

Optimized Wireless Web Browsing Using Mobile Agents*

Peter Graham, Scott Walkty and Randal Peters

Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada R3T 2N2
{pgraham,swalkty,randal}@cs.umanitoba.ca

Ken Barker

Department of Computer Science
University of Calgary
Calgary, Alberta, Canada T2N 1N4
barker@cpsc.ucalgary.ca

Abstract

The use of wireless data communications is increasing rapidly despite limited bandwidth and reliability. Algorithms have been developed to optimize the available bandwidth (e.g. through compression) but the widespread adoption of wireless data communications has been slowed by the need to agree on optimization standards and to implement and deploy code to implement such standards.

In this paper we propose a framework that uses mobile agents to dynamically deploy application-specific, bandwidth optimization code. No agreement is required on the set of available optimizations only that base stations support a specific mobile agent execution environment. If a given base station does not support mobile agents, it can still be used but in an unoptimized fashion.

In our system, the mobile unit “downloads” an agent to its base station that dynamically loads optimization code as needed over the fixed network. The optimizations loaded are determined by the mobile unit (via the agent) and correspond to optimizations the mobile unit supports. This framework supports optimization specialization at very low cost since little code must be downloaded from the mobile unit. A prototype implementation of the framework using the Java-based Aglets mobile agent system is presented.

Keywords Bandwidth Optimization, Mobile Agents, Mobile Computing, Wireless Communications

1. Introduction

Optimizing the use of low-bandwidth wireless links will be fundamental to the success of mobile computing. While work has been done to address this problem, solutions are still largely piecemeal. Further, solutions that work in one case may provide little or no benefit in others. Part of the

problem is the difficulty of deploying optimization code into the network. Creating new protocols and getting them adopted takes a long time and to make matters worse optimizations for new applications are being produced at a far faster rate than protocols are being developed.

To address the slow deployment problem, at least two techniques have been proposed (active networks [3] and mobile agents [6]) that allow new code to be dynamically loaded into the network to implement newly developed “protocols”. In the case of wireless web browsing, such techniques can be used to deploy application and environment specific code to optimize use of the wireless link.

In this paper we present a framework for using mobile agents to optimize wireless communications. We describe how this framework can be used to improve the performance of wireless web browsing and also discuss two prototype implementations we have built. The framework we describe solves the slow deployment problem, allows for great flexibility in the optimizations performed, avoids potential protocol mismatches (between mobile units and the base stations they connect to), and is completely transparent to existing browsers, servers, and network protocols.

1.1. Organization

The rest of this paper is organized as follows. In Section 2 we review related work. Section 3 discusses the problem and our assumptions. Our solution is described in Section 4 and we discuss our prototype implementations in Section 5. We present our conclusions and discuss directions for future research in Section 6.

2. Related Work

2.1. Mobile Web Browsing Architectures

Wireless access to the World Wide Web is an active research area. Most systems are based either on proxy servers

*This research was supported by Natural Sciences and Engineering Research Council grants OGP-0194227, OGP-173259 & RGP-0105566.

(where a proxy machine transforms data prior to delivery to the mobile unit) or on the “client/intercept/server” architecture (where transmissions between web clients and servers are intercepted and transformed by processes on both the fixed and mobile units. Mowgli, WebExpress, and Mobiscape use this architecture:

Mowgli [11] was an early mobile web browser systems to use the client/intercept/server architecture. It employs optimizations including prefetching of inline images, caching data on the mobile and fixed units, using persistent sockets to reduce connection setup and teardown times, and background fetching of pages (asynchronous operation).

WebExpress [4, 7] was designed to support applications characterized by repetitive and predictable responses. For example, submitting a request for a stock quote will return a standard stock quote page with the desired information. Although the specific information in the quote changes with each request, the structure of the page remains the same. WebExpress stores “page templates” on the fixed and mobile proxies and transmits only the differences when new pages are requested. WebExpress also employs caching (on the fixed and mobile units), virtual sockets (persistent sockets used for many web page requests), and header reduction.

Mobiscape [1] focuses on flexible caching policies that enable users to specify what information should always be available in the cache by way of a user profile. Mobiscape also compresses HTML pages before they are transmitted across the wireless link.

Other systems have also been implemented that do not use the client/intercept/server architecture. For example, Mowser [9] finds end systems that are capable of providing different content to resource limited mobile devices and performs content negotiation with those end systems to request data in a more suitable format (e.g. lower resolution images to save bandwidth). Mowser employs a wired side proxy that optimizes data before sending it to the web browser.

Other projects, although not specifically focused on mobile web browsing, are also relevant. Zenel [14] designed a dynamic proxy system based on filters. Filters are executable programs that can be downloaded from the mobile unit (or other locations) and executed on the fixed unit to offer services that overcome the limitations of wireless connections. An HTTP compression filter was implemented as part of the test suite for the system. The HTTP filter essentially implements a dynamic form of the client/intercept/server architecture.

The Dataman wireless web project [8] uses a wired side proxy to manage HTTP connections by prefetching pages residing on the same web server as previously requested pages. The BARWAN project [2] defines a general proxy model for use with mobile applications. A proxy conforming to their model is capable of transforming the data before sending it to the web browser in a number of ways including

lossy, type-specific compression and aggregating data from several sources and organizing it for the mobile user. Both Dataman and BARWAN also provide other optimizations.

2.2. Mobile Agents for Mobile Computing

The use of mobile agents in mobile environments has gained significant recent attention particularly with respect to support for disconnected operation.

The Magenta system [12] was designed for use in weakly connected environments. In addition to the usual mobile agent system functionality, Magenta offers a selection of mobile specific features. Magenta can adjust to the dynamic appearance and disappearance of execution environments and supports remote instantiation of mobile agents. A mobile client that is not running a copy of the Magenta agent system can launch a mobile agent on a remote site that will execute on its behalf. Magenta also provides tolerance of execution environment failures and a directory service for agents. The directory service allows mobile units to track the progress of agents they have dispatched following a period of disconnection when the mobile units reconnect.

Agent TCL [5] offers several network sensing tools (to determine connectivity and available bandwidth) that can be used to adapt agent behavior. It also implements a docking system for disconnected operation. Each mobile unit has an associated docking station on the fixed network that stores agents destined for the mobile unit while it is disconnected and then forwards the agents to the mobile unit when it reconnects. The docking station is responsible for detecting connectivity and is transparent to the mobile agent.

3. The Problem and Environment

3.1. Issues in Optimizing Wireless Data Access

The fundamental issue in providing wireless data access (for web browsing or other applications) is management of the limited bandwidth available for transferring data¹ which can have a serious impact on the web applications that can be supported. This problem is addressed by limiting the amount of information that is sent over the wireless link and this can be done in different ways. Indirect techniques such as mobile-unit caching seek to decrease the need to transfer data but incur a high cost when used in devices with constrained memory capacity. Direct techniques seek to *transform* the data that is transmitted so that fewer bytes are sent.

Direct techniques can be applied at different levels. For example, batching small messages and sending them in a single packet can save protocol overhead. Changes to the

¹For example, Cellular Digital Packet Data (CDPD) uses the existing cellular phone network to support data communications but offers a peak bandwidth of only 19.2Kbps.

applications themselves are also possible and may result in greater savings. For example, a custom web browsing environment that automatically selects text-only pages would consume far less bandwidth than a conventional browser. Unfortunately, such custom applications are costly and difficult to develop and get accepted and they often provide highly inferior functionality. Application transparent techniques (e.g on-the-fly compression) offer reasonable bandwidth savings without requiring new browser or server software. They are the focus of this paper.

Other issues such as security, power consumption, and management of disconnection also affect the quality of a wireless system but are not directly addressed in this paper. It is worth noting, however, that some of these issues can also be addressed within the framework presented in this paper. For example, providing encryption in addition to, or in place of, compression is straightforward. Also, some low-level techniques (such as batching messages) can easily be incorporated into our framework and these can have a positive side-effect of lowering power consumption in addition to decreasing bandwidth usage.

3.2. Assumptions

In this paper we focus on the details of our optimization framework. As such, little will be said about the nature of the optimizations performed and nothing will be said about the necessary software required to support the mobile computing environment itself (e.g. DHCP, MOBILE-IP, SLP, etc.). At all times, we assume transmission using IP networks (including the wireless “segments”).

4. An Agent-Based Solution

We have designed a framework for optimizing communications between mobile units and the fixed-network base stations they are linked to using a client/intercept/server architecture. The protocol uses mobile agent technology to make it easy to deploy custom optimization code where and when it is needed. While the work presented here is in the context of mobile web browsing, the framework is generally extensible to other applications.

4.1. The Framework

An optimized communication between a mobile unit and its fixed-network base station (the “fixed unit”) involves two components, the MU_{opt} and FU_{opt} as shown in Figure 1. These components replace the conventional intercept proxies to provide dynamic optimization in a fashion similar to Zenel [14]. For example, if simple run-length data encoding is required between the mobile and fixed units then the MU_{opt} and FU_{opt} will implement the necessary encoding

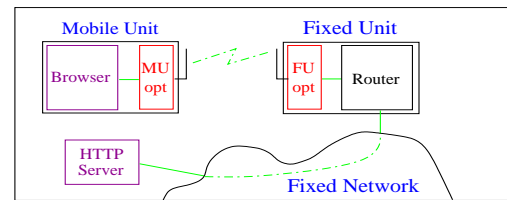


Figure 1. An Optimized Communication

and decoding. Generally, they may, of course, perform any useful data transformation(s).

There are normally many different types of data sent between the fixed and mobile units and it makes sense to be able to apply different optimizations on different types of data. Further, over time, a mobile unit may experience variations in the wireless link quality and it would be desirable to alter the optimizations performed accordingly. For example, on a critical communication, an optimization might add data redundancy when the link quality is poor and deal with decreased performance rather than a failed application.

Each mobile unit supports a collection of optimizations (loaded under the assumption they will be useful) and each such optimization is uniquely identified. The mobile unit constructs an “optimization table” indexed by the current link quality and the type of data to be transmitted. Each entry in the table contains the unique identifier of the desired optimization routine to be used in transmission (refer to Figure 2). Whenever data is transmitted or received, the optimization routine indicated in the table is used by the mobile and fixed units to transform the data.

For this approach to work, the optimization routine used by the mobile unit in each case must be the same as that used by the fixed unit. This need for agreement slows the deployment of new optimization techniques and introduces the potential for mismatches and failed communication in conventional systems. To avoid this problem, we encapsulate the optimization table in a mobile agent which the mobile unit downloads to each fixed unit it connects to.

The mobile unit defines the optimization table based on the optimization routines it supports. These optimization routines, however, are not necessarily available at the fixed units. Once downloaded to the fixed unit, the agent is therefore responsible for ensuring that the required optimization routines are loaded into the fixed unit as required. This is done by using optimization location and download services that run on the fixed network.

Using agents this way offers several benefits:

- Any optimization routine may be used at any fixed unit without the need to pre-deploy it.
- There is no possibility for mismatches between the

		Link Quality			
		Good	Fair	Poor	Disconnected
D a t a T y p e	Text	Null	Run Length	LZW	hold
	Image	Null	Comp	Drop	hold
	Video	Grey Scale	Re-Quant	Drop	hold
	Audio	Null	Drop	Drop	hold
	○ ○ ○				

Unique Optimization Identifier

Figure 2. Optimization Table

fixed and mobile units.

- When a fixed unit fails to support the required agent execution environment communication may continue but will be unoptimized.
- No changes are required to browsers, servers, or the protocols that they use to communicate.
- Flexibility is enhanced since different optimizations and tables can be used based on application, user, and environment characteristics.

4.2. Locating and Loading Optimization Code

When the mobile agent (running on the fixed unit) must acquire an optimization routine, it does so using the fixed network in two steps: locating a server for the routine, and loading the code from that server. Fixed units typically service multiple mobile units and it is likely that the mobile units will want to apply some of the same optimizations. To avoid redundancy and to decrease the average latency of code loading, each fixed unit caches optimization routines that are in use or that have been recently used.

Service location first checks the fixed unit cache to see if the required optimization routine is resident. If it is, that copy is used. If not, then a distributed location service is queried at a well known network address.²

The location service does not return the optimization code but instead will return a list of candidate servers of the code. The agent is then responsible for selecting an appropriate (nearby) server and initiating a transfer of the code. This decoupling of the location and provision of the optimization code enhances the scalability of the system.

²Such a service can be replicated for efficiency as long as a consistency maintenance protocol is enforced. Since new optimization routines will be added infrequently and since this is the only time when consistency maintenance must be applied, the required protocol will be simple.

When new optimization routines are to be added, a unique identifier must be generated and then the corresponding code must be associated with the identifier. This process is overseen by the optimization location server which assigns the identifier thereby reserving it for the new optimization routine and then selects one or more optimization servers to which it sends the code (with the newly assigned identifier). Optimization code is read-only and thus may be freely replicated between servers. A replication scheme based on usage would allow replicas of optimization routines to migrate to new optimization servers when mobile units in the server's vicinity begin using the code.

4.3. Applying Optimizations

Once an optimization routine is loaded at a fixed unit then that routine may be used for communication with any mobile unit that requests it. Each optimization routine corresponds to the component labelled "FU Opt" in Figure 1. The process of locating and (if necessary) loading a routine results in an entry in the optimization table for a particular mobile unit being set to refer to the loaded code.³ Optimizations may be applied to data transfer in both directions. Thus, when an agent running on the fixed unit receives a message requesting a particular document, that message is processed using the routine specified in the optimization table. This permits, for example, HTTP requests to be encoded. Once processed, an HTTP request is then routed (by the Router in Figure 1) into the fixed network for service. Eventually a response is returned to the fixed unit. This response is also encoded using the optimization routine identified in the optimization table and sent to the mobile unit. In this way communication between the mobile and fixed units may be optimized in *both* directions resulting in greater bandwidth savings.

5. Prototype Implementations

The initial prototype implementation [13] was designed to show that the use of mobile agents in mobile browsing applications was feasible. It provided optimizations for common web traffic (including image re-quantification, audio and video dropping, and data compression using dictionary techniques) and focused on optimizing data flowing from the fixed unit to the mobile unit. The prototype was IP based and used the Java-based Aglets [10] mobile agent system.

A number of limitations in the first prototype were identified including a lack of focus on downstream (mobile unit to fixed unit) optimization. While upstream web traffic does greatly exceed downstream traffic, the use of the mobile agent approach introduces new downstream traffic for

³Because changes in link quality are often frequent and to improve efficiency, entire rows of the table are actually loaded as a unit.

downloading the agent. In the initial prototype, each downloaded agent encapsulated all optimization code that might be required at the fixed unit. For complex optimizations (e.g. re-quantizing an image), the corresponding code is large so agent download represents a significant bandwidth overhead. The result is high latency in initiating browsing when a mobile unit connects to a fixed unit. This problem is compounded when the user is mobile and the fixed unit connected to changes frequently.

Despite its limitations, the first prototype did illustrate the feasibility of the technique. The system was deployed in an environment using 2.4GHz WaveLAN wireless NICs operating at a relatively fast, 2Mbps. Web queries (to pages including text, images, audio and video, java code, frames and forms) were posed from each mobile unit under various connection situations and the system performed well.

The success of the first prototype lead us to develop a second that includes several enhancements. This new prototype is now partially complete. It currently downloads an aglet which locates and loads required optimization code dynamically (as described earlier in the paper). The implementation, however, currently has no location service, uses FTP to provide the code requested by aglets and implements only a trivial cache for optimization code at each fixed unit.

6. Conclusions and Future Work

In this paper we have presented a framework that uses mobile agents to allow for the dynamic optimization of the use of a wireless link. This approach avoids the problems commonly associated with creating and deploying standardised protocols which include delays in deployment and lack of flexibility. We also described two proof of concept implementations we have constructed to illustrate the feasibility of our techniques. The second implementation improves on the first by dramatically decreasing the size of the agent that must be downloaded and is the one discussed in the paper.

Our future research will include adding support for the automatic transfer of agents (and their loaded optimization routines) from one base station to the next during handoff. This will save the cost of downloading the agent code at the new fixed unit, will ensure that optimization routines that are being actively used will be available at the new fixed unit and will permit the agent to maintain any state information that may be valuable. We will also focus on extending the system beyond the realm of web browsing alone (to support truly application specific optimization), the use of proprietary wireless protocols rather than IP, and the automatic placement of optimization code on servers at locations near where they are needed.

7. Acknowledgements

Quat Do has contributed significantly to the initial implementation of the second prototype system.

References

- [1] C. Baquero, V. Fonte, F. Moura, and R. Oliveira. MobiScape: WWW Browsing under Disconnected and Semi-Connected Operation. In *Proc. Portuguese WWW Conference*, 1995.
- [2] E. Brewer, R. H. Katz, E. Amir, H. Balakrishnan, Y. Chawathe, A. Fox, S. D. Gribble, T. Hodes, G. Nguyen, V. N. Padmanabhan, M. Stemm, S. Seshan, and T. Henderson. A Network Architecture for Heterogeneous Mobile Computing. *IEEE Personal Communications Magazine*, 5(5):8–24, 1998.
- [3] K. Calvert, S. Bhattacharjee, E. Zegura, and J. Sterbenz. Directions in Active Networks. *IEEE Communications*, 1998.
- [4] H. Chang, C. Tait, N. Cohen, M. Shapiro, and S. Mastrianni. Web Browsing in a Wireless Environment: Disconnected and Asynchronous Operation in ARTour Web Express. In *Proc. MOBICOM'97*, pages 260–269, 1997.
- [5] R. Gray, D. Kotz, S. Nog, D. Rus, and G. Cybenko. Mobile Agents for Mobile Computing. Technical Report PCS-TR96-285, Dept. of Computer Science, Dartmouth College, 1996.
- [6] R. S. Gray, G. Cybenko, D. Kotz, and D. Rus. Mobile agents: Motivations and state of the art. In J. Bradshaw, editor, *Handbook of Agent Technology*. AAAI/MIT Press, 2000.
- [7] B. Housel and D. Lindquist. WebExpress: A System for Optimizing Web Browsing in a Wireless Environment. In *Proc. MOBICOM'96*, pages 108–116, 1996.
- [8] T. Imielinski. Mobile Computing: the DataMan Project Perspective. *Mobile Networks and Applications*, 1(4):359–369, 1996.
- [9] A. Joshi. On Proxy Agents, Mobility and Web Access. Technical Report 99-02, CSEE Department, University of Maryland, Baltimore County, 1999.
- [10] D. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison Wesley, 1998.
- [11] M. Liljeberg, T. Alanko, M. Kojo, H. Laamanen, and K. Raatikainen. Optimizing World-Wide Web for Weakly Connected Mobile Workstations: An Indirect Approach. In *Proc. 2nd Intl. Workshop on Services in Distributed and Networked Environments (SDNE'95)*, pages 132–139, 1995.
- [12] A. Sahai and C. Morin. Mobile Agents for Enabling Mobile User Aware Applications. In *Proc. Autonomous Agents'98*, pages 205–211, 1998.
- [13] S. D. Walkty. *A New Architecture to Support Efficient Web Browsing in a Wireless Mobile Computing Environment*. MSc thesis, University of Manitoba, 2000.
- [14] B. Zenel. A General Purpose Proxy Filtering Mechanism Applied to the Mobile Environment. *Wireless Networks*, 5(5):391–409, 1999.