Mobile Code For Mobile Devices:

Migration for Improved Application Performance

By

Omid Afnan

BASc

A thesis submitted to

the Faculty of Graduate Studies and Research

in partial fulfillment of

the requirements of the degree of

Master of Applied Science

Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

Acceptance Form

Abstract

This work proposes a migration decision algorithm that considers the environmental conditions on a set of neighbouring servers and attempts to select the ideal execution platform at any give time. The algorithm is targeted toward handling the case of a weak mobile device connected by wireless links to a set of more powerful proxy servers. The algorithm allows migration from the mobile device to a proxy server, and subsequently back to the mobile device, or any other suitable proxy that can communicate with the hand-held. This paper starts with a summary of other proposed approaches for dealing with lower capability mobile devices, and then introduces the previous research upon which this particular approach is based. The proposed algorithm, an emulation platform, and a set of emulation results are presented. We conclude that the migration algorithm must predict future performance on each platform, and avoid reacting to short-term changes in the performance.

Acknowledgements

I would like to acknowledge first and foremost my loving and supportive wife Nooshfar, without whose patience, support and insistence I would never have finished this Thesis. Similarly, I would like to thank Dr. Kunz who gave a part-time student a chance, along with some excellent guidance and lots of patience. Thanks to Sarah Cosentino for her assistance in working out some difficult Voyager and coding problems. And finally, I would like to acknowledge CrossKeys Systems Corporation, ISOPIA Inc., and Sun Microsystems Inc. for their financial support.

Table of Contents

List of Tables

List of Figures

1 Introduction

This work focuses on the study of application execution on mobile devices and the possible improvement of application performance through load sharing between less capable mobile devices and more powerful fixed hosts called proxy servers. In order to build a system that supports this type of load sharing, tools are required to allow the migration of code between mobile devices and proxy servers. Furthermore, algorithms are needed to determine when a migration will improve performance and which proxy server is the best destination for the migration.

Previous work in this area has led to proposed algorithms for deciding when to migrate code modules from a mobile device to a proxy server. Other work has focused on the criteria for moving code from one proxy to another. Here we attempt to bring together these two ideas and, by combining and extending the previous work, propose an approach that allows migration to occur between a mobile device and several proxies.

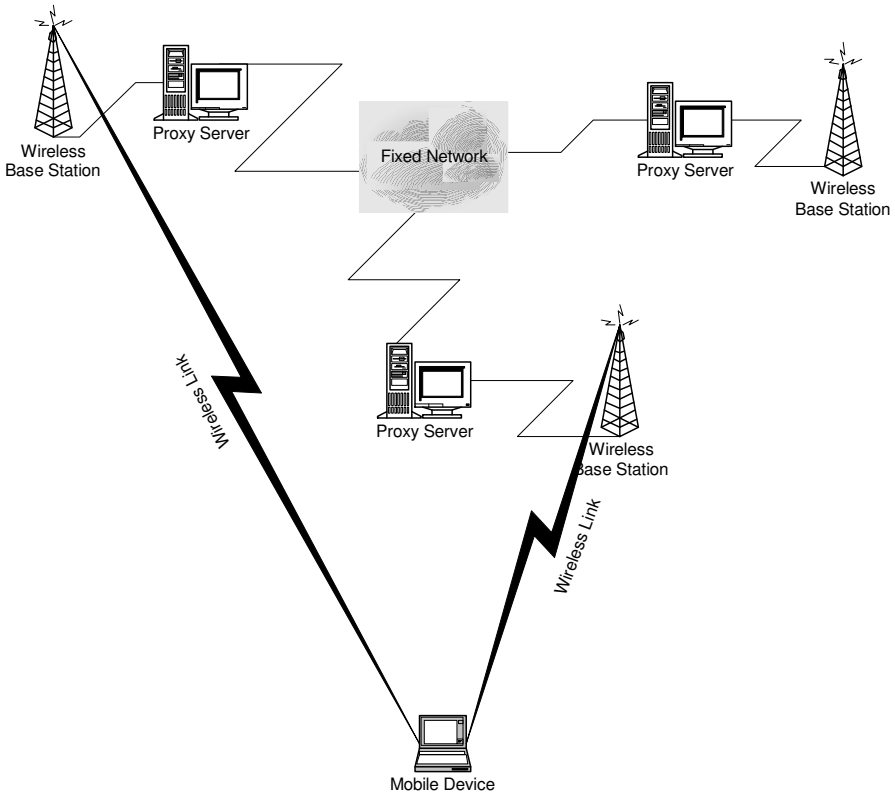

Figure 1 – Basic Mobile Network Environment

Figure 1 outlines the basic network environment under study in this work. A mobile device is linked to a large fixed network of computers using wireless links. These links are established between the mobile device and fixed hosts that have wireless transmission capability and are usually referred to as base stations. The base stations also have wireline links to a larger, fixed network of computers (e.g. the Internet). These base stations are essentially edge devices between the fixed and mobile networks. When these base stations take on processing tasks on behalf of the applications executing on the mobile device they are referred to as proxies. The role of proxy is useful in cases where the mobile device is unable to perform all the tasks

related to an application and therefore requires a more powerful host to offload some of those tasks.

This approach is based entirely on the movement of code modules between hosts in a client-proxy-server model where the client is a mobile device. The main contributions of this work are:

1. Combining code migration from client to proxy with code migrations between proxies,
2. Experimentation with dynamic migration scenarios (i.e. based on changes in the host and network conditions), and,
3. The design, creation and testing of an emulation platform built from commonly available hardware and software for use in assessing migration scenarios.

We begin with a survey of related work in Chapter 2. Major issues in mobile computing are presented and proposed solutions or approaches from various sources are summarized for each issue. Chapter 3 presents the concept of code migration and the general algorithmic outline of making migration decisions. Chapter 4 outlines the proposed protocol for making migration decisions using an extension of an already proposed neighbourhood approach. This chapter also discusses monitoring of the host environment and migration costs. In Chapter 5 the experimental platform used to validate this proposed approach is described, followed by experimental results in Chapter 6. Chapter 7 presents conclusions and outlines areas for future consideration.

2 Summary of Related Work

Designing an infrastructure suited for mobile clients requires design and engineering work in a number of areas in order to address challenges unique to mobile networks and devices. In this chapter we examine several previously proposed solutions for addressing existing limitations in the areas of bandwidth, protocol design and client resources.

No one approach addresses all limitations. Since we are interested in understanding the main issues, and possible solutions to them, this chapter is organized by types of limitations. For each category of limitations some specific proposed solutions are covered. These solutions are also summarized later in terms of general architecture, mobility models and scope of problems solved. The solutions reviewed in this paper are summarized in Table 1.

Proposed Solution	**Research Institution**	**Protocol Layer**	**Summary**
AgentTCL [Gray96]	Dartmouth College	Application	System supporting agents that move between full-featured mobile hosts and fixed hosts to complete computational tasks.
InfoPad [Le95], [Sesan94]	University of California – Berkeley	Infrastructure	In-building or campus system for portable desktops or *pads* that allow input and output but do not have their own processing capability.
InfoStations [Frenkiel00]	Rutgers	Infrastructure	High capacity base stations provide islands of connectivity to mobile devices traveling along predictable routes.
MASE [Meggers98], [Kreller98]	EU ACTS	Infrastructure	Middleware layer intended to enable multi-media applications on mobile networks with focus on QoS handling over cellular phone networks for mobile devices.

Mobile IP [Perkins96]	IETF	Infrastructure	Protocol for supporting mobile IP-based hosts based on a packet forwarding and reroute notification system.
Mobiware [Campbell97], [Angin98]	University of Columbia	Application	Middleware layer with open APIs focusing on QoS handling at several levels and allowing applications the flexibility of choosing how to respond to QoS changes.
Mowgli [Liljeberg96]	University of Helsinki	Infrastructure	Middleware specifically addressing Web browsing for mobile devices based on improvements to HTTP and TCP for the wireless portion of the connection to the mobile client.
Mowser [Bharadvaj98]	University of Missouri-Columbia	Infrastructure	Simple middleware layer providing transcoding to alter content streams to match mobile clients capabilities.
Odyssey [Noble99], [Satyanarayanan96]	Carnegie-Mellon	Application	Approach based on the concept of *fidelity* and a toolkit providing APIs that applications can use to respond to environment changes and select different fidelities.
Rover [Joseph95], [Joseph97]	MIT	Application and Infrastructure	Toolkit supporting transparent data stream modification, as well as supporting mobility-aware applications by providing queued RPC and Relocatable Dynamic Objects (RDOs).
TACC [Fox98]	University of California - Berkeley	Infrastructure	Architectural approach using four elements of *transformation, aggregation, caching* and *customization* at a proxy mediating between host and client. Coupled with scalable design based on adjustable pools of workers on the proxy.
WAP [Wap02]	WAP Forum	Application	Application and architectural framework being developed by consortium of primarily mobile phone manufacturers to enable the next generation of smart phones with processing Internet access capabilities.
WebExpress [Housel98], [Rodriguez99]	IBM	Infrastructure	Currently shipping hardware and software solution aimed at reducing and optimizing traffic flow for Internet and Web applications using proxy architecture.

Table 1 – Summary of Works Reviewed

2.1 Client Limitations

Limitations in the mobile client include such items as low power, weak processors, limited battery power, low-resolution video, and lack of audio. While generally recognized as limitations, client side issues are less often addressed by the solutions examined.

Most research considers one of two extremes: Fully functional devices such as laptop computers with no client limitations, or, dumb terminal devices (e.g. InfoPads), which are simple extensions of desktop input/output devices. These can almost be viewed as degenerate cases that do not allow us to focus on intermediate or dynamic conditions.

One intermediate design point that is addressed is that of intelligent cellular phones. The WAP standard and the corporate partners involved have addressed this type of device more extensively. In this case a more interesting combination of low-resolution imaging, weak processor and limited battery power are addressed together. This does not address a general computing solution as it is heavily focused on an information-browsing appliance. However, it is still significant as this type of device is likely to be a big part of the mobile domain.

WAP actually specifies a "microbrowser" which addresses several client limitations. While it is optimized to have a small memory footprint and low demand processing, it also allows adaptation of graphics to the monitor resolution of the mobile device. This solution is probably the most comprehensive approach to client limitations, but is very much geared toward solving the browser problem, rather than general application execution.

2.1.1 Weak Processor

Since clock speed and power consumption are closely related, mobile devices are likely to have slower or weaker processors. When addressed, the general approach is to offload the mobile device by shifting the processing to a proxy or gateway computer. This would be a powerful, non-mobile computer that is in direct contact with the client. This is the approach taken in solutions that work at the application level by introducing new intermediate components (such as Mowgli, TACC and WebExpress).

The approach taken by middleware solutions, such as WAP or MASE, is to encapsulate the weaker process by creating simple APIs or reduced operating system calls. In this approach, applications are forced to consider processor limitations by using a correspondingly limited set of functions. This approach requires rewriting of client code, which can be a drawback given the large existing base of applications.

2.1.2 Low Resolution Monitor

Limitations in displays are the most commonly addressed issue on the client side. Again, the most common approach for gateway solutions is to handle the problem in the proxy. Filtering and compression are the methods of choice.

MASE and WebExpress, for example, use filtering to remove unsupported content types from the data stream going to the client. Filters must be configured for the mobile client specifying which data types and what resolutions can be accepted. WAP is similar in that the microbrowser matches content to client capability.

Another solution is to replace content in the data stream with other content better suited to the mobile clients capabilities. This approach, often referred to as

transcoding, is based on some sort of lossy compression. In the case of Mowser, for example, video streams are replaced by representative sample stills. The approach in Berkeley's TACC architecture is also to replace content with lower resolution versions. This is set in a profile for the mobile device and is based on MIME types. This approach again depends on an upstream processor that can make these transformations efficiently and intelligently. A configuration method is also needed for populating the client preferences on the proxy, which is currently handled in a static manner by most solutions.

2.2 Bandwidth

The most widely addressed issues in mobile computing relate to bandwidth availability for mobile devices. Most approaches try to use the existing bandwidth more efficiently while handling periods of disconnection gracefully.

Splitting the connection between server and mobile client at the proxy is the most common approach for reducing the impact on the fixed network while addressing the wireless network's limitations. Using two different protocols on either side of the proxy opens up the possibility of using protocols specifically designed for wireless links.

In addition to using special protocols, a proxy on the fixed network allows caching and buffering in order to address the problem of disconnection. Since disconnection from, and reconnection to, the network are very common in mobile devices, the term *partial connectedness* is often used instead.

2.2.1 Low Throughput

To date wireless connections have lagged wireline connections in bandwidth and throughput by orders of magnitude. While absolute capacity for wireless links will certainly rise, and the gap between wireless and wireline is likely to be reduced, some disparity is always likely to remain. This means that a special approach will be needed, in the foreseeable future, for more efficient data handling on wireless links.

Compression of the data stream is one solution, as seen in Mowgli. However, simple compression introduces additional overhead on the mobile client that then needs additional processing for decompression. This may be one reason why compression is not a common solution. A variation on this approach (Mowgli, WAP) is to use a binary encoding format for the data stream over the wireless link to achieve some compression, but keep the decoding simple. This is more efficient than the standard protocols (e.g. HTTP) that contain human readable data. In this scenario, simpler compression means lower bandwidth gains, but also less processing demand on the client.

Data filtering (MASE, WebExpress, TACC) is usually used based on the mobile client's inability to display certain types of data, but it can also decrease bandwidth needs. To use it for this purpose, additional profile information has to be compiled for each client based on the wireless connection's quality of service. Currently, most filtering is based on static profiles of the mobile client that cannot support QoS of the link. Solutions like Mobiware that dynamically track QoS would be better suited to make use of dynamic filtering.

Again, middleware approaches take a slightly different approach. MASE and Mobiware both track QoS on the link and try to find the best connection (when multiple paths are available). In these solutions, QoS is tracked and links are possibly switched based on QoS changes. Applications are notified of changes in the QoS so that they can respond to such changes. This effectively pushes the problem up to the application level and forces the application to decide what to do in the case of reduced bandwidth. This is a very flexible approach, but requires greater awareness of network links by the application.

Another application level solution is provided by Odyssey. This toolkit provides APIs that allow applications to register ranges of resource availability. Once the required resources (most commonly bandwidth) stray outside the registered range, the application is notified. The application is expected to change *fidelity*: to select another quality level that requires lower (or higher) resource levels. The application is made aware of its environment and is required to adapt accordingly. The authors suggest that this solution can be applied to many systems as the adaptation portion can be coded outside the main application [Noble 99].

2.2.2 Partial Connectedness

Another area that is addressed by several approaches is that of partial connectedness. A common approach to addressing temporary breaks in connectivity is buffering at a proxy host on the edge of the fixed network. The InfoPad and InfoStation systems both take this approach. InfoPad deals with the case of a simple input/output tablet that is used to access applications on a server. In this case all output is buffered on the Pad Server while the Pad is out of range. A slightly more

complex scenario is presented in InfoStations where a mobile client makes requests over a constantly connected channel (GSM link) but receives responses only at information "islands" (antenna towers) where bulk transfers can be done. Again, information is collected and buffered at a proxy until a high bandwidth connection is available, and then transferred in bulk. InfoPad assumes brief disconnections within a building environment, while InfoStations are proposed for use with cars on highways. Buffering on the request (client) side is also a feature in some systems such as WebExpress.

Extensions to buffering include pre-fetching and background transfers. Pre-fetching is used by Mowgli and WebExpress to try and fetch related documents based on the current document. For example, links found on a Web page can be pre-fetched as soon as the Web page is loaded based on the assumption that the user is likely to click on those links in the near future. This information can be cached on the proxy until it is requested, or it can be transferred in the background before it is requested, but while connectivity is good. Then, if the page is requested during a disconnected phase, it will still be available. These solutions depend on the resources available on the client along with a prediction of how the connectivity to the mobile client may change over time.

Beyond buffering it is desirable to actually allow processing to continue at the proxy on behalf of the mobile client during periods of disconnection. This is proposed in both AgentTCL and MASE, which propose the use of agents to continue processing on behalf of the client while the client is disconnected. AgentTCL is based on a comprehensive design where agents queue on the client or proxy and migrate

when there is sufficient connectivity. These agents would then perform work autonomously and return to the client when finished.

The QoS handoff schemes discussed in Section 2.2.1 are also relevant here as a change in QoS can be used in the algorithms for deciding when to cache, when to transfer, and what approach to take for prediction (e.g. aggressive prediction). Mobiware and WebExpress are more suited to this kind of work as they handle changing QoS levels. InfoPad established a fixed QoS that is not as useful in this context.

2.2.3 Latency

Latency is a very difficult question to address in the context of data transfers over wireless links. With mobile clients the likelihood of disconnects is very high which means that latency will increase due to retransmits, reconnect overhead and general lack of connectivity. There are not many proposed solutions in this area. Pre-fetching, background transfer of predicted data, and caching on the mobile client are all proposed by both Mowgli and WebExpress.

Predictive fetching of data is inherently difficult and very dependent on the type of application being run. While it may be easy to predict Web page fetches, other applications (e.g. synchronous, collaborative ones) are not as predictable. To be most effective this would have to be tied in to on-board caching which drives up the need for storage resources on the mobile client.

2.3 Protocol Stack

Most data communication protocols currently in use have been designed for non-mobile networks. These protocols often do not function efficiently if simply applied to wireless links or mobile clients.

Lower layer (physical and data link) protocols for wireless links have been widely established, as this is the minimal requirement for having wireless networks. Current issues to be addressed are mostly in the network and transport layers where TCP/IP is the most widely used protocol set. Since these protocols do not make any special allowance for mobile or wireless communication, using them leads to inefficiency and lower performance.

Current approaches focus on splitting the connection at the edge of the fixed network. The protocol stack is then left unchanged in the fixed or wireline network while a new protocol is used over the mobile section of the network. These mobility aware protocols can then handle issues like handoff between mobile bearer services more efficiently.

2.3.1 Application Layer

Application layer solutions require the application to be aware of mobility issues and to deal with them in a way appropriate to the particular application. This idea is further discussed in the section on Architecture (Section 2.4) below.

2.3.2 Transport Layer

At the transport layer, reliable data transfer and connection handling are the key issues. In networks supporting mobile clients, handoff from one service point to another is a common occurrence. Although handoff occurs at the lower (physical)

layers the transport layer connections must also be fixed. Handoff can include short periods of disconnection until the channel is re-established.

Mobiware addresses handoff and short disconnections by trying to decrease the overhead of connection handling. It does this by bundling all connections to a single device and then allowing the bundle to be handled as one unit. Using other transport protocols, each connection would be torn down and re-established individually, leading to greater delays, even for short disconnects which are very common in mobile networks. This is very valuable in applications based on the Web, where a number of higher-level connections can exist even for single tasks.

WAP introduces the idea of *suspend* and *resume* functions for connections. This allows lightweight connection reestablishment for handoff or short disconnects. It also allows the mobile device to suspend its connections when going into power saving modes without paying the high overhead of connection initialization when the device "wakes up".

MASE offers similar functionality to WAP by holding connections at the proxy. One of the strengths of introducing a split connection is the ability to keep the TCP/IP connection active between the proxy and fixed server, while allowing temporary disconnects over the mobile network. Other split connection solutions like Mowgli and WebExpress offer the same advantages.

2.3.3 Network Layer

The network layer is where address resolution and searching for hosts is performed. Again, mobile devices create additional problems in routing. The split connection approach allows the fixed network to continue using existing protocols

while the proxy takes care of locating the mobile device. WAP and MASE both use this approach and simply rely on cellular phone networks to locate the mobile unit. Other solutions like TACC, Mowgli and InfoPad route data to the proxy located on the fixed network, but expect that proxy to be in direct communication with the mobile client. In either case, the approach seems to be that the proxy does not change (i.e. no handoff between proxies), but the proxy is expected to either be in direct contact with the mobile device, or route through a cellular network to reach it. As much of the routing as possible should be done in the fixed network. Any algorithm that is partially executed on the mobile clients is more likely to suffer from performance problems.

A more comprehensive solution is Mobile IP that primarily addresses routing for mobile devices. The basic approach here is to have forwarding agents which are part of the subnet where the mobile device last connected, and which can forward packets to the new location indicated by the mobile device. To further optimize the rerouting, the source host can be notified of the reroute so that further packets can be sent directly to the mobile device's new location. This is very much like a traditional post office with change of address notifications.

2.4 Architecture

All of the solutions investigated for this summary use the client-proxy-server architecture. This is an extension of the common client-server architecture where one host (server) provides certain services or application of which another host or device (client) makes use. The proxy is another host logically placed between the server and

client in order to mediate, route, or otherwise facilitate communication between them. Some architectures use the term proxy, while others use gateway or mediator.

The client-proxy-server approach can be implemented at different levels in the protocol stack. When it is applied to the network or transport layers the result is *mobility-transparent*[1] solutions. The use of proxies at the application layer requires the application to support mobility concepts, which is referred to as a *mobility-aware* solution.

2.4.1 Mobility-Transparent Solutions

Mobility solutions that are transparent to the application are desirable because of the reduced impact to the applications. They allow most of the fixed network (Internet) to remain architecturally unchanged. They also allow traditional protocols and software application designs to operate as they always have. They support the use of new and optimized protocols between the fixed network and mobile clients. Given the fact that fixed servers have thus far been several orders of magnitude more powerful than mobile devices, these solutions also allow the use of such powerful servers to perform resource intensive filtering, compression and protocol conversion tasks.

In [Badrinath 00] a framework is presented for generalizing this kind of data stream adaptation approach. The authors propose a generic view of the data stream between source and destination nodes that is made up of multiple Adaptation Agencies (AA). Each AA is further decomposed into an Event Manager (EM), a

---

[1] The terms mobility-transparent and mobility-aware are borrowed from [Joseph97].

Resource Monitor (RM), and zero or more Application Specific Adapters (ASA). The EM and the data stream itself provide inputs to the ASA. The ASA will attempt to transform the data stream and may make requests for additional resources from the RM. The RM is also informed by the EM of changes in resource utilization, etc. The authors show that many proposed mobility solutions, including Odyssey and Conductor, map well into this framework.

Most of the solutions summarized in Table 1 also map to this framework while the adaptation performed in the proxy varies with each solution. Most proxies are placed at the edge between the traditional fixed network and the mobile network. InfoPad and InfoStations use a proxy that is exactly on the edge of the two networks and is mostly used for caching or buffering data during disconnections. Mowgli, WAP, MASE, WebExpress and TACC all use the proxy to modify and adapt content to suit the mobile device. They use the greater computational and system resources of the gateway machine to "digest" the data stream and pass on a version which is suited to the wireless bandwidth and end-user device capabilities. AgentTCL uses proxies in a symmetric form where client and server roles are assigned as needed to the fixed host and mobile client.

2.4.2 Mobility-Aware Solutions

While transparent support for wireless networks and mobile clients is favoured as a non-intrusive method for integrating mobile networks with fixed networks, it can also be argued that the mobility cannot be fully exploited without the application’s involvement. In addition, the load created for proxies carrying out the transparent adaptation may create a scalability problem as wireless devices increase

in number. The proxy model results in massive amounts of data being passed over networks, followed by resource intensive computations on the proxies, which ultimately result in data being discarded or modified. This suggests that end-to-end solutions that involve the application in the adaptation decisions may provide better scalability [Joshi 00]. However, infrastructure solutions that are meant to be transparent can also be designed to address scalability as outlined below in Section 2.4.3.

This leads to the concept of applications that are aware of the mobile environment they are operating in. The common approach to achieve this is to structure mobility support into a middleware layer that the application can use to be notified of, and respond to, network changes. MASE, Mobiware, Mowgli, and Mowser all fall into this category. MASE and Mobiware each focus on QoS issues in slightly different ways, while Mowser and Mowgli focus exclusively on enabling better HTTP and TCP performance. In all cases, the use of a middleware approach gives greater flexibility for application development. Applications can be developed for many purposes and as long as they call on the functions of the middleware they can support some mobility. However, the short-term drawback of this approach is that many applications have to be re-written or re-engineered to use them. This may be less of an issue for areas where applications are only emerging now or are changing rapidly anyway.

An example of a toolkit intended for mobility-aware applications is Rover [Joseph 97]. The toolkit allows portions of the application's code to be moved from one host to another. This is the solution most similar to that being investigated in this

thesis. Rover defines two main concepts: Queued Remote Procedure Calls (QRPC) and Relocatable Dynamic Objects (RDO). RDOs are units of code that can be moved from one host to another. QRPCs are remote requests that can be queued while a link is down or a host is disconnected, and are serviced once a connection is re-established.

2.4.3 Scalability

Mobile devices are generally designed to be personal devices that are ubiquitous and easy to use. This implies that as design and cost issues are worked out the number of such devices will rapidly increase. Therefore, networks and protocols supporting mobile computing must address scalability.

InfoPad and InfoStations both attempt to support scalability for communication with mobile devices. InfoPad adopts a cellular communication network for communicating with individual pads. InfoStations attempt to address the cost of communication networks for mobile units by providing a smaller number of high-powered transmitting stations that create islands of connectivity. Both approaches try to minimize cost in the communication infrastructure.

TACC addresses scalability of processing on a proxy in support of filtering and transforming data streams headed to a mobile client. TACC segments work in the areas of transformation, aggregation, caching and customization. TACC then maintains a pool of "workers", for each area, which can be created and destroyed based on the offered load. This modular approach allows the system to increase processing capacity in the area that is needed at the time when it is needed. This

appears to be a sound approach to resource allocation and bottleneck removal as shown by experiments performed by the team at Berkeley.

MASE attempts to support scalability by using simplified APIs to encapsulate mobile device functionality. This seems to be based on the assumption that if the mobile devices are simple, then the APIs representing them should be simple, and therefore supporting a number of such simple devices should not be resources intensive. However, experiments with an implementation using CORBA suffered from performance problems, leading to a redesign of the system to reduce CORBA dependency.

Mobiware uses a multi-layer architecture that should support scalability and performance. However, use of technologies like Java and CORBA have similarly created problems in experiments.

Other solutions like WAP and WebExpress are intended for deployment on specialized servers designed on telephony principles. Although experimental results are not available, the intent seems to be to support scalability using specialized hardware.

AgentTCL is an example of a system that does not seem to take scalability into account. This solution requires mobile agents to queue in different nodes and then migrate as connectivity becomes available. The queuing and transport of these agents suggest a solution that will not scale well with an increase in the number of applications on a device, or the number of devices in general.

2.5 Adaptability

Each solution offers a different level of adaptation to changes in the network or client. Filtering and data conversion approaches like Mowgli and WebExpress allow a device profile to be setup for a mobile client but do not offer dynamic modification of that profile based on changing device parameters (e.g. decrease in power) or changing network condition (e.g. decrease in bandwidth). More thorough solutions like TACC indicate that such dynamic changes are supported.

Solutions measuring QoS also come in static and dynamic varieties. InfoPad establishes a fixed QoS level for its connections. QoS levels not being met will likely result in loss of connectivity. Mobiware on the other hand will measure QoS and supply the hooks for allowing the application to react and adjust. MASE also tracks changes in QoS and triggers changes based on them.

None of the proposed solutions indicate the ability to shift processing from the mobile client to the proxy and back again based on changing conditions. A system like TACC that allows dynamic device profiles can be used to support such a scenario. Also, AgentTCL should allow this since agents could be programmed to migrate off a device upon detection of changes in resources or capabilities. However, this does not seem to be a part of AgentTCL at this time as the client environment is not taken into account.

2.6 Modeling the Mobile Devices

In each of the approaches outlined above the mobile device is modeled, either implicitly or explicitly, as having a particular form and set of characteristics. The majority of proposed solutions address themselves to making Web browser functions

available on low-powered mobile devices. Filtering and transcoding are usually performed based on MIME types. Mowser and Mowgli are basically solutions for the Web. TACC is also based on MIME types, but its architecture should be applicable to a wider range of problems. InfoStations solve the problem of small requests resulting in large amounts of data being retrieved (basic Web model).

Some solutions make particular assumptions about the hardware platform being addressed. InfoPads only target a portable desktop scenario. WAP focuses primarily on cellular phone, albeit more advanced ones than currently available. The large user base projected for some of these platforms easily justify specific solutions for those platforms.

At the same time more generic solutions that can be applied to a wider range of hardware platforms and software applications are desirable for reduced adoption overhead. Middleware solutions, such as those outlined in Section 2.4.1, simplify mobility related actions but require the application to make the fundamental adaptation decisions, and are therefore applicable to all application classes. Examples like MASE, WebExpress and Mobiware, attempt to solve general classes of problems such as QoS handling and content reduction. Applications using these services must decide themselves what to do when QoS changes, or when to use content reduction.

2.7 Modeling the Patterns of Mobility

The solutions summarized here all support fully mobile clients as opposed to simply nomadic ones (i.e. clients which only periodically change network location). Most of the solutions allow mobile devices to change locations on a continual basis, but some impose limitations. For example, InfoStations assume a predictable route

for the mobile device. The ideal case for this approach is a one-dimensional model, such as a car traveling down a road. More complex behaviour, such as a person walking in a city, becomes more difficult to predict. InfoPads allow full mobility, but only in geographically limited areas such as a building or campus, and with minimal client capability.

WAP and MASE are both based on an underlying cellular phone network. They support full and continuous mobility. WebExpress goes a step further in claiming that its content optimization strategies can be used in fixed networks as well as mobile networks to improve throughput and reduce latency.

Mobiware and MASE both support QoS measurement and handoff. These solutions have a more complex mobility model in which they allow for the existence of several bearer services at one geographic location. This opens up the possibility of finding a better channel and switching to it even if the device itself is not mobile.

2.8 Relationship to Our Work

The research presented in this thesis is based on mobility-aware applications using a mobility support infrastructure. The architecture is based on a client-proxy-server approach at the application layer of the protocol stack. The proxy is an application layer entity and does not perform any distillation or transcoding. It is generally a host that is neither the display client (mobile device), nor the source information server. The application is (re)written to be distinctly divided into portions that can be executed on different hosts. These portions are written to use the appropriate infrastructure tools that support the actual movement of the code modules, remote execution, etc.

The infrastructure tools are similar in structure to the framework suggested by [Badrinath 00]. Environment monitors inform an application specific adapter of change in the environment, allowing the adapter to decide when code modules are to be moved. However, the adaptation here is not in the data stream, but in the deployment of the application modules themselves.

Overall, this represents an end-to-end solution [Joshi 00] where the application and the underlying support layers work together to deal with mobility problems and still deliver an acceptable service to the end-user. The contribution of this work is to extend the scope of code mobility previously investigated [Omar 00] and [Wang 99] to allow migration from mobile clients to proxies, between proxies, and from a proxy back to the mobile client. This extension of the previous algorithms is tested using scenarios with dynamic host and network conditions.

3 Migration Decision Algorithms

In this chapter we examine some general concepts that are part of the process of code migration from one system to another. In the context of this thesis, code migration is intended as a means for improving application performance. The migration may take place between the hand-held device and a proxy server (in either direction), or between proxy servers. Generally, the migration process consists of the following steps:

1. Partitioning the application,
2. Monitoring performance,
3. Detecting that performance is poor,
4. Creating a list of possible destination servers,
5. Deciding to handoff, and,
6. Performing the migration.

The monitoring of performance is a continuous activity that must be carried out by all host systems in the network. This may be a native function of the operating system, or it may be accomplished by a daemon process created for this purpose. Poor performance, however, is both application-dependent and subjective. A somewhat simpler task is to predict whether certain performance indicators are measured to be better on any other available server at a given time. It should also be noted that if an application is performing well on a given device it might not be necessary to migrate it, even if the resulting migration will improve performance. For example, if a video clip were being played at the required number of frames per second, it would not be necessary to try and increase the performance of the player.

It is not the intent of this work to address the measurement of subjective application quality. Therefore neither minimum nor maximum performance is defined or measured. The primary focus here is on comparing relative performance, before and after code migration, using different migration patterns. While performance after a migration may still be poor, we will consider our migration strategy successful if some improvement can be measured.

Partitioning of the application into stationary and movable segments, creating a list of possible destinations, and the actual decision criteria for handing off that partition, are areas that are critical to this thesis. They are covered in the following sections.

3.1 Partitioning

Partitioning of the code refers to the act of breaking up an application into distinct segments that can be migrated between hosts. The decision of how the code for an application is partitioned can be based on criteria such as minimum coupling, minimal partition sizing, or ease of separation. Minimum coupling is the most relevant in terms of system performance in a distributed mode. Since code partitions are likely to execute on distinct servers, the inter-partition coupling relates directly to the network traffic generated in our distributed system. Optimal partition sizing is more relevant for calculating initial migration costs as well as allowing for limited host resources at a migration destination.

In [Omar 00] partitioning is considered from the point of view of minimizing coupling between partitions. Coupling is measured by network traffic generated between partitions. The Greedy Graph Partition algorithm is introduced for arriving at

an optimal partition of the application based on a weighted graph representing all the modules in the application, and their interactions based on a particular execution cycle. Using an MP3 player as an experimental platform that work concludes that the only partition that improves performance is one where the display modules remain on the mobile client while all the decoder functions migrate to a proxy server.

Based on the above-mentioned work it appears that it is not worthwhile to investigate multiple partition distributions of the MP3 player. The optimal partitioning for each application can be different and therefore generalizations may not be appropriate. However, it would seem logical that an MPEG player would behave similarly to an MP3 player as these are based on the same technology. Since these are the two applications available for our research (as well as being extremely popular applications for personal devices), we have chosen not to investigate different partition distributions in this work. Instead we use the one-time partitioning, arrived at by [Omar 00] by creating one partition containing the presentation portion of the application, and one partition containing the decoding and computation engines.

3.2 Destination Selection

In [Omar 00] migration only occurs between the mobile client device and a proxy server. In this model the selection of a destination is not necessary as the client is assumed to have one choice at a given time. In [Wang 99] it is assumed that part of the application is already running on a proxy server, but that migration to another proxy server will be needed at some time. It is assumed that there is more than one proxy server that is in communication with the mobile client and therefore a choice

exists as to which host should take over execution of the application. Here we attempt to put these two types of migration into one algorithm.

The Group of Neighbours (GON) algorithm proposed in [Wang 99] is based on the exchange of information between a given proxy server other proxy servers that also have wireless connections to the hand-held. Hosts track information about their own operating environment and pass this information on to neighbouring proxies. Therefore each proxy ends up maintaining a list of local neighbours plus their last known system state. The decision to move, and which host to move to, is made based on this list. This forms the basis of the approach used in this thesis as well.

3.3 Handoff Decision

The decision as to when a portion of the application is to be handed off, or migrated, to another host is made based on performance improvement or resource depletion criteria.

The primary resource that can be depleted in mobile client devices is the battery power. Loss of power will affect both the ability to perform local computations and the ability to migrate (transmit) code or receive and display results. It is possible to circumvent loss of power through code migration, but this will not be further investigated in this work due to the difficulty of emulating power loss on our experimental platform. Since the loss of power is similar to the depletion of other resources, it should be possible to draw conclusion regarding the usefulness of our algorithms based on other experimental measurements.

Storage is another resource that can be depleted on a hand-held device. In this case we will only consider memory-based storage as most hand-held devices only

have memory chips. This is changing with the introduction of micro hard drives suitable for hand-held devices, possibly making storage limitations a non-issue. Even if this does turn out to be the case it can be argued that the hard disk acts as an extension of the memory system and therefore our treatment of the memory system in this research can be extended to cover the availability of hard disk storage.

Processor availability may also be depleted. In other words, the processor may become busy with other tasks and therefore make fewer processing cycles available to the application. This is more common in a multi-user scenario and therefore less likely to occur on a mobile client device that is usually a single-user device. It is still possible that system tasks or user-initiated tasks running in the background may create a load on the mobile device. However, this kind of resource depletion is a major factor for proxy servers being used to offload the hand-held device. As the proxy servers are shared systems, likely serving a multitude of hand-held devices, the tracking of system load is a fundamental part of the handoff decision algorithm.

Performance improvement includes increased speed of execution due to a more powerful computing device, increased network bandwidth or decreased latency due to better network conditions or connections, and improved response due to relative loads on the available hosts. In order to attempt to take advantage of better host or network conditions, it is necessary to be able to measure performance factors in the current environment as well as measuring the environment of other hosts. This leads to the need for environment monitors that can report on local conditions, sharing of information about the environments on other hosts, as well as a function for comparing these conditions.

4 Protocol Extension

In this chapter we cover some additions or extensions to the work outlined in the last chapter in order to create a specific approach for performance improvement which uses migration of code between mobile client and proxy, as well as between proxies.

The approach taken is to extend the concept of "neighbourhood" as it applies to potential migration destinations. The first extension is to include the mobile client device within the neighbourhood, making it possible to apply the same algorithm for migration decision between client and proxy, as well as proxy to proxy. Another extension is to define the idea of "distance" between two hosts as being measured by a combination of the migration cost and potential performance gain differential between the two platforms. These extensions are discussed in the following sections.

4.1 Environment Monitoring

As discussed above, the changing conditions of the mobile client device and various proxy hosts on the fixed network are of interest to us. Therefore, the execution environment must be monitored on any platform where a part of the application may execute. Since we are interested in the performance of the environment in support of the application's performance, the environment parameters being monitored are chosen based on the application and are the same for the mobile device as well as the proxies.

For each host, the **effective system capacity (ESC)** is monitored. This is intended as a measure of the current processing capacity of a host loaded with any

number of tasks. While many systems provide a processor speed or load measurements, these are usually only measures of clock speed or the length of the run queue respectively, which cannot be used to measure relative performance between systems. If the system reported processor load could be combined with the processor speed we would have a portable measure of the system load [Arndt 98]. Again, those operating systems that report processor speed often report only the clock speed, which is relative to the architecture of the system and is not necessarily comparable across systems. To overcome these problems, we directly measure the **unloaded system capacity (UPC)** by running a processor intensive code segment, measuring its execution time, and using the inverse of that value. In this case, we use a routine for creating Magic Square matrices that results in a mix of integer and floating-point math, along with array manipulation. This measured number is then used as the processor speed.

To get a more accurate ESC, the processor load is calculated in two ways. If the processor is not fully loaded, then we use the percentage of time that the processor is not idle as the processor load. But, we also have to capture the case where the processor is overloaded and is using the scheduler to handle a queue of jobs. In this case we use the more traditional UNIX/Linux style "load" measure. The algorithm, therefore, defines a load factor, λ, based on the above and defined as:

$$\lambda = \begin{cases} a+1, & t_i < 0.15 \\ 2-t_i, & t_i \geq 0.15 \end{cases}$$

where t_i *denotes* the percentage of time the CPU is idle, and a denotes the load factor reported by the Linux operating system. When the CPU is very busy (low values of t_i), the load is likely to come from multiple processes and we use the Linux load measure and add 1. This gives us an approximation of the current number of processes (including our own) that would compete for processing resources. When the CPU is idle (high values of t_i), we subtract the actual value of the CPU utilization from 2. This means that for low CPU utilization, λ will range from 1 (completely unloaded CPU) to 1.85. For higher utilization, the value will range from 2 to infinity. It should be noted that the Linux load measure is the time averaged process run-queue length. At higher loading the Linux load measure indicates how many resources may be competing for the CPU [Arndt98]. The ESC is then defined as:

$$ESC = \frac{USC}{\lambda}$$

As noted, the depletion of memory is also of interest to us. Low memory on the mobile unit can result in the rejection of new jobs or the suspension of the existing job. On proxies, where memory is always backed up by hard disk storage, low memory results in frequent swapping and decrease in performance across all active applications. Memory is therefore a factor in performance and a count of **free memory (FM)** is monitored for each host.

Environmental factors related to the wireless connection between the mobile device and a proxy server must also be considered. The **mobile to proxy bandwidth ($\mathbf{BW_{m,p}}$)** between a host and the mobile device is the main factor that affects the performance of the application. In order to determine whether a host should become the active proxy, it is important to measure the speed of the connection it has to the mobile. In deciding whether or not to migrate the proxy portion of the code from one host to another, it is necessary to compare this measure between potential migration targets.

The bandwidth is measured by sending two ping packets from the system to the hand-held. A ping server application is run on the system emulating the hand-held in order to respond to these pings. The two packets are of different sizes and the difference in round-trip times is measured. This is intended to provide a bandwidth measure by considering the incremental cost of sending each bit rather than the loaded cost of sending a packet (including overhead). This measure is called the PingDelta in this work.

The bandwidth between proxy systems is not considered. We assume a high speed network is available to connect these systems and that the network is redundant

enough to ensure the same bandwidth over time. Also, since a neighbourhood is by definition localized, we will not have to deal with long-haul connections or other network conditions that change unpredictability.

The code transfer delays are considered in the migration decision algorithm as a fixed cost. This cost drives up the threshold for the minimum performance increase required from a desirable target destination. In the experimental setup migration cost is minimized by having the application code pre-loaded on each host. In a production system having a shared code server accessible by all hosts can minimize these costs. Even if code has to be transmitted between hosts, the most common case will be to transfer it between two proxy servers connected by a high-speed (wireline) network, not over the slower wireless network.

4.2 Thresholds for Migration

The decision to migrate the pre-selected code partition is based on the environmental parameters being monitored. The intent is to migrate the code every time better performance is available elsewhere. Deciding whether performance will be better elsewhere is based on a comparison of current system metrics compared to metrics for potential target systems. There are a number of parameters that are introduced for deciding on when to migrate. The intention of this work is not to discover the ideal values for these thresholds in a dynamic or general way, but rather to test their usefulness while assuming that reasonable values can be found in some way.

Since a minimum performance criteria is not being set, a migration will happen every time better execution conditions are detected on another platform. This

means that each time new information is available for the neighbouring servers, the migration decision algorithm is run to decide whether a migration is favourable.

The decision algorithm simply tries to find a host where none of the environmental measures are worse than the current system and where at least one factor is significantly improved. To achieve the necessary comparison, it should be noted that even two successive measurements, on the same host where no load changes have taken place, are unlikely to yield the same exact value. Therefore the algorithm uses ranges of values for determining “equality” and “improvement”. In this thesis equality is considered any value that is better than 95% of the original value, whereas significant improvement is taken as a 120% or better performance for any measured value. Special consideration must be given to boundary values, particularly for the PingDelta measurement. These considerations are shown in Rule 3 of the algorithm below. The need for these special considerations are outlined in Chapter 6. Instantaneous performance values are also not reliable. Transient conditions, such as periodic waking up and polling performed by suspended applications, Java garbage collection, and periodic operating system tasks that occur in the background, can easily skew single measurements. Therefore, instead of using instantaneous measurements, we use average values calculated over a number of samples. The number of previous samples used determines how quickly or slowly the algorithm will react to changes in environmental variables. In the experiments outline below we use a moving window of 20 samples, each spaced 10 seconds apart.

The use of average values is a backward looking strategy for assessing the current state of the system. A more aggressive strategy, and one that is used here, is to

try and predict the future behaviour of the system. To do this, we measure the standard deviation along with the mean. The standard deviation for the set of samples provides us with an indication of the stability of the measured values. In order to be able to compare standard deviation values to fixed threshold, we first normalize each of the measured values in the set against the mean value, and then calculate the standard deviation. This provides a dimensionless measure that can be compared to an absolute threshold. The threshold value of 10 is used when considering standard deviation measures. If the standard deviation is greater than 10 the system is considered volatile.

This stability, together with the expected load created by the application being controlled and migrated, can be used to predict the future performance of a system being considered for migration. The simple approach taken here is to look for targets where the environment has a certain level of stability and is capable of performance that is equal to or better than the load created by the application. This is reflected in the general algorithm as shown below:

```
FOR (each potential target):
 target.suitability = 0.0

 // Rule 1 – check for basic memory
 IF
    (memory available on target does not meet minimum standard)
 THEN
    skip to next host
 ENDIF

 // Rule 2 – try to find the ideal target with higher speed AND
 // higher bandwidth (lower PingDelta)
 IF ( target.speed > current.speed * minImprovementFactor
    AND  target.pingdelta < current.pingdelta /
                                  minImprovementFactor)
       target.suitability = 1.0

       // Rule 2a – the measurements on the target should be
       // stable over time.

       IF ( variance for target.speed > acceptable threshold
            OR  variance for target.pingdelta > acceptable threshold )

          target.suitability = 0.75
       ENDIF
 ENDIF

 // Rule 3 – try to find a target with EITHER higher speed OR
 // higher bandwidth (lower PingDelta)
 IF (
    (target.speed > current.speed * minImprovementFactor
    AND  ( target.pingdelta < current.pingdelta / minEqualityFactor
             OR target.pingdelta <= 10 ) )
    OR
    (target.speed > current.speed * minEqualityFactor
```

```
        AND  ( target.pingdelta < current.pingdelta /
                                    minImprovementFactor
                AND target.pingdelta > 10 ) )
        )
        target.suitability = 0.5

            // Rule 3a – the measurements on the target should be
            // stable over time.

            IF ( variance for target.speed > acceptable threshold
                 OR  variance of target.pingdelta> acceptablethreshold)

               target.suitability = 0.25
            ENDIF
    ENDIF
    END FOR
    MigrationTarget = (Host with suitability rating >= 0.5)
    IF MigrationTarget is not null THEN migrate to MigrationHost
```

In this algorithm we introduce the concept of suitability while considering multiple environmental factors and their combinations. The algorithm does not only try to determine whether there will be an improvement, but rather tries to determine relative degrees of improvement. Therefore, a target where all conditions are better is a perfectly suitable target. Another system where only some conditions are better is a less suitable target, and one with no improvement is not suitable at all. In our implementation we use a suitability score of 0.5 as a minimum for triggering migration. In this case it means that the target system must be significantly better in at least one performance measure, and show consistency in past measurements. This

threshold score can be adjusted for a more aggressive or conservative migration strategy.

An extension of this approach would be to use fuzzy logic techniques in the decision algorithm. This could be done as a two level problem where the measured environment factors are mapped to fuzzy sets representing levels of improvement. Then fuzzy rules would be applied to combine these variables in a min-max type combination. This extension could result in better interpretation of disparate system measures, but is left for future study.

4.3 Extension of Neighboring

In [Wang 99] a group of neighbours (GON) is defined as the set of proxy servers that have direct communication links to the mobile device. The GON concept is used to define where to look for possible alternative execution platforms. It is assumed that the partition of the code that was migrated off the mobile device will be passed on to other proxies, but not back to the mobile device. Here we extend the GON approach to include the mobile device as a potential migration target. In other words we include the possibility that sending the code back to the mobile device may, in some scenarios, produce better results than shipping the code to another proxy.

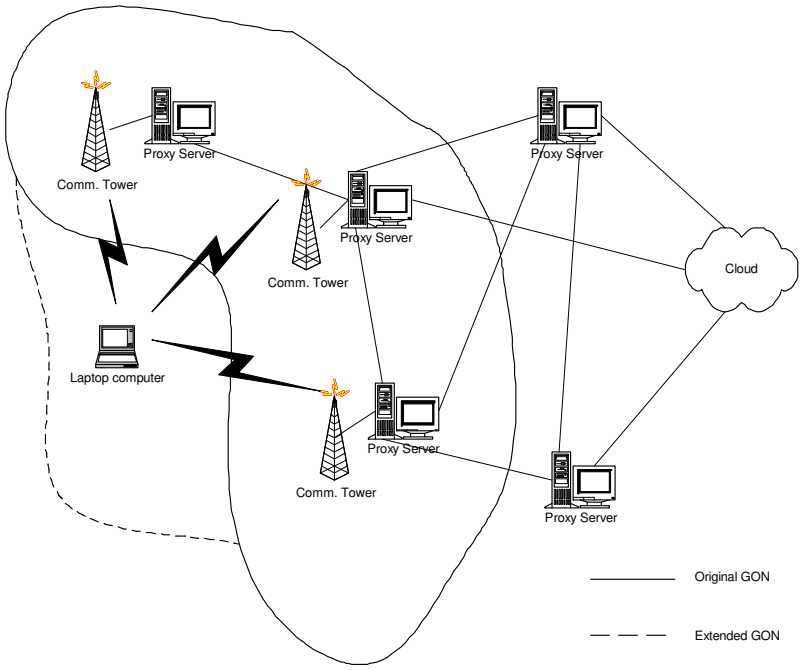

Figure 2 – Extension to Group of Neighbours

Figure 2 shows the extension of the neighbouring concept to include the mobile device. Note that hosts not in direct contact with the mobile device are not considered as neighbours for the purpose of the migration algorithms.

5 Experimental Tools and Applications

The experimental platform for exploring the performance of the mobile code system is based on using real applications and emulated execution environments. This approach was selected to provide the closest approximation of a real-world system given that hand-held devices and wireless networks of the required flexibility and capability are not cheaply or readily available at this time. It also provides better control over the execution environment and allows repeatability of tests.

The entire platform is run on Intel based systems using the Linux operating system, which provides easy querying of system metrics. An MPEG and an MP3 application written in Java are used in conjunction with the Voyager ORB. A number of other programs were created to monitor system metrics, create loads, and control migration. Each major tool is briefly outlined below.

5.1 Tools

Figure 3 shows the basic distribution, across various servers, of the experimental tools used for our experiments. These tools are described in the following sections.

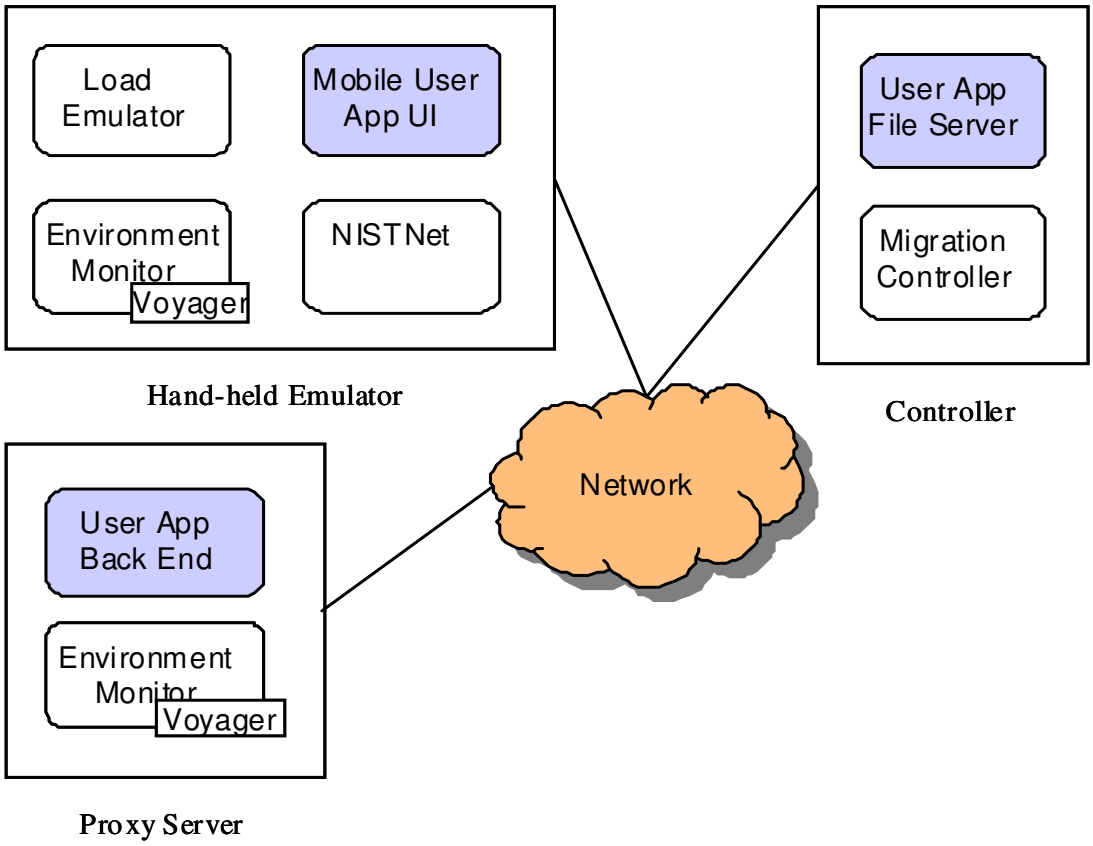

Figure 3 – Distribution of Test Tools on Servers

5.1.1 Voyager

Voyager is an object request broker (ORB) with extra features supporting code mobility. An ORB is part of the CORBA architecture for distributed computing and is intended for connecting together service requestors and service providers. It generally acts as an intermediary that provides mechanisms for the creation of remote objects, publishing of services offered by such objects, and subscription or use of such services by other objects.

In the case of Voyager, additional constructs are provided for handling mobile code. This makes it a suitable middleware layer for use in the mobile code

experiments summarized in Chapter 6 . To allow code migration, several services are provided:

- Ability to create objects remotely. This is performed using the *Factory.create()* method which takes parameters for the location at which the object is to be instantiated.
- Ability to refer to objects by proxy interfaces. The *Factory.create()* method returns a proxy for the object created. The proxies for other objects can be retrieved using the *Proxy.of()* method.
- Mobility is handled by a *Mobility* class which allows method invocations such as *moveTo()* with parameters specifying a target host.

Use of Voyager requires the Voyager ORB to be installed on each server in the network. It also requires Voyager libraries and method invocations to be included into the source code of the application. Voyager requires a mobility-aware programming model and provides supporting tools for explicit handling of mobility. As outlined in Section 2.4.2, requiring applications to become mobility-aware introduces an adoption barrier. However, as a research tool, it simplifies the software development part of the work by providing a number of services that are difficult to implement without an existing structure. Also, Voyager provides better performance with less effort than specially developed, experimental mobility toolkits [Omar 00].

Additionally, we use ORB features for both publishing, and subsequently invoking, services across the network. This is particularly useful for accessing the environment monitors and neighbourhood maps of migration targets.

5.1.2 Load Emulator

In order to use common desktop computers to mimic mobile devices, a special emulator was developed. This emulator runs on a Linux environment and reduces the memory and processor speed available to the application of interest, thereby emulating the execution environment of a typical mobile host. Test applications are executed within this emulator running on desktop Linux hosts rather than using actual hand-held computers. This allows us to use existing lab equipment, as well as reducing the system and communication problems often encountered in new generations of hand-held devices.

Most current operating systems implement sophisticated processor scheduling algorithms which make it difficult to deterministically reduce the processor time available for a single job. Therefore the emulator increases the processor load by running a number of extraneous child processes. The emulator attempts to reach a certain level of resource usage at start time and does not attempt to add or subtract children to maintain that level. In order to create a realistic scenario we want to be able to create an initial load and then let the operating system react as it would normally to the introduction of additional load when an application is migrated.

The emulator can also reduce the amount of memory available to the application. Again, the reduction of memory is not a simple task as most operating systems provide memory managers expressly designed to create the illusion of unlimited memory for the application. The emulator is implemented using C++ as this language has fewer safeguards than Java for the usage of system resources. This feature is not used in the set of experiments outlined here.

5.1.3 Environment Monitors

Environment monitoring is performed by a Java application that can read system information from the "/proc" pseudo-directory in Linux. In Linux, system and process information is presented in a form that mimics files and directories that can be accessed using normal file processing techniques. The monitor can also read values from a file of "recorded" values in order to reproduce specific scenarios. The monitor can be set to poll system parameters periodically and write these to file, providing a mechanism for creating recorded script files.

The monitor is also responsible for reading and publishing the neighbourhood map. This is a text file that lists the addresses of all hosts considered part of the neighbourhood. The metrics collected by the monitor are system load, processor speed, free memory count, and bandwidth to the hand-held.

All metrics, with the exception of the bandwidth, are polled only when requested from a client using the environment monitor's services. This is to reduce unnecessary polling on systems not currently part of the decision algorithm. The monitor measures the bandwidth independently and periodically. This is done because the pings may take a relatively long time. Since the migration decision algorithm must poll several systems, and the polls should occur as near in time as possible, it is preferable to have the bandwidth measurement already available when the poll occurs in order to minimize wait times.

5.1.4 Migration Controller

The migration controller is located on a single host and tracks the current location where the back end partition of the user application is executing. It reads the

neighbourhood map for that host and starts to poll the listed neighbours. The migration decision algorithm (see Section 4.2) is coded in the controller. Every time it finds a host that is a suitable migration target it signals the back end partition to move to the new location. It does this using Voyager commands to control the back end, which is also Voyager enabled. It then reads the neighbourhood information for the new location and continues to poll.

5.1.5 NISTNet

NISTNet is a network emulation tool developed as a project of the National Institute of Standards and Technology Internetworking Technology Group [NIST 02]. It runs on Linux only and manipulates network traffic at the IP protocol level. It can be installed on a router within a network from which point it can affect all traffic that is routed through it. In this mode, an inexpensive Linux host can be used within any kind of IP network to emulate more complex network interactions. It can also be installed on a single IP host in which case it will only be able to affect the traffic originated from or destined for the particular host.

In our experimental setup, NISTNet is installed on the host emulating the hand-held device. In our model, the hand-held device uses a wireless network that has a lower bandwidth than the proxy machines that reside on a fixed network. Therefore, the traffic between the hosts representing the proxy machines is neither modulated nor measured during the experiments. The bandwidth to the hand-held is reduced using NISTNet and measured by the Environment Monitors located on the proxy machines (see Section 5.1.3).

Due to the fact that the Environment Monitors use single “ping” style packets for measuring bandwidth, NISTNet must be set to transmit packets after the delay period it introduces. NISTNet provides the choice of sending the packet before, in the middle of, or after the inter-packet delay that it introduces when trying to emulate a given bandwidth level. Since the Monitor uses a single ping packet’s travel time in bandwidth calculations, it is important that the delay due to bandwidth be measurable for the single packet. Therefore transmission after the delay is the right setting. This behaviour is set in the “Config” file for NISTNet and must be set before compiling the tool.

NISTNet shapes traffic by buffering incoming packet streams and transmitting them with delays in order to emulate the decreased bandwidth, loss, duplication, latency and other effects observed in an IP network. In this set of experiments, only the bandwidth modulation feature is used.

5.2 Mobile Test Applications

In order to experiment with the migration algorithm outlined above we required sample applications. Ideally, such applications would be the same as those most commonly used on hand-held devices. In the previous research ([Omar 00], [Wang 99]) two applications have been developed: an MPEG player, and an MP3 player. These are in fact typical applications for use on personal devices, as well as being processing and bandwidth intensive, which makes them good test applications. Each application has been modified slightly to reflect a simpler partitioning of the code. Based on the results of [Omar 00], each application is broken into a display portion and a single large partition that can be migrated.

5.2.1 MPEG Player

The MPEG Player is a completely Java-based application that plays standard MPEG files. Its primary components are a scanner, a decoder and a displayer. The displayer instantiates a scanner object set to read in the appropriate source file. It then instantiates a decoder object and passes it the scanner object. The decoder then uses the scanner to read in the source file and decodes the stream to produce and combine the various frame types created in MPEG streams. The decoder produces single frames that are passed back to the displayer to be rendered graphically using Java AWT classes and methods.

The displayer portion of the code always remains on the hand-held device. The decoder and scanner, and all the related classes (such as the Huffman and DCT decoders) are available as one unit for migration to other hosts. The decoder and scanner objects are created using Voyager's factory classes and subsequently managed using Voyager enabled interfaces. This allows the use of Voyager migration features. The displayer, who ultimately contains the main loop of the application, also provides a method that the Migration Controller can call to trigger the migration of the back-end.

The MPEG player has two threads. The displayer uses one thread; the other is a user interface thread that handles input from the buttons on the UI for starting and stopping the playback. The display thread theoretically provides the mechanisms for suspending activity while the back-end objects are migrated to a new host. However, normal thread operations cannot be applied during a migration, as these semantics are not defined across Java virtual machines. Therefore the decoder and scanner had to be

altered to be re-entrant in such a way that they could be prematurely halted, and then restarted from the same point in their processing. This was coupled with similar semantics in the displayer's main loop so that it could be made to pause while a migration took place.

The migration process of Voyager consists of the following steps:

-Serialization of all the data and state information for a class, and all of its contained classes;

-Creation of objects on the destination server;

-Transmission of serialized data to the voyager server, and population of data into the new objects;

-Destruction of the objects on the originating server.

Voyager also provides stub methods that can be coded for pre- and post-migration activities on both the originating and destination servers. The MPEG file being decoded and played is always served by a separate host with a Voyager-based connection point which is defined during application startup.

5.2.2 MP3 Player

A second test application, an MP3 player, is also completely implemented in Java, making it easy to adapt for use with Voyager. This implementation of an MP3 decoder reads in files in MP3 format and produces WAVE format output files. There is no graphical UI, as there is in the case of the MPEG player. Execution is initiated from the command line and runs to completion with no further user interaction. The output is a complete WAVE file written to disk on the server where execution was started.

The original player is divided into three separate, mobility-enabled segments: The front-end, or output; the decoding logic; and, the file server, which reads the original MP3 file from disk and passes bits to the decoder. This segmentation allows the file server portion to be placed on a separate host, as would be the case with streaming audio. The front-end which creates the output WAVE file and the decoder is separate so that the front-end can be kept on the hand-held device while the decoder back-end is migrated by the Migration Controller during the experiments.

6 Experimental Results

Based on the preceding discussion and tools, a number of experiments were conducted. The purpose of these experiments is to:

1) Determine whether application performance can be improved based on the ideas and algorithms outlined,
2) Verify and adjust the parameters of the migration algorithm, and,
3) Integrate and stabilize the test-bed platform

The following sub-sections summarize the description of the tests performed and the results obtained.

6.1 Experimental Test-Bed

The tests are conducted on a set of Intel-based computers running the RedHat Version 7.0 Linux operating system. Five such servers are set up on an isolated network. The network consists of a fast Ethernet system with an observed bandwidth in the 10Mbps range. Depending on the experiment being conducted each machine plays the role of controller, hand-held, or proxy server.

One host acts as a common fileserver, exporting an NFS directory structure containing both the executable tools, and collected test output. The Voyager servers access this common file system for copies of Java class files used in running the user application for the experiments. This also means that code segments are not transferred between hosts during migrations. Therefore, a migration consists only of a hand-off of control and state information. While this results in much faster

migrations, there is still a non-zero hand-off cost, particularly between heavily loaded hosts.

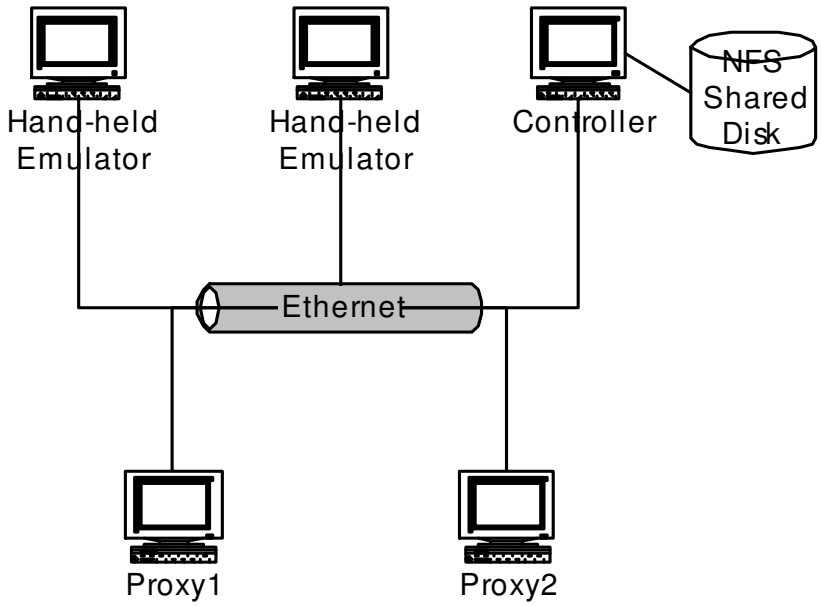

Figure 4 – Servers Used for the Experimental Test-bed

Figure 4 shows this experimental test-bed. The following sections further describe the particular equipment and environments used to perform the tests in this thesis.

6.1.1 Hand-held Emulators

Hosts emulating hand-held devices execute the load emulator program, a PingServer, an Environment Monitor, and a bandwidth control. A load value of two (2) was used for all hand-held emulators. This results in a measured speed in the Environment Monitors of about 280 to 320. Unloaded machines in our test-bed had speed measures of 2800 to 3200. This speed measure has no meaningful unit. It is simply a relative measure arrived at using the methodology outlined in Section 4.1.

However, having hand-held devices that are slower than standard desktop computers by an order of magnitude is quite reasonable.

The hand-held must also provide a PingServer, which Environment Monitors on the proxies use to bounce packets back and forth so they can measure actual bandwidth to the hand-held. Since the migration algorithm only considers bandwidth between proxies and hand-helds, there is no need for a PingServer on any other type of machine. The Environment Monitor is the program that sends the ping packets to the PingServer. Each hand-held must also have a running Environment Monitor, as the hand-held is one of the nodes considered as part of the neighbourhood when a migration decision is made.

NISTNet provides bandwidth control. Since this tool only affects incoming traffic streams, it is run on whichever host is emulating the hand-held device. Depending on the experiment, bandwidth reduction rules are either added at the beginning or during the execution period of the test.

The actual application (e.g. the MPEG player) is always started on the hand-held emulator. The load generation and monitor programs are started first, followed by the user application. The application must create its own Voyager server and Voyager-enabled objects. It must also publish a Voyager handle and port, which the decision maker can contact for requesting migrations. There is no need to start a separate Voyager server on any host. The back-end portion is migrated from the hand-held emulator at a later stage depending on the particular dynamics of the execution.

6.1.2 Proxy Servers

Hosts acting as proxy servers only need to run the Environment Monitor program, and, during selected periods, the load generation tool. The Environment Monitor incorporates a Voyager server that has knowledge of the path for loading the application class files. Therefore, when migrations are required, the Monitor provides the execution environment for launching the application's back-end classes.

Loads are created on proxies using the same load generator as for the hand-held emulation. Loads are either started manually at the beginning of the execution for the duration of the test run, or they are started further into the test. For pulse shaped loads the Linux **`at`** command is used to schedule the execution of the load generator, as well as a later **`kill`** command to terminate the processes created.

6.1.3 Controller

In order to separate loads created by the decision maker algorithm and other supporting tools, we designate one host as the Controller. This host executes the migration decision algorithm, and also acts as the fileserver for the actual data file being processed by the user application in the experiments. For example, the model used is that an MPEG file would be streamed or served from another host in the fixed network. Since the performance of such a file server is not considered as one of the issues being dealt with here, it is acceptable to reuse the Controller to also act as the fileserver. A single controller can also execute the decision algorithm and fileservers for multiple concurrent experiments.

6.2 Configuring the Test-bed

To simplify the configuration of the overall test-bed, and the execution of particular scenarios, each tool is designed to read its operating parameters from a text configuration file. An execution scenario is first identified and then the configuration files for the Environment Monitors, Migration Decision Maker, and neighbourhood maps are created. In the actual test-bed, these configuration files are located on the same, shared file-system as the Java class files (see Figure 4). This allows complex and error-prone configuration settings to be set once and used many times, often across scenarios.

6.3 Base Scenarios

A number of base scenarios were executed on the test-bed. These are tests that show the execution of the user application if migration is not available, or is not triggered. The most common ones are:

BaseA – The user application executes completely on the hand-held emulator. The hand-held is loaded as described above. The decision maker is not run, so no migrations occur. The input file is served from the Controller host.

BaseB – The user application begins on the hand-held with a decision maker running. A migration is made to an unloaded proxy. The proxy then becomes loaded to a high level (load of 10), but execution completes without another migration.

BaseC – The user application begins on the hand-held, migrates to an unloaded proxy and completes execution there.

These base cases are used in the following sections for measuring changes in performance as various system parameters are varied. The first experiment compares some of these base cases to each other. Subsequently, BaseA is used primarily to show the absolute improvement over the case of having no code mobility. BaseB is used to compare against the case of simply splitting the code across two servers, with no consideration of multiple migrations. BaseC is primarily used for comparison with cases where bandwidth is varied.

6.4 Execution of Scenarios

In addition to the base scenarios outlined above, additional execution scenarios are used to exercise particular aspects of the migration algorithm and the test-bed in general. Configuration files are created to capture the physical set-up needed for these scenarios. From the set of five hosts available, we attempt to keep the roles (e.g. hand-held or proxy) constant to minimize variability due to host differences. In some experiments where all hosts are not needed multiple iterations of the same scenario are run concurrently by splitting the system into several smaller sets of hand-helds, proxies and controllers.

Each scenario is iterated at least seven times to produce a reasonable set of results, and the average values across iterations are summarized in the tables and charts below. A larger number of iterations can be used to reduce the variability in the results. However, the results already show acceptable variances (e.g. variance of 200,000 on an average of 5,000,000 ms execution time). Furthermore, in an emulated environment, as in the real world equivalent, there are a number of unpredictable variables (e.g. garbage collection, background system tasks, user loads, etc.) which

would likely lead to even larger variances than those observed here. Therefore the results are considered with an eye toward significant gains or losses in performances, rather than small improvements.

A particular observation about the MPEG player's execution characteristics is required. In the graphs presented below it can be noted that the performance of the player is not even across subsequent frame repaints. This is due to the MPEG encoding standard [ISO 98] that uses a combination of fully represented I-frames, specifying a complete single image, followed by P- and B-frames which are frames that are predicted or interpolated from other previous (or future) frames. This means that some frames will take a very short time to decode due to their simple encoding, while others require more time-consuming decoding (i.e. the I-frames which are fully encoded JPEG images). Because of this the graphs showing the per-frame rendering times clearly show three average duration levels for a single experiment. For this reason these graphs are presented as point graphs only to provide the clearest depiction possible. To compare two test scenarios, each of the three frame types from one test has to be compared to the matching set of data-points in the other test. This can be done visually, as the pattern of time distributions is usually the same across experiments, with a translation up or down the y-axis.

The charts presented in the following sections graph the execution time for each frame of an MPEG movie played. The x-axis shows the frame number, and the y-axis shows the length of time to paint that frame. It should be noted that the Java-based MPEG player cannot currently play movies at the required frame rates. This is

true of the original MPEG player and even more pronounced after the application is converted to use Voyager.

6.4.1 Simple Migration Due to Load

The first experiment consists of executing the BaseA and BaseC scenarios. The purpose of this experiment is to compare results from simply running the user application to completion on the lower powered hand-held, versus migrating the back-end of the application to an unloaded proxy. This case validates the essential concept that splitting the application across two machines with a network connection can lead to performance improvement.

BaseA scenarios require only a single hand-held emulator host plus a fileserver. The fileservers for multiple experiments were run on the same platform, with 4 hand-held emulators executing iterations of the scenario concurrently.

BaseC scenarios require two hosts, a hand-held and a proxy, with the Decision Maker running so that the initial migration from hand-held to proxy occurs. This migration occurred within the first 15 frames in each case.

Scenario	Average Total Exec Time (ms)	Base To Compare To	% Improvement
BaseA	8,691,987		
BaseC	4,787,537	BaseA	45%

Table 2- BaseA and BaseC Execution Results

Table 2 shows that this simple move creates a saving on the overall execution time. A savings of 45% against remaining on the weak hand-held device is a significant enough gain to meet our criteria of searching for large gains rather than

marginal improvements. Figure 5 shows that this saving is evenly distributed across the entire program's execution. In other words each frame is being processed more quickly with significant overall gains. This suggests also that a short period of improved performance is unlikely to have as large an impact.

Figure 5 –Execution Times For Base Cases

6.4.2 Migration Back to Hand-held

We saw in Section 6.4.1 that a migration to an unloaded proxy produces gains in a small system with only one hand-held device and a single proxy available for offloading processing. We now examine the best strategy in the case that the proxy server becomes loaded to find out whether it is better to remain on the proxy or return to the hand-held.

In the LoadedProxy scenario, execution begins on the hand-held, followed by a migration of the back end to an unloaded proxy. At about 10 minutes into execution a large load is created on the proxy (the average start time of this load, across all reuns, is shown as a vertical bar in Figure 6). The proxy's load is large in keeping with the idea that a shared proxy is more likely to become heavily loaded during peak usage times. The back-end is then returned to the hand-held where it finishes execution.

Scenario	Average Total Exec Time (ms)	Base To Compare To	% Improvement
BaseA	8,691,987		
BaseB	16,812,517	BaseA	-93%
LoadedProxy	8,785,646	BaseB	48%
	8,785,646	BaseA	-1%

Table 3—BaseA, BaseB and LoadedProxy Execution Results

Table 3 compares the LoadedProxy results against the case of remaining on the weak hand-held (BaseA), and the case of staying on the proxy even when a load exists (BaseB).

Going to the proxy for only 10 minutes until a load is introduced and then migrating back to the hand-held appears here to be slightly less efficient than just staying on the hand-held. However, the 1% difference is negligible and is also too fine a measurement for the tolerances of the test-bed, as stated above. Therefore, it would seem that staying on the hand-held, or moving to the proxy and back again in a short time, are about equivalent from a performance perspective. However, Figure 6 shows that the LoadedProxy case has a higher inter-frame time during the early part of execution. This seems to be due to the migration and settling costs of the system (state transfer plus extra garbage collection incurred by Voyager). Therefore, it would be preferable to not migrate since there is no real overall performance gain, but there is a period of degraded performance.

Staying on the loaded proxy would not be a better choice. Here we see a –93% penalty for failing to leave the proxy once it becomes loaded. The largeness of this number is partly due to the fact that the load on the proxy is heavy, as mentioned above. As seen in Figure 6, each execution cycle is considerably slower during the lifetime of the execution after the load on the proxy is created. Obviously there is no indication in BaseB of the migration costs seen for the LoadedProxy scenario. However, it is also clear that paying the penalty on a few of the execution cycles would be better than staying long term on the loaded proxy.

Figure 6 – Execution Times for Migration Back to Hand-held vs. Base Cases

It would seem that having a generally conservative approach to migration is warranted. Short-term moves to other machines do not produce the gains desired and introduce short-term performance degradation that can be noticeable to the user. At the same time, good prediction of the future is required as failing to react to a long-term load has significant negative impact on overall performance.

6.4.3 Transient Loads

Having examined short-term gains in the previous section, we now examine the impact of short-term losses or loads. Transient loads are emulated in the Square-noMig, Square-Mig and TwoSpike-noMig scenarios.

The Square-Mig case covers the situation where a square shaped load pulse is seen on the proxy server after execution has been transferred from the hand-held. The length of the load is about 15 minutes and the magnitude of the load is high, as in the LoadedProxy case of Section 6.4.2. The approximate start and end times for this load are shown as vertical bars in Figure 7. The decision maker algorithm is set such that a migration back to hand-held occurs once the pulse starts, and another migration to the proxy occurs after the pulse dies off. The Square-noMig scenario is the same, except that execution continues on the proxy throughout the period of the pulse load.

The TwoSpike-noMig scenario is similar to the previous scenarios, but the square pulse is replace by two 3 minutes spikes, one at about 10 minutes and the other at about 22 minutes into execution. In other words, the two spikes occur in the same time period as the square pulse.

Scenario	Average Total Exec Time (ms)	Base To Compare To	% Improvement
Square-noMig	5,597,524		
Square-Mig	5,337,221	Square-noMig	5%
TwoSpike-noMig	5,893,441	Square-Mig	-10%
TwoSpike-noMig	5,893,441	Square-noMig	-5%

Table 4 – Pulse Load Execution Results

The table above shows that moving off the proxy during the brief square shaped load (Square-Mig vs. Square-noMig) yields only a 5 improvement over the case of just staying on the proxy. Again, a 5% difference is not valid for drawing conclusions based on the emulation test-bed. Furthermore, even a 10% gain is likely not sufficient given our criteria of having significant gains to offset the disruption caused to the user during certain frames.

From examination of the detailed execution information for the runs of Square-noMig, there is an extra time hit that is taken due to a garbage collection cycle that consistently starts between cycle 30 and 50 of the execution. This is approximately the same time as the load burst on the proxy and the cumulative effect is seen as a large spike in Figure 7. In the other test sets it seems that the garbage collection and the load are better separated in time.

The case of migrating back to the hand-held (Square-Mig) can be compared to both of the non-migrating cases. This is due to the fact that the Decision Maker would not trigger a migration back to the proxy between the closely spaced load spikes. In all cases the performance changes between migrating or not migrating for two spikes or a 15 minutes load seem to indicate that variations will be in the range of 5-10%.

This indicates that performance gains of sufficient magnitude will not be seen here and therefore the migration algorithm should also ignore short loads.

Figure 7 shows areas of increased delay around the 50 frame (cycle of execution) mark. This is the time during which the load spikes are occurring. As noted above, Java garbage collection has skewed the results in the case of Square-noMig. Otherwise, we see slightly higher cycle times for Square-Mig during that early period due to the additional migrations included in that test.

The results in this section suggest that migration based on short load variations does not produce significant gains. It can also create time-local delays in specific frame repaints that would be noticeable to the user. Therefore the migration algorithm should be set to ignore small load bursts as it should ignore short periods of non-activity.

Figure 7 – Execution Times for Transient Loads

6.4.4 Basic Bandwidth Variations

The experiments in this section consider the impact of bandwidth variations between the hand-held device and the proxy executing the back-end of the user application. So far, we have examined gains from finding a more capable or faster proxy host to execute the back-end across a very high bandwidth connection to the hand-held. Here we emulate lower throughput wireless connections by using NISTNet to throttle the packet transmission between proxy and hand-held.

The scenarios of BW=30kbps, BW=300kbps and BW=1Mbps have the exact same structure. Execution begins on the hand-held and is then migrated to higher speed proxy with high bandwidth. After migration the network connection speed immediately drops to the listed value (in bps) and execution completes on the proxy server. All proxies have very low loads before the application is migrated to them.

IMT-2000 [Magedanz 96] specifies 144Kbps for vehicular wireless systems, 384Kbps for pedestrian networks, and 2Mbps for indoor office wireless networks. The bandwidth settings for these experiments are approximately based on these settings, but with the low end further reduced to enhance the comparative results. Also, since the maximum observed speed of the network being used was around 10 Mbps, the high end of the limited bandwidth cases was set to 1Mbps to provide an order of magnitude variation from the open network.

BaseA and BaseC are both used as comparison points. BaseA helps us determine what would happen if in light of low bandwidth between hand-held and proxy we decide to execute on the hand-held only. BaseC is used to show the relative

impact of bandwidth, in general, as compared to the case of maximum bandwidth with the same set-up of hosts.

Scenario	Average Total Exec Time (ms)	Base To Compare To	% Improvement
BaseA	8,691,987		
BaseC	4,787,537	BaseA	45%
BW=30kpbs	15,351,201	BaseC	-221%
BW=300kbps	6,957,066	BaseC	-45%
BW=1Mbps	5,513,632	BaseC	-15%
BW=30kbps	15,351,201	BaseA	-77%
BW=300kbps	6,957,066	BaseA	20%
BW=1Mbps	5,513,632	BaseA	37%

Table 5 – Basic Bandwidth Variation Execution Results

Table 5 shows that, as expected, decreasing bandwidth results in slower performance. BaseC is the best case for the split application, running on wireline LAN with an observed speed in the 10Mbps range. As the bandwidth is reduced to 1 Mbps, 300Kbps and 30 kbps, the performance is significantly degraded. This seems to indicate that the MPEG viewer is bandwidth-dependent. This makes sense, as the application must transmit, over the wireless connection, pixel information for each screen repaint on the hand-held device.

Comparing the same test cases against complete execution on the hand-held only (BaseA) yields interesting results. The table shows that there are gains to be made by migrating across a low bandwidth connection and executing on a proxy that is more lightly loaded (or more capable) than the hand-held. While a bandwidth of

less than 300 Kbps continues to yield significantly poorer (-77%) results than just staying on the hand-held, moving to the proxy shows 20-40% improvement even though bandwidth is sacrificed. It should be noted that the speed measure for the hand-held device is about 300, while the same measure for the proxy is about 3000.

This suggests that our algorithm is correct in assuming that "equal" performance for a given parameter should not be considered in terms of strict equality of the measured numbers. There should in fact be a range of worse or lower values that are also deemed "good enough" if there is another parameter that measures significantly higher. This particular set of experiments also shows that the amount of degradation that can be considered acceptable is wider than at first imagined. Here we see 2 orders of magnitude reduction in the bandwidth being offset by one order of magnitude improvement in server performance.

Figure 8 simply illustrates the per-cycle execution times for these sets of scenarios: As expected gains and losses in performance are spread evenly across the per-frame timings.

Bandwidth significantly affects performance of the application. Therefore, the algorithm should look for better proxy to hand-held bandwidth when possible. However, execution is more sensitive to a faster proxy server than to decrease in bandwidth. This result may be application-specific and should be further tested by using other applications (see Section 6.4.6), and preferably other application types.

Figure 8 – Execution Times for Basic Bandwidth Variations

6.4.5 Multi-Proxy with Bandwidth Variation

In the next experiment the 2Proxy-LowBW scenario is tested. This scenario starts with execution on the hand-held with two proxies in the neighbourhood. The Decision Maker migrates the back-end to one of the proxies as both are lightly loaded. After about 10 minutes the bandwidth between the active proxy and the hand-held is reduced to about 300Kbps. The Decision Maker then migrates the back-end to the other proxy, which has no load and high bandwidth. Execution completes on this proxy.

Scenario	Average Total Exec Time (ms)	Base To Compare To	% Improvement
BW=300kbps	6,957,066		
2Proxy-LowBW	4,066,677	BW=300kbps	42%

Table 6 – Two Proxies with Decreasing BW Results

Table 6 shows comparison of this scenario with BW=300kbps, which was the case of continuing to execute on the proxy with poorer, but acceptable, bandwidth. We already saw in Section 6.4.4 that in such a situation going back to the hand-held is not a better choice. Here we see that migrating to another proxy instead, which has similar processor characteristics, but better bandwidth to the hand-held produces a 42% improvement. This reinforces the idea that while it is better to stay on the proxy if the only other choice is the hand-held, it is more advantageous to continue to search for another proxy that has a better connection to the hand-held.

Figure 9 – Execution Times with Two Proxies

Figure 9 shows that there is more volatility in the inter-frame times for the 2Proxy-LowBW scenario than we saw in the early cycles. This is due to the double migration required. This cost appears to be quite low and in fact is not very different from the natural fluctuations in the application's execution cycles. It would seem that this level of volatility is acceptable given the large overall gain.

6.4.6 MP3 Player Tests

In order to test the generality of the migration algorithm, the base cases outlined in Section 6.3 were repeated with the MPEG test application being replaced by the MP3 player described in Section 5.2.2. Table 7 shows the results for all the base cases.

Scenario	Average Total Exec Time (ms)	Base To Compare To	% Improvement
MP3,BaseA	4,737,215		
MP3,BaseB	7,629,504	MP3,BaseA	-61%
MP3,BaseC	4,345,505	MP3,BaseA	8%

Table 7 – Base Case Results for MP3 Player

The case of executing on an unloaded proxy (MP3,BaseC) provides only a marginal improvement of 8% over merely executing on a weak hand-held device. This level of improvement is below the threshold that we would consider sufficient improvement to warrant migration. In the case of the MPEG player a similar migration resulted in 45% improvement. This may suggest that the MP3 player is more bandwidth dependent that the MPEG player. Although in these test scenarios the bandwidth of the network is left at its maximum, this bandwidth is still lower than

the internal "bandwidth" of running both partitions of the application on the same server.

As expected the case of executing on a loaded server (MP3,BaseB) results in a large degradation in the performance. It can be concluded that the small gain from moving to a more powerful proxy is quickly offset by a load being added on that proxy.

Scenario	Average Total Exec Time (ms)	Base To Compare To	% Improvement
MP3,BW=30kbps	128,389,753	MP3,BaseA	-2610%
MP3,BW=300kbps	14,284,244	MP3,BaseA	-202%
MP3,BW=1Mbps	5,537,255	MP3,BaseA	-17%
MP3,BW=30kbps	128,389,753	MP3,BaseC	-2855%
MP3,BW=300kbps	14,284,244	MP3,BaseC	-229%
MP3,BW=1Mbps	5,537,255	MP3,BaseC	-27%

Table 8 – Reduced Bandwidth Cases for MP3 Player

To further investigate the dependency of the MP3 player on bandwidth between the front-end and back-end partitions, we run the tests of Section 6.4.4. Table 8 shows results for the three cases of bandwidth being reduced between the hand-held and the proxy. These results further confirm that the MP3 player is indeed very bandwidth sensitive. The decrease in performance is more pronounced and more rapid than in the case of the MPEG player. The case of 30 kbps bandwidth takes so long to execute that we did not perform the full set of seven iterations as in other experiments. Also, it is apparent that the increase in processor speed or capability is

completely offset by the negative impact of the decreased bandwidth. In the case of this application a much higher minimum bandwidth, along with significant processor improvement, is required to ensure that migration to a proxy server is worthwhile.

Figure 10 shows the per cycle execution times for the base cases of the MP3 player. The cycles of the decoder's execution are much more closely clustered than those of the MPEG player. This is due to the fact that the audio in an MP3 file is not encoded using the interpolation techniques of MPEG encoding, and therefore all decoding iterations complete in a smaller range of times. The profile for MP3,BaseB shows a notable step-increase when the load is introduced 10 minutes into the execution. This effect is easier to see in this case since all the cycle times are intrinsically closer in value in the absence of system load variations. The execution of the bandwidth reduction cases is not graphed as it simply shows one well-clustered band of cycle times for each of the bandwidth levels tested.

The results of the MP3 player tests suggest that some information is needed about the sensitivity of the application under control to various environmental parameters. Specifically, the level of coupling between the partitions, and the load generated by each partition should be factored into the migration decision.

Figure 10 – Execution Times for MP3 Base Cases

6.5 Adjustments to the Migration Algorithm

The execution of the experiments, and the resulting outcomes, led to a number of adjustments to the migration algorithm. The first version of the algorithm checked to ensure that all environmental parameters on a potential target host were at least as good as those on the current host. Putting this condition first meant that potentially good targets were disqualified. For example, a host with 5 times CPU capacity, but 1.5 times lower bandwidth to the hand-held would be ignored since the bandwidth is not considered to meet the minimum for equality. The algorithm does use thresholds for approximating equality, and these thresholds can be (and were) adjusted. However, a complementary solution is to rearrange the decision algorithm to look at strong winners first, and look at boundary conditions last.

The measurement of bandwidth to the hand-held requires special handling. The Environment Monitor on the hand-held emulator will always measure an average value of zero for the delay (PingDelta, or relative "slowness", in our case) from the hand-held to the hand-held itself. This causes a problem as our approach of specifying a multiplicative factor for calculating "equality" or improvement will not work. Having said this, the monitor also does not manage to consistently measure zero on the hand-held either. Transient values greater than 10 are periodically recorded by the monitor and these throw off the variance calculations in the migration algorithm. Therefore, we need a special rule to handle this lower bound of the PingDelta range, which means any value of about 10 or less. PingDelta values for other bandwidth

settings were observed to be in the range of 10 to 500, indicating that 10 is a good lower bound value. This rule is introduced in two parts:

1) When looking for an equally good PingDelta value, consider any value less than 10 on the potential target as passing the equality test. **Justification:** The equality test tries to ensure that the potential target is no worse than the current environment. Any host with a PingDelta of less than 10 essentially has the highest bandwidth possible and should pass this test. At the same time, a purely mathematical test comparing a current PingDelta of 1 to a target host's PingDelta of 3 would fail if equality was set to a minimum of 80% (i.e. $1 \div 0.8 = 1.25$, and since $3 > 1.25$ the rule would fail when it should not).

2) Only look for an improved PingDelta value if the current host's PingDelta is higher than 10. **Justification:** If the current host has a sufficiently low PingDelta, then there is no point in looking for a slight improvement by moving elsewhere. Again, the multiplicative approach used for values above 10 is justified, but for low values we should recognize that a 6 is as good as a 2 even if mathematically there is a factor of 3 improvement.

Also, for the migration algorithm, the threshold value for acceptable variance of the PingDelta values is separated and set to a higher value.

Overall, the algorithm was changed to take on a more fuzzy approach to determining a suitable target. Initial versions of the algorithm only distinguished

between suitable and unsuitable hosts. The migration would target the first suitable host. Further refinement led to the creation of a gradation of suitability, which is summarized in the table below.

Criteria	Suitability
All environmental factors clearly better than current host.	1.00
All environmental factors clearly better than current host, but measurements show high variance.	0.75
One environmental factor is clearly better and others are roughly equal.	0.50
One environmental factor is clearly better and others are roughly equal, but measurements show high variance.	0.25
All others.	0.00

6.6 Evolution of the Test-bed

The set-up of the test-bed was simplified during the development of the experiments. At first Voyager servers were started individually and separately from other tools such as the MPEG player or the Environment Monitors. This was useful during the development and integration of the test tools as it provided separate logging facilities and the ability to track errors down to individual components. The correct sequencing and timing for the starting of these components became more difficult as all the pieces were brought together in the final test-bed. Therefore, components were combined where it made sense. The Voyager servers, which provide mobility and remote access, were integrated into the Environment Monitors. The monitor becomes the only universal component that needs to be executed on all

the hosts that are part of the mobility experiments (except the platform where the migration algorithm executes).

Using Voyager as the middleware for distribution is relatively simple, but not problem-free. The low performance of the system is the most obvious issue. Profiling tools were used to identify performance bottlenecks in the MPEG player. There are areas where such improvements can be made (e.g. string and array handling), but these are relatively minor compared to the overhead introduced by simply adding interface based programming required by Voyager. The performance of the MPEG player on a single machine without using Voyager, compared to the same platform with use of Voyager, showed one to two orders of magnitude difference in performance. While beyond the scope of this work, it is clear that better distribution tools are required. It should also be noted that the version of Voyager being used is a free research edition with fewer features than the full product. Also, the mobility functions of Voyager have been removed from the professional product until a future release. All this suggests that other equivalent tools will be needed.

The version of Voyager used also drives the version of Java being used. Version 1.1.8, used in this test-bed, is quite out of date at this time and should be replaced by more current versions that provide better garbage collection and general performance improvements [Sun 00].

7 Conclusions & Future Work

The experimental results in the previous section suggest that a system for measuring the system and network environment where an application is executing, and using this information to trigger the migration of sections of application code, can result in improved performance. Furthermore, a functional emulation test-bed is proposed, built, tested, and shown to be usable in the execution of test scenarios and the observation of the behaviour of migration algorithms under consideration.

General observations about performance improvement are possible. The migration decision algorithm should be written such that it avoids reacting to short-term changes. Migrations due to short load spikes, or migrations that result only in a short period of improved performance, do not provide a significant gain in the overall execution time. Such moves also incur a cost that manifests itself in time-local degradation of the user experience (slower frame repaints, for example), which further reduce the small positive gains in overall performance.

The foregoing suggests that a good prediction approach is required in the migration algorithm. In this thesis the time-averaged past performance, along with a measure of the variance of individual samples of that performance, are used as a predictor for the future. In the limited scenarios tested, this appears to be a sufficiently accurate prediction mechanism for preventing low improvement migrations.

Bandwidth between proxy and hand-held is shown to have a significant impact on the performance of split-code applications. In the case of one test application (MP3 player) bandwidth is by far the most significant parameter. For the

other application (MPEG player), a trade-off is easily detected between CPU performance and bandwidth. In the latter case a sufficient improvement in CPU capability can offset a larger amount of bandwidth degradation. This observation suggests that while the same general approach can be taken for multiple applications, the parameters and their relative weights cannot be fixed. It also suggests that the sensitivity of the application to bandwidth and CPU should be made a part of the algorithm.

As mentioned in [Omar 00], a given application should be partitioned into front-end and back-end pieces based on an algorithm that tries to minimize the coupling between the partitions. While this is a valid point for the application designer, the migration algorithm still needs to measure, or be informed of, the inter-partition coupling that a split-code application has. This coupling ideally should be measured in terms of the bandwidth requirements, or as a sensitivity measure. In the same way, the back-end partition's processing needs should be measured or recorded for the algorithm. With these two pieces of information the migration algorithm can scale the relative improvement that it looks for when selecting migration targets. This approach requires both a way to measure the sensitivity, or desired range, for each environmental measurement used by the migration algorithm, as well as an adjustment of the algorithm itself. These, along with the following other areas of investigation, can be the subject of future investigation.

Other areas for further research include better prediction of future performance, especially for more complicated load profiles. This would benefit from a study of typical workload profiles observed on machines that would serve as

proxies in a production environment (e.g. a service provider's network). Accounting for such prediction would also suggest further refinement of the migration decision algorithm to allow handling of more complex interactions between the parameters being measured and the desired outcomes. The fact that most measurements are not precise, and the values are used to measure relative performance, suggests that a more formalized use of fuzzy logic is likely to help in creating a more adaptive decision algorithm.

A parameter that is being considered only tangentially is the cost of migration. Currently it is factored in as a consideration in setting the minimum threshold of improvement that will justify a migration, and this is done manually. The fact that in our test-bed migration costs are minimized due to the transmission of state information rather than actual code, makes this a small issue for now. However, in a production system it is likely that code for user applications would not be pre-populated on proxy servers and therefore the migration cost would be higher. This would probably warrant a more thorough treatment of cost in the migration decision algorithm.

References

[Angin 98] O. Angin, et al, "The Mobiware Toolkit: Programmable Support for Adaptive Mobile Networking," *IEEE Personal Communications Magazine: Special Issue on Adaptive Mobile Systems,* August 1998, pp 32-43.

[Arndt 98] O. Arndt, B. Freisleben, T. Kielmann, F. Thilo, "Scheduling Parallel Applications in Networks of Mixed Uniprocessor/Multiprocessor Workstations," *Proceedings International Society for Computers and their Applications, Parallel and Distributed Computing and Systems '98*, Chicago, IL, 1998, pp 190-197.

[Badrinath 00] B. R. Badrinath, A. Fox, L. Kleinrock, G. Popek, P. Reiher, and M. Satyanarayanan, "A Conceptual Framework for Network and Client Adaptation," *Mobile Networks and Applications,* Vol. 5, No. 4, 2000, pp 221-231.

[Bharadvaj 98] H. Bharadvaj, A. Joshi and S. Auephanwiriyakul, "An Active Transcoding Proxy to Support Mobile Web Access," *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems,* West Lafayette, Indiana, October 1998, pp 118-123.

[Campbell 97] A. T. Campbell, "Mobiware: QOS Aware Middleware for Mobile Multimedia Communications", *Proceedings of the 7th IFIP International Conference on High Performance Networking,* White Plains, New York, April 1997, pp 166-183.

[Fox 98] A. Fox, et al, "Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives," *IEEE Personal Communications*, Vol. 5, No. 4, August 1998, pp 10-19.

[Frenkiel 00] R. H. Frenkiel, et al, "The Infostations Challenge: Balancing Cost and Ubiquity in Delivering Wireless Data." *IEEE Personal Communications.* Vol. 7, No. 2, April 2000, pp 66-71.

[Gray 96] R. Gray, et al, "Mobile Agents for Mobile Computing." Technical Report PCS-TR96-285, Department of Computer Science, Dartmouth College, Hanover, NH, 1996.

[Housel 98] B. C. Housel, George Samaras, and David B. Lindquist, "WebExpress: A Client/Intercept Based System for Optimizing Web Browsing in a Wireless Environment," *Mobile Networks and Applications,* Vol. 3, No. 4, 1998, pp 419-431.

[ISO 98] ISO Standard MPEG-1 Coding of Moving Pictures and Audio, ISO/IEC JTC1/SC29/ MPEG 98, September 1998. Retrieved August 12, 2002 from http://www.cselt.stet.it/mpeg/standards/mpeg-1/mpeg-1.htm.

[Joseph 95] A. D. Joseph, et al, "Rover: A Toolkit for Mobile Information Access," *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, Copper Mountain, Co., December 1995, pp 156-171.

[Joseph 97] A. Joseph, J. Tauber, and M. Kaashoek, "Mobile Computing with the Rover Toolkit," *IEEE Transactions on Computers: Special issue on Mobile Computing,* February 1997, pp 337-352.

[Joshi 00] A. Joshi, “On Proxy Agents, Mobility and Web Access,” *Mobile Networks and Applications,* Vol. 5, No. 4, 2000, pp 233-241.

[Kreller 98] B. Kreller, et al, "UMTS: A Middleware and Mobile-API Approach," *IEEE Personal Communications*, Vol. 5, No. 2, April 1998, pp 32-38.

[Le 95] M. Le, F. Burghart, S. Seshan, and J. Rabey, "InfoNet: The Networking Infrastructure for InfoPad," *Digest of Papers, IEEE COMP- CON `95: Technologies for the Information Superhighway*, March 1995, pp 163-168.

[Liljeberg 96] M. Liljeberg, et al, “Enhanced Services for World-Wide Web in Mobile WAN Environment,” Report C-1996-28, University of Helsinki, Department of Computer Science, Helskinki, 1996.

[Magedanz 96] T. Magedanz, “Integration and Evolution of Existing Mobile Telecommunications Systems Towards UMTS”, *IEEE Communications Magazine*, September 1996, pp 90-96.

[Meggers 98] J. Meggers, et al., “A Multimedia Communication Architecture for Hand-held Devices,” *Proceedings of the 9th IEEE International Symposium on Personal Indoor and Mobile Radio Communications*, Boston, September 1998.

[NIST 02] NIST Information Technology Laboratory, NIST Net Home Page. Retrieved August 13, 2002 from http://snad.ncsl.nist.gov/itg/nistnet.

[Noble 99] B. D. Noble and M. Satyanarayanan, “Experience with Adaptive Mobile Applications in Odyssey,” *Mobile Networks and Applications,* Vol. 4, No. 4, 1999, pp 245-254.

[Omar 00] S. Omar, "A Mobile Code Toolkit for Adaptive Mobile Applications," Thesis (MCS), Carleton University, Ottawa, ON, 2000.

[Perkins 96] C. Perkins, ed., "IP Mobility Support," IETF RFC 2002, October 1996.

[Rodriguez 99] J. R. Rodriguez, et al, Mobile Computing: The eNetwork Wireless Solution, IBM International Technical Support Organization, Research Triangle Park, NC, March 1999, pp 205-210.

[Satyanarayanan 96] M. Satyanarayanan, "Fundamental Challenges in Mobile Computing," *Proceedings of the Fifteenth ACM Symposium on Principles of Distributed Computing*, Philadelphia, PA, May 1996, pp 1-7.

[Sesan 94] S. Seshan, M.T. Le, F. Burghardt, and J. Rabaey, "Software Architecture of the InfoPad System," Mobidata Workshop, November 1994.

[Sun 00] Sun Microsystems Inc, Java 2 SDK, Standard Edition, version 1.3, Summary of New Features and Enhancements, Menlo Park, CA, 2002.

[Wang 99] J. Wang, "A Proxy Server Infrastructure for Adaptive Mobile Applications," Thesis (MCS), Carleton University, Ottawa, ON, 1999.

[WAP 02] WAP Forum, "Wireless Application Protocol: White Paper," June 2000. Retrieved August 13, 2002 from http://www.wapforum.org/what/WAP_white_pages.pdf.