.10: Received data for the 45 accesses of 4 real web sites

	Trans (KB)	Recv (KB)	Updates	Cache Hits	Misses
Without Proxy	242.4	9238.0	0	0	657
Simple Proxy	39.1	8999.4	220	521	109
Intelligent Proxy	40.2	3211.8	72	512	120
Thresholds Proxy (Poll)	37.5	886.9	11	536	105
Thresholds Proxy (Push)	36.5	886.0	11	536	105

Table 5.1: Experimental results for each configuration. Cache hits/misses are only for mobile proxy and not the web browser's cache. Updates are the number of page changes that occurred, several updates may be sent in one batched transfer.

required to send is greatly reduced in all proxy configurations compared to the proxyless configuration. The reason is that, for the proxyless configuration, each time the client wants to view the data, each of the HTTP requests must be sent in their entirety over the wireless link. In contrast, in any poll-based proxy configuration, the mobile proxy needs to send, at most, two 28 byte packets to the edge server proxy: an update inquiry packet and an update acknowledgment packet (only if an update has occurred). This is because, after the first round of communication, the edge server proxy knows exactly what the client is interested in, and as a result the client no longer needs to send the entire request. In the push-based proxy configuration, the mobile device saves an additional 28 byte packet per request, since no update inquiry message needs to be sent by the mobile device. This contributes to the small reduction of 2.7% for the data transmitted in the push-based approach compared to its poll-based counterpart.

In contrast to the data transmission graph, Figure 5.10 illustrates the significant differences between the various proxy configurations in terms of data received over the wireless link during the 45 accesses of our trace. We can see that the simple proxy saves only 2.6% of the data over the proxyless configuration. This is because, as discussed in section 4.1, each of the sites in the trace changes rather frequently. In particular, the content on all sites changes at least once during each 4 minute polling interval of the experiment. As a result, the simple proxy method downloads nearly the entire batch of data each period.

The intelligent proxy reduces much of the wireless data received by reducing the number of updates the edge server proxy sends to the mobile client. As we can see from Table 5.1, by only sending updates when parts of the page of interest to the user change, we reduce the total number of updates by a factor of 3. This translates into a 65.2% reduction in the amount of data received when compared to the baseline proxyless approach.

Finally, the thresholds proxy reduces the amount of data received over the wireless

link even further. By setting reasonable thresholds for numerical changes that warrant sending an update to the client, we reduce the number of updates by a factor of 20 when compared to that of the simple proxy. For example, by setting a threshold for the change, we prevent the edge server proxy from sending several dozen updates while the value of the Euro fluctuates between $1.22736 and $1.22740. This drastic reduction in the number of updates sent results in an order of magnitude savings in the amount of data received over the wireless link. Again the push based approach offers marginal savings in comparison to its poll-based counterpart. In particular, the push-based proxy registers a negligible 0.1% savings as a result of not having to receive the 28 byte "No Updates Available" packet that the polling approach incurs in the situations where no content change has occurred.

5.3 Energy Consumption

Figure 5.11 presents the average energy consumed by the device per download period (i.e., loading each of the 4 web sites in the browser) for the 3 hour experiments presented in the previous section, in six configurations: the baseline proxyless configuration using the 802.11 connection and using the GPRS connection, our poll-based thresholds proxy configuration using the 802.11 connection and using the GPRS connection, and our push-based thresholds proxy configuration using the 802.11 connection and using the GPRS connection.

We can see that all configurations of the thresholds proxy are superior in energy conservation compared to their proxyless counterparts. Our proxy system reduces energy costs by factors of 2.1 and 4.5 when used over the 802.11 and GPRS connection, respectively.

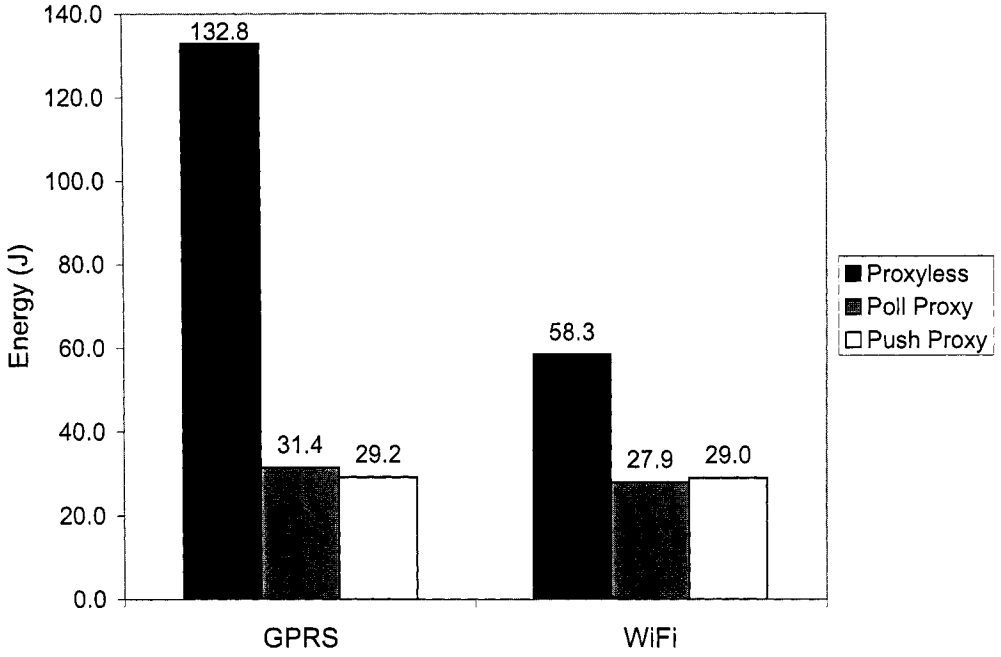

Figure 5.11: Average energy expenditure per download period

5.3.1 Energy Consumption in Push versus Poll Proxy

As illustrated in Figure 5.11, the differences in energy consumption between the push-based and poll-based proxies are small for both WiFi and GPRS. The push-based proxy using the GPRS connection conserves 7% energy per download period compared to the poll-based proxy. Maintaining a persistent connection with the edge server in the push-based configuration is more energy-efficient than requiring the device to create a connection, request an update, and tear down the connection during each period in this case. The push based proxy using the WiFi connection on the other hand, uses 4% more energy per download period than it's polling based counterpart. Constantly listening for incoming communication over the WiFi connection requires more energy than periodically sending update request packets. This push based proxy could potentially save more than the polling approach, if the device used a small listening window for receiving updates. However, the need to synchronize the clocks of both proxies over the WWAN

makes this approach unattractive.

5.3.2 Energy Consumption for Off-line Updates Using the Hybrid Approach

In this section, we analyze the energy consumption of the SMS based proxy system described in Section 3.4. The mobile proxy requests an update only when it receives an SMS message specifically informing it that there are updates available. The proxy uses the control information contained in this SMS message to determine the best interface to use for downloading the update. The proxy uses GPRS to download any updates under 30 KB and WiFi for updates over this threshold. This value was based on the results shown in Figure 5.8.

The average energy consumption of this proxy is illustrated in Figure 5.12 along with the best results for the GPRS and WiFi only proxies. The hybrid SMS proxy saves an additional 14% energy over the push based GPRS proxy and 10% over the polling WiFi proxy. The savings are a result of not having to send periodic update requests, or conversely, listening over the wireless channel for incoming updates, and from using the most efficient download method for acquiring the updates when they are available.

5.3.3 Energy Consumption for New Page Accesses

To determine the energy consumption for the case of visiting new pages i.e., cold cache miss, we ran an experiment where we viewed one of the pages in our trace with an empty browser cache and an empty proxy cache. The page used in this experiment contained 51 embedded files, consisting of dozens of small images, a couple of style sheets, and several javascript files. The HTML page and all of its embedded files were 185 KB. We used a proxyless setup as baseline for comparison.

The total energy required to view the selected page in each configuration is illustrated

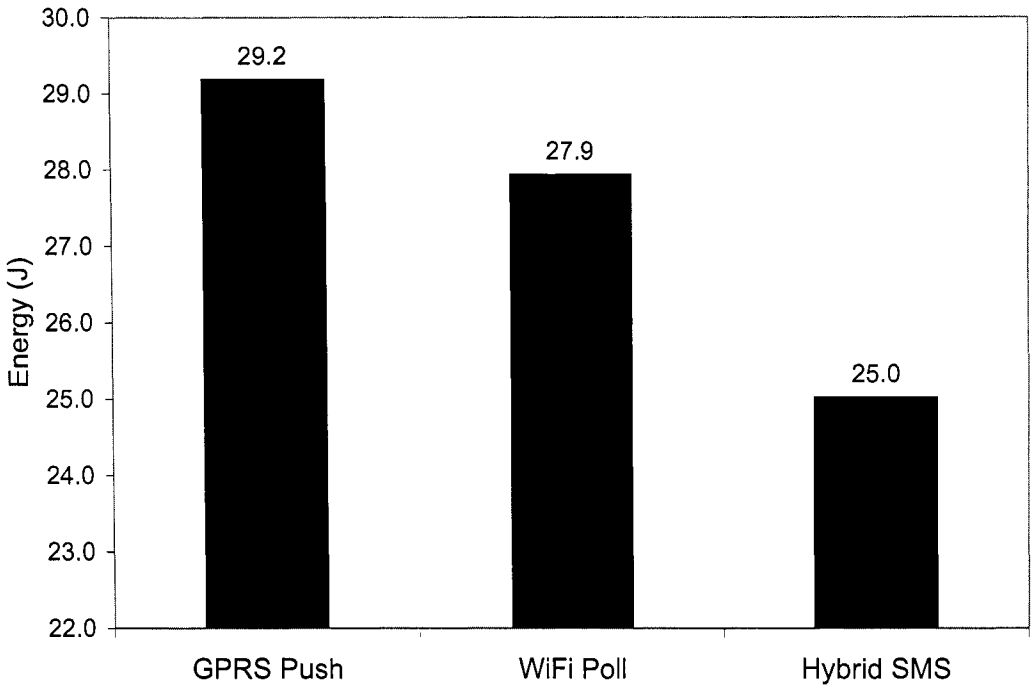

Figure 5.12: Average energy expenditure per per download period

in Figure 5.13. Compared to the proxyless approach, the energy expenditure is reduced by 69% when using the GPRS connection in conjunction with our proxy system, and by 15% when using the WiFi connection.

These savings are a result of the prefetching and batching that the edge proxy performs for all embedded objects in the HTML page.

Figures 5.14 and 5.15 illustrate the energy savings for each communication method during the download period. Since the WiFi connection is high bandwidth and low latency, the potential savings from prefetching and batching are minimal. However, we save 15% of the consumed energy by limiting the number of times the power hungry WiFi card needs to be used, as illustrated in Figure 5.14. Even though it takes our proxy system several seconds longer to load the page, after one round of communication, our mobile proxy is able to serve the remainder of the browser's requests locally. This results in energy savings overall. In contrast, the proxyless approach makes 52 individual

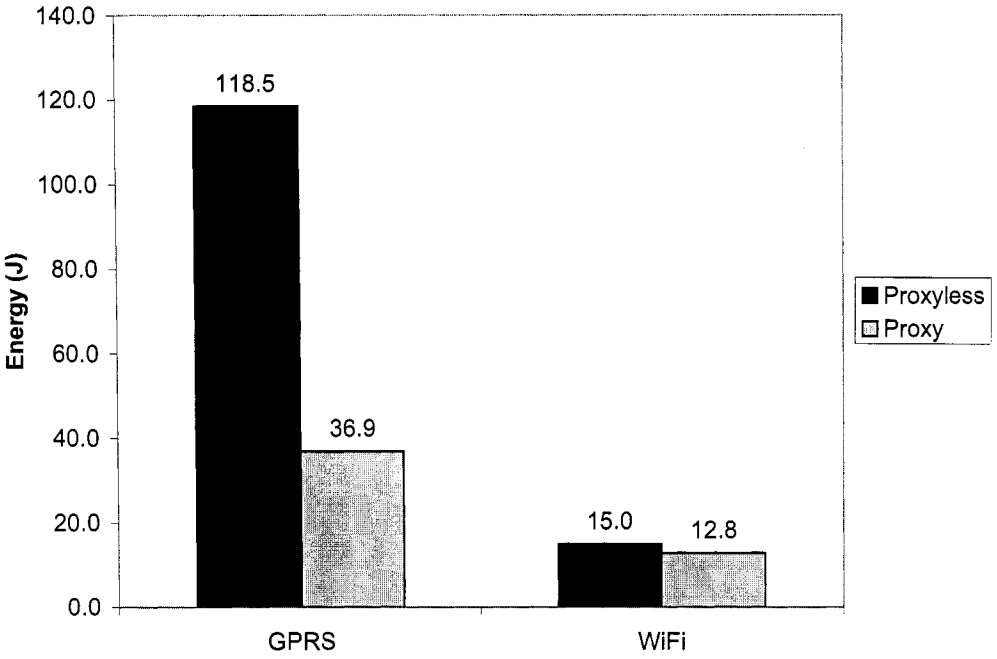

Figure 5.13: Total energy costs of downloading a cold page

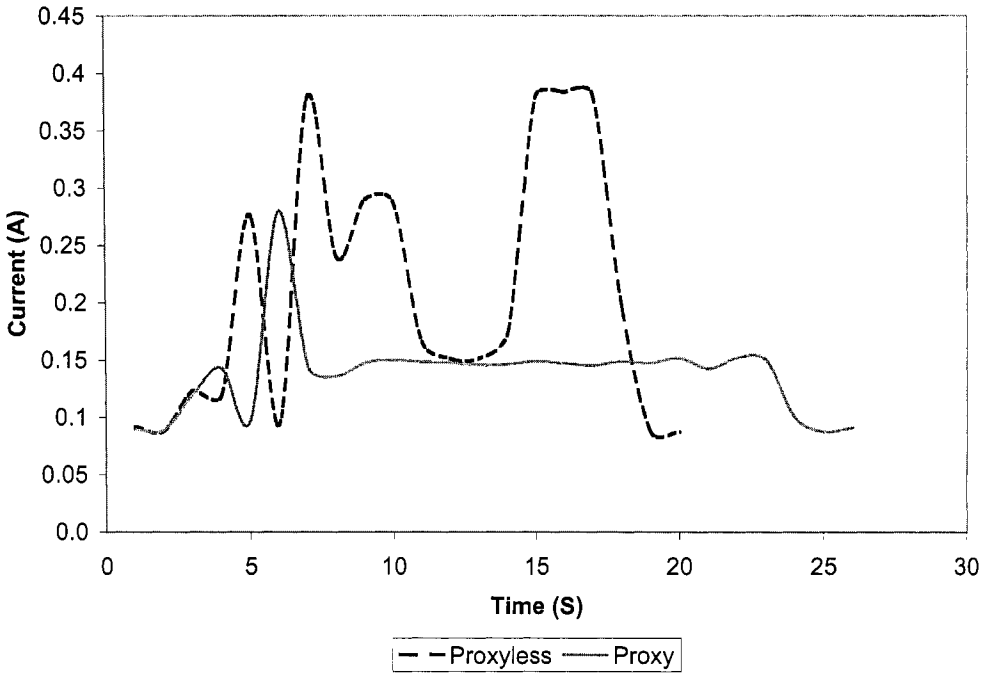

Figure 5.14: Downloading a new page over WiFi

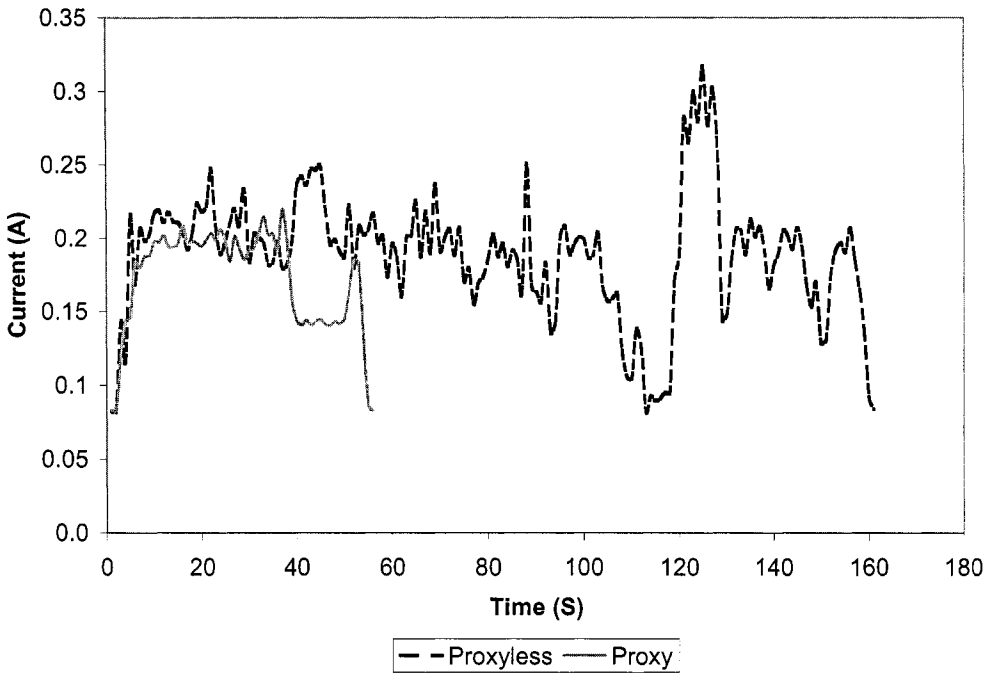

Figure 5.15: Downloading a new page over GPRS

requests over the WiFi connection.

The prefetching and batching behavior of the proxy system is much more beneficial when browsing over GPRS. As illustrated in Figure 5.15, the energy savings come from being able to load the page in nearly a third of the time when using the proxy. By downloading the web page and all its embedded files in one transfer, we are bypassing the large round trip time of the GPRS connection.

Chapter 6

Discussion

Our results have shown that our proxy-based approach achieves considerable savings in terms of both data communicated over the wireless link and battery consumption on the mobile device for both 802.11 and GPRS connections. In the GPRS scenario, the proxy is able to bypass much of the latency associated with the slow cellular link. On the other hand, the same proxy mechanism, when used over an 802.11 connection reduces the amount of time that the high energy network interface must be in use.

The push-based approaches have negligible data transfer savings, as well as negligible energy savings, compared to their poll-based counterparts. Moreover, we argue that push-based approaches using WiFi are challenging to implement in practice. First, a push-based approach assumes always-on semantics for clients, which is currently supported on cellular phones and Blackberries [43], but not over WiFi. In practice, due to power considerations, push-based communication is implemented by following a very tightly synchronized schedule for client and server. Otherwise, the client wastes power while waiting in idle mode for the server to contact it. For a push-based notification system to support WiFi devices, having an agreed upon communication schedule is not enough. The system has to also include a facility to buffer messages, as a mobile WiFi device may go offline at any time, as the user moves between hotspots. Second and more importantly,

a push-based approach requires mobile IP [37] support; the server has to have a way of contacting the client, while the mobile client's IP address is potentially changing as it connects from different access points.

The SMS based proxy is the most promising configuration. Most carriers offer free incoming SMS messages. Furthermore, receiving an SMS message has minimal energy consumption (as shown in Figure 5.7). As we have shown, this method conserves the most energy overall. Furthermore, it has the added benefit of being able to deliver content change notifications to the user, even if the user is in a disconnected state.

Finally, there is an obvious trade-off between the data freshness and the cost of maintaining it. Frequently pulling updates from the edge proxy implies high battery consumption on the mobile device. In our approach, the appropriate compromise is left up to the user. Our approach maximizes the battery savings for a particular data freshness time threshold set by the user. While we do not currently offer anything beyond the basic browser-based interface described in section 3.3, we can envision an adaptive approach where the update pull interval may be decreased or increased by the user at the press of a button. For instance, if the user anticipates an imminent period of disconnection, they may temporarily increase the update interval for their content.

Chapter 7

Related Work

Our research draws on related work in a variety of domains in mobile computing, including: proxy based systems for improved latency and energy consumption for web browsing on mobile devices, general techniques for energy savings in mobile devices, adapting content for use in a mobile environment, harnessing user profiles to improve system interaction and performance, proprietary methods for pushing content, and additional novel methods for data retrieval. In the following, we describe the related work in these areas and compare our system with the most relevant related works in each area.

7.1 Proxy-based Systems for Improving Web Browsing on Mobile Devices

Several other works, most notably WebExpress [24] and PAWP [45], have investigated using proxies to reduce energy consumption and/or perceived latency while using a mobile device to browse the internet.

WebExpress [24] uses two intercepts (proxies), one located on the mobile client and one on the server side, to significantly reduce the the cost and response time of mobile web browsing. These two proxies use many of the same, or similar, techniques we use

to reduce the amount of wireless communication. Both proxies cache recently requested graphics and HTML objects. The proxies reduce the protocol overhead by using one wireless connection for requests as opposed to creating multiple. They also reduce the size of the HTTP request header by storing some of the constant information, such as the browser's capabilities, at the server side proxy after the first request. This allows them to omit that information from subsequent requests made over the wireless link, and have it added back in by the server side proxy when it is sent to the web server. WebExpress also uses differencing to further reduce the amount of data to send. Using the currently cached item as the base object, the server side proxy sends the diff between that file and the new version. The mobile proxy can then recreate the new file by applying the diff to it's cached copy.

Although WebExpress is very similar to our work, there are several differences which have a substantial impact on the performance of the system as a whole. First of all WebExpress does not prefetch and batch embedded objects when new requests are received. This means that, unlike our system, WebExpress provides very little benefit when viewing pages for the first time. Cache updates in WebExpress are made on an individual item by item basis, as opposed to the cache wide update mechanism in our approach. So for pages with dozens of embedded objects, the device must still make dozens of requests to ensure it's cache is up to date. Most importantly WebExpress is not as user centric as our system, it does not keep client profiles, and does not update the cache based on the client's interests. As we have shown in Section 5.2, our user centric approach has a dramatic reduction on the amount of wireless communication that occurs.

The authors of PAWP [45] use a single proxy near the wireless access point to buffer WWAN web traffic to/from the device. By prefetching embedded objects and buffering requests the proxy can schedule the data to be delivered to the mobile client in alternating bursts of high and no activity. This allows the WiFi card on the device to have a more aggressive sleep pattern, which in turn conserves energy. PAWP was designed for, and

shows it's greatest benefit, in an environment where communication rates between the device and the proxy is substantially higher than between the client and the web server. As a result this solution would work well for WiFi based communication, but would show little benefit for communication using a GPRS connection.

The benefit of such a solution is that no additional software is needed on the client, as there is no local proxy. However, the network card driver would have to be updated to take advantage of this more aggressive sleep pattern.

Unlike our solution, the client is still required to make each individual request and download each individual file. The PAWP proxy does not cache any data, it only gathers it on request. Also, by not filtering out redundant data, the PAWP proxy results in a much larger amount of wireless communication than our proxy solution. However, our solution could take advantage of a more aggressive sleep pattern for the wireless interfaces to save more energy. Since our prefetching, and batch cache updates provide the same patterns of bursty high and no activity wireless traffic.

Chakravorty et al. [10] provide a comparative performance study of various techniques, at the application, session, transport and link layers, to improve web browsing latencies over GPRS. They examine both proxy based and proxy free solutions and conclude that proxy based approaches offer the greatest improvements. This is mainly due to the proxy's ability to use techniques such as extended caching, delta encoding, prefetching and batching.

Additional work [22] has attempted to improve user experience when browsing the web on a mobile device by relocating a proxies cache to follow the roaming user. In this work the mobile device uses a support station (SS) to connect to the web (you can view the support station like a cellular tower in a cellphone network). The SS contains a cache to improve performance when visiting previously viewed web sites. However, when the user is roaming, and constantly switching support stations, this cache must be built from the ground up at each new station. The authors propose using path prediction

to preemptively copy the cache to the most likely SS the client will visit next, thereby removing the need to build the cache from scratch.

7.2 Energy Conservation for Mobile Devices

Sinclair et al. [46] provide a method for conserving power by separating the data and control signals and using a wake event on the control line to wake up the device upon an incoming connection. This allows the authors to disable the power hungry WiFi card until it is needed. However, this mechanism requires specialized hardware. In contrast, in our situation we use commodity devices with no modifications and exploit client profile information in order to help conserve power.

Other work has shown show how applications can dynamically modify their behavior to conserve energy [32]. They then extend this idea to allow the operating system to monitor energy supply and demand and use this information to guide adaptation on the system to achieve a battery life of desired duration. By having the operating system monitoring the supply and demand of energy in the system as a whole, this approach can help extend the battery life of the device even when running multiple energy intensive applications. Our proxy system would work well in this scenario as the OS could adaptively adjust our update interval, as well as prune our cache based on the current and forcasted energy demands of the system.

Turducken [48] is a hierarchical power management architecture with the goal of providing always on functionality while extending the battery lifetime of the device. The target system for this work is a distributed data store, where in order to maintain consistency and execute requests, all the devices in the network must be powered on and connected to each other. As a result, power saving mechanisms that require the device to periodically sleep or suspend itself are not good alternatives. Instead the authors propose integrating several mobile computing platforms that operate at different power levels into

a single multi tiered device (for example a laptop, PDA and sensor in one package). This allows the device to operate at the power level of any of it's tiers depending on the work load. The device could function as a laptop for heavy tasks, as a PDA for light tasks, or as a sensor when waiting for incoming work.

Our centralization of update processing on the proxy is similar in spirit to the cyber-foraging approach [4] of offloading computation and storage from mobile devices to available resource-rich nodes on the Internet in order to save power and money.

7.3 Content Adaptation for Mobile Devices

Mohomed et al. [31] automatically adapt images for display on mobile devices based on the usage semantics of that content. They allow users who are unsatisfied with the current adaptation to take control of the adaptation process to make changes accordingly. These changes are recorded and then used in future adaptation decisions. Their system also allows the user to specify how much effort they are willing to put into the adaptation process in order to save bandwidth. If the user prefers being involved in the adaption process, then the system will conserve bandwidth by serving images at a lower fidelity, at the risk of having the user correct its decision. However if the user doesn't want to be involved, than the images will be served at a higher fidelity, increasing the bandwidth consumption but decreasing the chance the user will have to correct the system's decision. This work could be combined with ours to further reduce bandwidth consumption. The adaptation process could be applied to the cached items and their updates. The fidelity of the objects could be partially based on the number of times that the object has been used. This would reduce the penalty of frequently updating a cached object that is rarely accessed.

Subsequent work [12] has proposed a new browsing convention to facilitate navigating and reading web pages, that were designed for desktop computers, on devices with much

smaller screens, like PDAs. The document is divided into a two level hierarchy, where at the top level the page is displayed as a thumbnail, to allow for easy navigation to any part of the page. The second level is a set of sub pages that display the detailed information. They use a page adaptation technique which analyzes the structure of an existing page and splits it into small and logically related units that fit the display of the mobile device. The user can then use the thumbnail view of the page to jump from unit to unit, removing the need for the user to scroll over the page looking for information. The authors' idea of logical units would also benefit our system. For example, once the document had been broken down into units, our system could keep the thumbnail and the unit that contained information pertinent to the user up to date. This would greatly reduce the amount of data we send during updates, since we'd only be sending the portion of the page that contains the relevant information. All other logical units could be fetched on demand as they're accessed from the thumbnail.

Many schemes have investigated communication savings for web content serving, such as, through server directed transcoding [29], and optimistic deltas [5]. Our scheme is orthogonal to these techniques and could be used in conjunction with them for further data savings. Our approach determines when to push modifications, while the optimistic delta approach and the transcoding approaches focus on sending smaller updates (e.g., optimistic deltas sends just the modified portions of the page instead of the whole page).

7.4 User Profiles

Our work builds on the concepts of user profiles and data recharging [2, 13] that have been recently proposed to enable disconnected operation of mobile clients. However, existing research in this area focuses on user profile language specifications [2, 13] and on algorithms to be run by the service provider to optimize the creation and maintenance of super-profiles that combine the preferences of many users [13]. Similarly, most underly-

ing algorithms in publish/subscribe distributed systems [18] investigate efficient filtering when scaling a system of publishers and distributed brokers to supporting millions of subscriptions and of filtering hundreds of new events per second. Such systems are typically using evaluations based on simulation to investigate scaling with the number of clients.

While our work shares similarities with these systems, we adopt a user-centric approach and evaluation methodology. We avoid introducing new languages or complex interfaces that may prevent wide acceptance. Instead, we concentrate on providing the maximum benefits to the user in terms of cost, battery life and convenience through a profile-driven infrastructure with seamless integration into already familiar applications and environments.

Current search engines, when given the same query, return the same results regardless of who submitted the query. Sugiyama et al. [50] claim that search results should be adapted to the information needs of each user. For example one user searching for "java" may be interested in the programming language, while the next user may simply be looking for information on coffee. As a result the authors propose several methods for adapting search results based on adaptive user profiles. These profiles are created without any user effort by using modified collaborative filtering and detailed analysis of the user's browsing history. It would be very interesting if we could use this adaptive user profile idea to provide our monitoring functionality without having the user specifically tell us which information they are interested in. However, this is a difficult problem, as viewing a web page does not indicate which information within that page the user is interested in.

The PROSKIN project [19] is another research project that is attempting to use profiling to improve user experience when interacting with applications. The authors are attempting to identify and group users based on their distinguishing interactive behaviors so that they can adapt their user interface to provide a more personally optimized

experience.

7.5 Push Based Technologies

Push-based solutions for specialized types of information such as Bell's e-mail notifications [6], Research in Motion's Blackberry [43], location-based services for taxi availability [60] or food/entertainment advertisements [8] already exist or are being proposed. However, most types of information that are being currently pushed are simple text messages such as e-mail notifications, instead of the content the user is actively browsing.

A scheme for cutting down on network traffic by pushing content to the users is also presented by Baker et al. [3]. The main difference in their approach, however, is that the content itself is not pushed transparently to the user, only a message containing the URL of the new content is. It is up to the user to determine whether or not to fetch that content.

A variant of a push-based scheme for data refresh similar to our own appears in related work on broadcast disks [1, 57]. Broadcast disks are indexing algorithms for data that is being broadcasted over a wireless channel. Mobile nodes tune in to listen to the index being broadcasted. The index, which is assumed to be much smaller than the full data broadcast, identifies the time when a given data item will be transmitted. A node then knows when to wake up to catch the transmission of an item of interest. While these systems have similar goals to our own, they do not target data refresh for dynamic content browsing. Furthermore, in their case, the server maintains no knowledge of the content the user already has.

7.6 Data Dissemination

Google Short Message Service (SMS) [20] is currently one of the best ways to retrieve data while mobile. The fact that Google SMS uses SMS for the request and delivery of the data

also means that the client does not require a data connection to retrieve information, any cellphone, capable of sending and receiving text messages, will do. Considering the simple interface that SMS provides, Google SMS is a fairly robust service, capable of delivering information in a variety of categories (phone directory, driving directions, movie show times, weather, stock quotes, currency converter, and many more). The service is simple to use. The user sends a text message to Google, containing the question to answer ("weather dallas tx" or "5 usd in yen") and shortly thereafter Google responds with the answer ("Weather: Dallas, TX 88F, Scattered Clouds" or "5 U.S. dollars = 557.499992 Japanese yen").

There are two main problems with Google SMS. The first problem is the monetary cost. Although the service is provided free of charge from Google, and receiving SMS messages is free with most service providers, sending SMS messages costs money. Since you must send an SMS for each data item you want to retrieve, each time you want an update, monitoring the value of a stock quote, the value of a currency, and the current temperature throughout the day could become an expensive proposition. Contrast this with our solution where, once you've selected to have the the data monitored in your browser, you'll receive a SMS message each time the value changes, without having to spend the time or money to request it.

The second problem is that, although Google SMS is a robust service with a large number of categories, you are still limited to the data that they provide you. If you want the current value of that auction you've been watching for the past couple of days than you're out of luck. Our solution provides nearly the same functionality as Google SMS while providing the mechanism to use it on any data available on the internet.

RSS [56, 7] is a system with which web servers can push new content directly to users over the internet. This system involves the user subscribing to that particular web server's content and then each change appears on the user's desktop. However, the underlying principle of this system is that the client regularly polls the site automatically. This

gives the illusion of a push based system, but underneath the system is an automated web polling service. The benefit of RSS comes from being able to view all the data in one spot using a feed reader. These feed readers check each subscription for new data and display the new information from each source on one page. The problem with this type of service is that you have to rely on the web master of the site you are interested in to provide an RSS feed. Even if the site provides a RSS feed, it may not cover the information you are interested in. For example an online store may list new products in it's RSS feed, but does not list price.

Chapter 8

Conclusions

We introduce a novel approach to transparent, automatic data refresh for mobile devices. Our approach is centered around a general purpose mechanism for letting the user specify her interest in changes to specific parts of pages. We avoid introducing new languages or complex interfaces that may prevent wide acceptance. Instead, the user loads her favorite pages on her mobile device browser and highlights areas of interest in those pages using the regular browser's cursor. We offload the detection of updates to content that matches the user's interest, onto a fully-connected edge proxy. Subsequently, either while the client is actively browsing or while attending to everyday activities of travel, shopping, work and play, the mobile device performs automatic data refresh transparently to the user.

Our approach is fully implemented using both WiFi and GPRS communication on an actual mobile device and evaluated on real world data traces. Our results show that our general purpose proxy system saves data transfers to and from the mobile device by an order of magnitude and battery consumption by up to a factor of 4.5. These savings are due to the fact that, typically, there are frequent changes to parts of dynamic content web pages that the user is not interested in, such as the time of day or an ad banner. In addition, many changes in the n-th decimal of numerical values can be typically ignored. We have shown that, a push-based approach provides minimal gains over a poll-based

approach. Additionally, we have shown that by using the existing SMS infrastructure to deliver notifications on dynamic content changes, we can offer an energy efficient and user friendly way to keep the clients up to date with their content of interest.

Bibliography

[1] Swarup Acharya, Rafael Alonso, Michael Franklin, and Stanley Zdonik. Broadcast disks: data management for asymmetric communication environments. pages 199–210, 1995.

[2] R. Agrawal and E. L. Wimmers. A framework for expressing and combining preferences. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, August 2000.

[3] Shaun Baker. Active internet services: Pushing content to the people. In *whitepaper for SIP development group of Siemens Switzerland*, 2000.

[4] R. Balan, J. Flinn, M. Satyanarayanan, S. Sin, and H. Yang. The case for cyber foraging. 2002.

[5] Gaurav Banga, Fred Douglis, and Michael Rabinovich. Optimistic Deltas for WWW Latency Reduction. In *Proceedings of the 1997 USENIX Technical Conference*, January 1997.

[6] Bell Canada. Blackberry 7750 wireless handheld. http://www.bell.ca/shop/en_CA_BC/Sme.Sol.Wireless.Solutions.BlackBerry.page.

[7] Mel Blackman. Pushing web content with really simple syndication. *e-Pro Magazine*, 2001.

[8] I. Burcea and H.A. Jacobsen. L-ToPSS - Push-oriented Location-based Services. In *4th VLDB Workshop on Technologies for E-Services (TES'03)*, 2003.

[9] R. Chakravorty and I. Pratt. Practical Experience with HTTP and TCP over GPRS, 2002.

[10] Rajiv Chakravorty, Suman Banerjee, Pablo Rodriguez, Julian Chesterfield, and Ian Pratt. Performance optimizations for wireless wide-area networks: comparative study and experimental evaluation. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 159–173, New York, NY, USA, 2004. ACM Press.

[11] Surendar Chandra, Ashish Gehani, Carla Schlatter Ellis, and Amin Vahdat. Transcoding characteristics of web images. In Martin Kienzle and Wu chi Feng, editors, *Multimedia Computing and Networking (MMCN'01)*, volume 4312, pages 135–149, San Jose, CA, January 2001. SPIE - The International Society of Optical Engineering.

[12] Yu Chen, Wei-Ying Ma, and Hong-Jiang Zhang. Detecting web page structure for adaptive viewing on small form factor devices. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 225–233, New York, NY, USA, 2003. ACM Press.

[13] Mitch Cherniack, Eduardo F. Galvez, Michael J. Franklin, and Stan Zdonik. Profile-driven cache management. In *International Conference on Data Engineering (ICDE)*, 2003.

[14] Cingular. Wireless rate plans. http://onlinestorez.cingular.com/cell-phone-service/wireless-phone-plans/.

[15] CNN. Toronto weather. http://weather.cnn.com/weather/forecast.jsp?locCode=YYZ.

[16] CNN Money. Origami looks paper-thin. http://money.cnn.com/2006/03/09/ technology/business2_origami/.

[17] eBay. Online auctions. http://www.ebay.ca/.

[18] Francoise Fabret, H.Arno Jacobsen, Francois Llirbat, Joao Pereira, Kenneth Ross, and Dennis Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 115–126, 2001.

[19] Nick Fine and Willem-Paul Brinkman. Informing intelligent environments: creating profiled user interfaces. In *EUSAI '04: Proceedings of the 2nd European Union symposium on Ambient intelligence*, pages 15–18, New York, NY, USA, 2004. ACM Press.

[20] Google. Google short message service us. http://www.google.com/sms/.

[21] GSM World. Today's GSM platform. http://www.gsmworld.com/technology/gsm.shtml.

[22] Stathes Hadjiefthymiades and Lazaros Merakos. Proxies + path prediction: improving web service provision in wireless-mobile communications. *Mobile Networks and Applications*, 8(4):389–399, 2003.

[23] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini. Application transformations for energy and performance aware device management. 2002.

[24] Barron C. Housel and David B. Lindquist. Webexpress: a system for optimizing web browsing in a wireless environment. In *MobiCom '96: Proceedings of the 2nd annual international conference on Mobile computing and networking*, pages 108–116, 1996.

[25] IntelliProg Inc. Webbrowersce. http://www.intelliprog.com/products/prod03.html.

[26] John Ioannidis, Dan Duchamp, and Gerald Q. Maguire Jr. IP-based protocols for mobile internetworking. Technical report, Zürich, Suisse, 1991.

[27] John Ioannidis and Gerald Q. Maguire Jr. The design and implementation of a mobile internetworking architecture. In *USENIX Winter*, pages 489–502, 1993.

[28] Jamster. Text alerts. http://www.jamster.com/jcw/specials/text_alerts.do.

[29] Bjorn Knutsson, Honghui Lu, Jeffrey Mogul, and Bryan Hopkins. Architecture and performance of server-directed transcoding. *ACM Transactions on Internet Technology*, 3(4):392 – 424, November 2003.

[30] Robin Kravets and P. Krishnan. Power management techniques for mobile communication. In *MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 157–168, New York, NY, USA, 1998. ACM Press.

[31] Iqbal Mohomed, Jim Chengming Cai, Sina Chavoshi, and Eyal de Lara. URICA: Usage-aware interactive content adaptation for mobile devices. In *Proceedings of the 1st EuroSys Conference*, April 2006.

[32] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile application-aware adaptation for mobility. In *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 276–287, New York, NY, USA, 1997. ACM Press.

[33] O2. O2 data tariffs. http://www.o2.co.uk/business/tariffs/datatariffs/.

[34] A. W. O'Neill and G. Tsirtsis. Edge mobility architecture - routeing and hand-off. *BT Technology Journal*, 19(1):114–126, 2001.

[35] OpenNetCF.org. Smart device framework 2.0. http://opennetcf.org/CategoryView.aspx?category=Home.

[36] Paypal. Paypal money transfers. https://www.paypal.com/.

[37] C Perkins. IP Mobility Support. RFC 2002, October 1996. ftp://ftp.isi.edu/in-notes/rfc2002.txt.

[38] Thomas Phan, George Zorpas, and Rajive Bagrodia. Middleware support for reconciling client updates and data transcoding. In *Proceedings of International Conference on Mobile Systems, Applications, and Services*, June 2004.

[39] Thomas A. Phelps and Robert Wilensky. Robust intra-document locations. In *Proceedings of the 9th international World Wide Web conference on Computer networks*, pages 105–118, Amsterdam, The Netherlands, The Netherlands, 2000. North-Holland Publishing Co.

[40] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 89–102, New York, NY, USA, 2001. ACM Press.

[41] Johan Pouwelse, Koen Langendoen, and Henk Sips. Dynamic voltage scaling on a low-power microprocessor. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 251–259, New York, NY, USA, 2001. ACM Press.

[42] Ramachandran Ramjee, Kannan Varadhan, Luca Salgarelli, Sandra R. Thuel, Shie-Yuan Wang, and Thomas La Porta. Hawaii: a domain-based approach for supporting mobility in wide-area wireless networks. *IEEE/ACM Trans. Netw.*, 10(3):396–410, 2002.

[43] Research in Motion. Blackberry. http://www.blackberry.com.

[44] Rogers. Rogers wireless. http://www.shoprogers.com/store/wireless/.

[45] Marcel C. Rosu, C. Michael Olsen, Chandra Narayanaswami, and Lu Luo. Pawp: A power aware web proxy for wireless lan clients. In *WMCSA '04: Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'04)*, pages 206–215, 2004.

[46] E. Shih, P. Bahl, , and M. Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom)*, 2002.

[47] Tajana Simunic, Luca Benini, Andrea Acquaviva, Peter Glynn, and Giovanni De Micheli. Dynamic voltage scaling and power management for portable systems. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 524–529, New York, NY, USA, 2001. ACM Press.

[48] Jacob Sorber, Nilanjan Banerjee, Mark D. Corner, and Sami Rollins. Turducken: hierarchical power management for mobile devices. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 261–274, New York, NY, USA, 2005. ACM Press.

[49] Maarten Struys. Receiving sms messages inside a managed application. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetcomp/html/ReceivingSMSMessages.asp.

[50] Kazunari Sugiyama, Kenji Hatano, and Masatoshi Yoshikawa. Adaptive web search based on user profile constructed without any effort from users. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 675–684, New York, NY, USA, 2004. ACM Press.

[51] T-Mobile. T-Mobile cellular service. http://www.t-mobile.com.

[52] Text It. SMS text facts. http://www.text.it/mediacentre/facts_figures.cfm.

[53] Amin Vahdat, Alvin Lebeck, and Carla Schlatter Ellis. Every joule is precious: the case for revisiting operating system design for energy efficiency. In *EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 31–36, New York, NY, USA, 2000. ACM Press.

[54] Andrs G. Valk;. Cellular IP: a new approach to internet host mobility. *SIGCOMM Comput. Commun. Rev.*, 29(1):50–65, 1999.

[55] XE.com. Currency site. http://www.xe.com/.

[56] XML.com. RSS description. http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html.

[57] A.J. Xu, W. Lee, and X. Tang. Exponential index: A parameterized distributed indexing scheme for data on air. 2004.

[58] Yahoo! Yahoo! finance. http://finance.yahoo.com/.

[59] Wanghong Yuan and Klara Nahrstedt. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 149–163, New York, NY, USA, 2003. ACM Press.

[60] Zingo Taxi. Location based services. http://www.springwise.com/newbusinessideas/2003/09/zingo_taxi.html.