MobiGo: A Middleware for Seamless Mobility

Xiang Song, Umakishore Ramachandran
College of Computing
Georgia Institute of Technology, Atlanta, GA, USA
{*songx,rama*} @ *cc.gatech.edu*

Abstract

Nominally, one can expect any user of modern technology to carry a handheld device such an iPAQ or cellphone and utilize resources in the environment to remain connected and enjoy continuous services while travelling. We present a middleware infrastructure, called MobiGo, that provides seamless mobility of services among these environments. We identify three different kinds of environments (spaces)– self-owned, familiar, and totally-new and three axes for supporting mobility, namely, hard state, soft state, and I/O state in these spaces. MobiGo provides the architectural elements for efficiently managing these different states in the different spaces. Focusing on a specific demanding video service, we describe an implementation and performance results that show that MobiGo enhances user experience for seamless mobility.

1. Introduction

With the rapid advance in technology, it is becoming increasingly feasible for people to take advantage of the devices and services in the surrounding environment to remain "connected" and continuously enjoy the activity they are engaged in, be it sports, entertainment, or work. Such a ubiquitous computing environment typically consists of two components: an environment and a mobile platform, as shown in Figure 1. Users carry the mobile platform, such as a PDA or a cellphone, and enter/leave environments on the go. The environment can always provide more resources that are impossible to carry (such as a big screen), or not available on the devices (such as language translation).

The focus of our work is to provide seamless mobility in such a ubiquitous service system. For example, a user should be able to switch the sports highlights played on his/her cellphone to a high quality display that became available in the environment without discontinuity. There are three properties that a user would expect for such migration of his/her activity. First and foremost, the experience should be seamless (i.e., no discontinuity in the user experience). Second, the user should have control over the mobility (if so desired). Third, the user should be able to trust the environment.

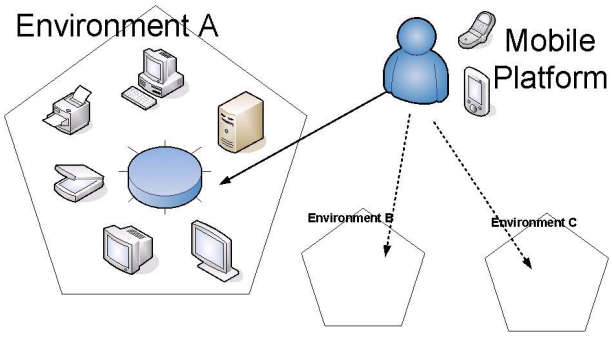

Figure 1. A Ubiquitous Service System

Seamless mobility is at the core of *service level virtualization* of a user activity that preserves the above three properties. Service level virtualization hides the details of the service instantiation (launching a new application and redirecting I/O, etc.) from the user. The goal of service virtualization is to dynamically provide users with *functionalities* (such as video playing) that become available in the environment utilizing the concrete platform specific services (such as *MediaPlayer* on Windows, and *mplayer* on Linux). Migration from one service in an environment to a similar service, with the same functionality but perhaps speaking a different protocol, in another environment is critical in achieving seamless mobility.

In this paper, we present a middleware architecture, MobiGo, for seamless mobility that offers the following features:

(1) Dynamic service discovery in the environment
(2) Dynamic determination of the best strategy for managing application states for migration
(3) A proof of concept implementation for supporting seamless mobility of video service
(4) Performance evaluation that shows the viability of MobiGo for video service migration.

In the rest of the paper, we first present salient issues for seamless mobility in section 2 and position our work in section 3. After presenting the MobiGo architecture in section 4, we introduce a ubiquitous video service as a specific example to focus our work in section 5. Section 6 describes the implementation details of the system, followed by performance results in section 7. We present concluding remarks and possible future work in section 8.

2. State Migration

Seamless mobility in a ubiquitous service system generally means state migration between environments. We identify four dimensions for state migration: what to move, where to move, when to move and how to move.

2.1. What to move

Seamless mobility boils down to migrating the dynamic state of a service from one environment to another. We recognize three categories of state associated with a service: *hard state*, *soft state*, and *I/O state*.

Hard state refers to data or preferences stored in persistent storage (such as files). It can be part of the mobile platform people carry (such as files in an Intel Personal Server [20]) or it can also be served from a repository on the Internet (such as a web server). Soft state refers to the volatile state of the service currently being accessed by the user. An example of soft state is the pause point of a movie when a user stops a movie at one location and wants to resume it at another location. I/O state refers to the current state of the input and output of the service being migrated. An I/O redirection is a form of I/O state migration.

2.2. Where to move

The motivation for seamless mobility arises from the fundamental premise that the ambient environment may have more resources to enhance the user experience than the mobile platform. We classify the ambient environment into three groups: self-owned space, totally-new space and familiar space.

In a self-owned space, a user has full control over the environment. The hardware and software configuration and settings can be customized at will. An example service in such a space is a video service that can be moved from the TV screen in the living room to the one in the bedroom.

In a totally-new space, a user has no control over the environment or any specific service. In such a scenario, dynamic discovery of new services becomes important. For example, a rest area along a highway may offer a "VCR" service on a large display. The user will be able to discover this service and avail it for watching his favorite movie from the point where he left off prior to the trip.

The familiar space is in between the above two categories. An example would be a preferred traveller's lounge in an airport or a hotel where the user has stayed previously. The user may be familiar with services in this environment but does not have total control as she does in her own home. On the other hand, the environment may allow customization of the services by user's previous preferences.

	Discovery	State Management		State Migration		Trust
		Hard	Soft & I/O	Explicit	Implicit	
Self-owned	❍	❍	●	❍	●	❍
Familiar	◗	●	●	●	◗	◗
Totally-new	●	●	●	●	●	●

Legend: ❍: less important ◗: mid important ●: more important

Table 1. Requirements for Seamless Mobility

2.3. When to move

The third dimension to seamless mobility has to do with when to migrate the service. The choice is either explicit under user control or implicit on recognizing some user cues. Typically, in a self-owned space, a user may desire implicit migration whereas a user may want explicit control in a totally-new space.

2.4. How to move

Since all the three states (hard, soft and I/O) have to be present in the local environment to perform the service, it comes to the question of how those states can be transferred. One straightforward way may be using the mobile device to store all necessary states. On the other hand, a user may only carry minimum information, such as a unique user ID, and expect the local environment to retrieve the states from his/her previous environment.

2.5. Requirement for Seamless Mobility

Based on the above discussion on the dimensions of seamless mobility, we summarize some general requirements for a ubiquitous service system below. These requirements may or may not be critical in some spaces as shown Table 1.

- Rapid discovery of services in the environment
- Efficient management of service states
- Giving user the choice of explicit or implicit control
- An intuitive model of trust for privacy and integrity

3. Related Work

Seamless mobility implies that the source and destination platforms should have something in common. Virtualization technology can easily provide interoperability

among heterogeneous platforms. Different levels of virtualization are shown in Table 2. To the best of our knowledge, no other system focuses on service level virtualization targeted in this paper.

Levels of virtualization	Example systems and related work
Service Level	MobiGo
Middleware Level	CORBA
Device Level	VNC, Sun Ray, uMiddle
Operating System/VM Level	Xen, IBM SoulPad, Microsoft Desktop on Keychain
Hardware Level	Register renaming in processors

Table 2. Virtualization levels

While not representing a truly seamless mobility solution, Microsoft Remote desktop [25], Virtual Network Computing (VNC) [4] and Sun Ray [15] represent thin-client approaches to allow a user's display to be dynamically moved to suit the user's convenience and preference through the connection to the "home" machine. . "Your Desktop on Your Keychain" [6] and IBM's SoulPad [13] remove the necessity of connecting to a "home" machine by migrating the entire VM state as well as file system to the storage devices. These approaches can be lumped into the category of OS level virtualization combining the migration of both hard and soft states for achieving mobility.

Networked File System (NFS) [2] provides users with access to files remotely by mounting the remote file system on the desktop of the user. Ubidata [5] is a project that gives an illusion to a user as if she has a big file system on her mobile platform and can potentially access any data she has wherever the files are located. CoFi [23] is another system that enables authoring multimedia content and collaborative work on mobile devices. Xmove [11] enables users to map the virtual X server to different physical X servers so that the I/O state can be migrated. Once again these solutions are appropriate for their goals and are at best complementary to our goal of seamless mobility.

HP's Cooltown project [14] gives a "web presence" to every artifact in art galleries to make it self-describing . The Gaia project [19] intelligently customizes services to meet different user's requirements by providing a "smart space" running in a fixed environment with a fixed set of devices. These projects focus on a single type of environment and doesn't target for different spaces identified in previous section.

Cyber foraging [21][22] and object offloading [9] are techniques for exploiting the resources available in the environment to increase the productivity of a mobile user. The work by Goyal and Carter allows mobile devices to install their own applications in the environment by providing a virtual operating system to each mobile client [8]. These approaches are both operating system level virtualization and are good examples of how a mobile device can take advantage of the computational facilities in the environment to hide their resource limitations. uMiddle [12] is a middleware system that supports interoperability across multiple protocol families at device level. The problems there projects address is orthogonal to what we are targeting and can complement our solution in achieving continuity of services in multiple environments.

As should be evident from the above discussion, the related work surveyed in this section have different goals from ours. Consequently, they do not address all the requirements we identified in the earlier section. Nevertheless, they represent technologies that are very relevant to the overall theme of supporting mobility in a ubiquitous computing setting. Table 3 summarizes these projects with respect to the different spaces and different kinds of states that need to be managed for service migration.

		Hard	Soft	I/O
Own	E	USB Drive SoulPad	Key-chain SoulPad	xmove
	I	NFS, Ubidata Gaia	Cyber Foraging Gaia	Gaia
Familiar	E	USB Drive SoulPad	SoulPad	VNC, Sun Ray Remote Desktop
	I	Ubidata	Cyber Foraging	–
Totally-new	E	USB Drive SoulPad	SoulPad	VNC Remote Desktop
	I	Ubidata	–	–

Legend: E - Explicit control; I - Implicit control

Table 3. Summary of related works

The granularity of migration is another interesting aspect to look at. A complete migration of virtual machine like [6] is a coarse-grained approach where users need to carry all states and resume the entire VM on the target environment. At the other extreme, a mobile disk drive carries only hard states and users rely on the software services in the target environment to open the files and manually resume the soft state (e.g., drag the time bar to the appropriate position). The former approach forces users to carry too much if they just want to move some services but not the whole desktop. The latter migrates only hard state and needs users' inputs for mobility. We believe both of them are not good enough in a ubiquitous service system wherein a user may want to *dynamically* decide on the need for a specific service. Our MobiGo system is designed to solve these seamless mobility problems at the right granularity: it migrates all states necessary to provide continuity of a particular service but no more (i.e., not entire VM). Furthermore, it also caters for the dynamic needs of a user on the go.

4. MobiGo

4.1. Architecture

MobiGo is our middleware system providing mechanisms for migrating service states to achieve seamless mobility. We should emphasize that we are not building a "smart space" that expects people to come and use the services. Instead, we are dealing with how people can *move* a

service from one place to the other seamlessly. As figure 2 shows, the key components in MobiGo are Service Control, Application Wrapper, Authentication and Service Repository.

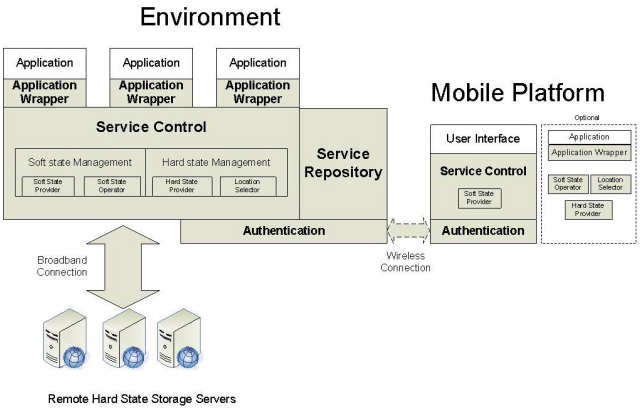

Figure 2. Architecture for MobiGo

Service Control is the core of the entire system. Its main functions are (1) collecting necessary information to launch a service (2) pausing/resuming services according to user's need and (3) collecting necessary information for later restoration of the service when user moves. Service Control contains Soft State Management and Hard State Management to help with state storage and retrieval. Soft State Provider in Soft State Management is the storage of soft states and Soft State Operator manipulate those states between services and Soft State Provider to pause and resume services. The main job of Hard State Management is to decide where to retrieve the states if it is available in multiple places since we only consider large read-only hard states currently(mainly for video service). Location Selector in Hard State Management selects the best source from a pool of available sources with various bandwidth and latency to achieve best possible performance.

Application Wrapper is a small, per-application module that makes legacy applications "controllable" by MobiGo. It can either use available API of a particular application or simulate the behavior of keyboard and mouse to send commands to applications just as if a real person is manipulating the application.

The authentication module has built-in authentication mechanisms that helps the environment to verify a user's identity and helps users to identify their surrounding environment.

Service repository stores a list of available services in the current environment for mobile users to dynamically discover and use. It listens to a designated multi-cast address and is the first place for the mobile platform to contact when a new user enters the environment.

4.2. How it works

4.2.1 Discovery and description of the services

We have our own discovery protocol between the mobile platform and environment, allowing the mobile platform to obtain service list from current environment. Each service description contains service name, service type and I/O devices associated with the service. Services with the same type are interoperable. User first chooses the desired I/O device and then selects a service on that device. She can also optionally select from a list of previous saved state to resume a paused service. In case the paused service is not available in the local environment, she can select from a list of alternative services that are interoperable with the paused service.

We use a new discovery protocol instead of existing protocol families like Bluetooth or UPnP because we target *service* level interoperability, i.e., user should be able to move between two different but interoperable services. By using a new discovery protocol, user can potentially use any *virtualized* service (e.g., movie playing) regardless of the concrete services and protocols. MobiGo moves the burden of dealing with interoperability, which may be computationally intensive for the mobile platform, to the powerful environments.

4.2.2 Choices of the user

To run a service, all three states (hard, soft and I/O) have to be present in the local environment. Whereas I/O is always local, users can choose to either explicitly or implicitly migrate the hard and soft states. Correspondingly, we consider two strategies to support user control: *carrying* and *fetching*. For carrying strategy, the mobile platform carries all necessary states to run a service. For fetching strategy, the mobile platform provides minimum information that identifies the user (e.g., a unique UID) and the environment fetches the necessary states for the user (most likely from the user's previous environment).

Fetching is desirable for implicit migration. The environment in this case should be connected and *stateful*, i.e., it has to know the user and be able to locate her previous states. However, it may be difficult, if not impossible, to use fetching strategy in some cases, especially in an isolated environment with no connection to outside. In such a scenario, carrying strategy is the best choice. Carrying requires the mobile platform to be *stateful* but the environment can be *stateless*. Carrying strategy requires the user to explicitly inform the environment when she leaves. It also takes some time to download the states to the mobile platform before she can actually depart.

Fetching strategy may be a little slower than carrying when user's previous environment is far away from the cur-

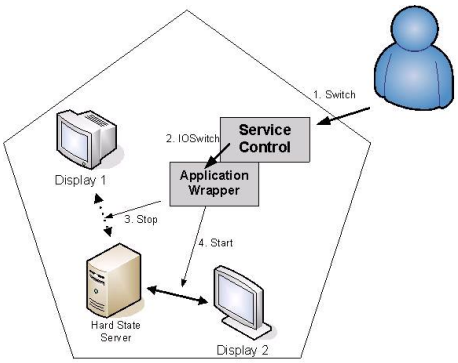

Figure 3. I/O Migration

rent one. But it releases the burden of maintaining the states from the mobile platform to the environment so that the resources on devices (such as storage and computing power) can be saved for other use. It is useful when the resources on the mobile platform are highly constrained and users don't always know when they will enter and leave the environment ahead of time. Carrying strategy requires more resources on mobile device and requires explicit notification of departure. However, it can be used more generally in most scenarios, regardless of whether the environment is standalone or connected.

4.2.3 State Management

I/O migration moves a service from one I/O device to the other in a single environment. When a mobile platform issues a *Switch* request (either explicitly or implicitly), Service Control extracts the target I/O device and send a command *IOSwitch* to the application wrapper. Application wrappers may have different techniques to switch the I/O, depending on the interface the application provides. There is no state storing/retrieving in the process, as shown in Figure 3.

Figure 4 shows the migration of soft state using carrying strategy: when a user is leaving an environment, the mobile platform sends out a *PAUSE* request. Service Control first pauses the service, retrieves the soft state and saves it to the mobile platform. When the user enters a new environment B, the mobile platform discovers available services and contacts Service Control to provide the soft state with a *RESUME* request. Service Control then interprets the soft state and instructs application wrapper to resume the execution of the service.

Figure 5 shows the hard state migration where the video file is duplicated in multiple places. Due to the typical large size of the video, it is better to find a best source (i.e., a low latency and high bandwidth connection) to stream the video, even if the video is carried in the mobile platform. The choice of file source in one environment may be different in another depending on the connectivity. Upon entering

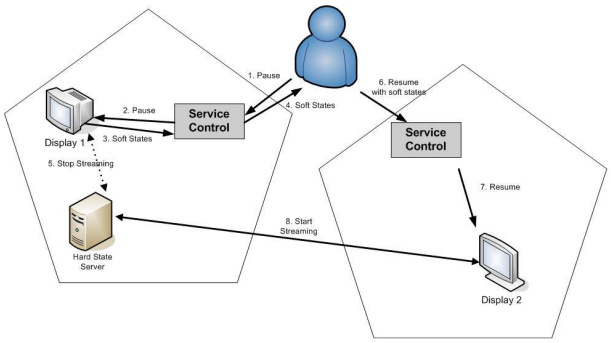

Figure 4. Soft State Migration

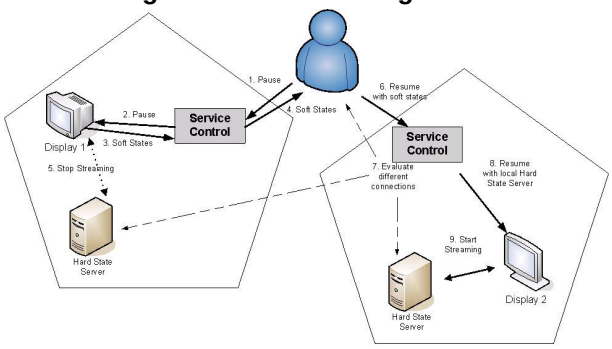

Figure 5. Hard State Migration

an environment, the mobile platform provides possible file sources and the Location Selector evaluates different connections (including wireless connection between device and environment) before choosing one to stream the video.

4.2.4 Trust

The authentication modules ensure the security and trust of the entire system. The trust should be mutual for both the mobile platform and the environment. Currently we only allow one user per service at any given time to avoid conflict. Services in use are not discoverable by new entrants of the environment. A user will get an error message if he wants to launch a service that is in his service list but currently used by another user. We plan to incorporate other people's work [27] into MobiGo to enhance the management of user's identity and trust of the environment.

5. Supported Applications

A number of applications that people use everyday can be supported by MobiGo, such as email, browsing, video playing and music listening. To make the discussion concrete, we narrow down our focus on a specific application: ubiquitous video service, which demands continuity, high quality and efficient management of state. Here is an imaginary application scenario for this service. Bob (a frequent traveller on business trips) has a collection of movies that he

Field Name	Description	Example
FILENAME	The name of the file	Terminator-II.mpg
LOCATION	An array that includes all possible sources of the file	user@video.foo.org: /share/Terminator-II.mpg
STATE	Internal soft state of the file (such as played time)	10000 ms from start of file
MODIFIED	Last modified timestamp	Oct 31. 2006, 11:09s

Table 4. Structure of Ubiquitous Virtual State

has access to from a server somewhere on the WWW. One day, he is at home in San Francisco, watching one of his favorite movies, when he suddenly gets a phone call from his company, asking him to travel to New York to meet a customer. He uses his PDA to "pause" the movie and drives to the airport. Upon arrival at the airport, he finds out that his flight is delayed by 2 hours due to "weather" in Chicago. In the Crown lounge while waiting for his flight, he "resumes" the movie where he left off on a big screen (available in the lounge). Upon boarding call, he leaves the Crown room still watching the remaining minutes of the movie on his PDA at the gate area.

To enable the above scenario, the environment first suggests through its discovery service the available video service and displays; upon explicit selection by the user, the environment implicitly migrates the soft state (pause point) from PDA, and streams the hard state (video file) either from the web server or from the user's home machine.

6. Implementation

6.1. Soft State and Hard State Management

We use a pivotal data structure, called Ubiquitous Virtual State (shown in Table 4), associated with each video to facilitate soft and hard state migration. There may be additional bookkeeping information such as owner and summary in this structure. LOCATION in UVS is used for managing hard state and STATE is for managing soft state.

UVS is created when user downloads or gets access to the video file and will be updated when user is moving around. For example, when a user buys a movie from a website, the website provide the UVS to the user and it may specify the streaming server addresses in the UVS if such service is available. Thus, upon entering a new environment, by providing UVS to the environment, user can allow the system to choose the best source to stream the video. When user leaves an environment, the STATE field is updated and will be transferred to the new environment later for resuming the service.

6.2. Application Wrapper

In current implementation, we have two application wrappers: (1)a native mplayer wrapper that simulates the input commands from standard input (stdin) to control mplayer and (2) a UPnP windows media player wrapper receiving UPnP commands. Service Control knows the types of wrapper for each application.

6.3. Location Selector

In our current implementation, we use an intuitive algorithm in the location selector to decide the best place to retrieve the hard state: it first determines the required bandwidth of streaming a video from a server based on its frame rate and resolution (e.g., a 352×240 30fps MPEG-1 video requires 1.5Mbit/s bandwidth). It then chooses the lowest latency connection among all candidates that can cater to this bandwidth requirement.

We also notice that the measurement of latency and bandwidth of connections may be time-consuming, especially for wireless link between device and environment. Therefore, the environment profiles the wireless connectivity as soon as the user is authenticated, even before she chooses to use any service. Performing this pre-pinging helps in reducing the end-to-end latency experienced by the user. The pre-ping statistic serves as a baseline for evaluating other sources of hard state specified in the UVS.

7. Performance

In order to show how the system performs quantitatively, we conduct experiments using the following set up: (1) a mobile platform: an Arm-Linux iPAQ with 802.11b wireless card (2) two environments: <A> a 4×450MHz UltraSPARC-II SMP machine with 4 GB memory running Solaris, and <B> a $2\times$3.2GHz Xeon SMP machine with 4 GB memory running Red Hat Enterprise Linux 4.0, connected through wired gigabyte network. We use a streaming video service for testing, which requires 512KBytes - 1024KBytes buffering before starting.

7.1. Migration latency

The first experiment is to measure the switching cost of three state migration discussed in section 4.2.3. I/O migration cost is the latency incurred in the application wrapper to switch displays in a single environment. The cost of soft state migration, which implies I/O migration, includes pausing the video, dumping the soft state to the mobile device, feeding the soft state to the other environment and resuming the video. Hard state migration cost, while including the soft state migration cost, also involves the link evaluation time. We use carrying strategy for the migration of soft state (1KB) and fetching strategy for hard state (512KB buffer). We can see from Figure 6 that the one-time switching cost ranges from 0.5s - 1.5s, which are acceptable from

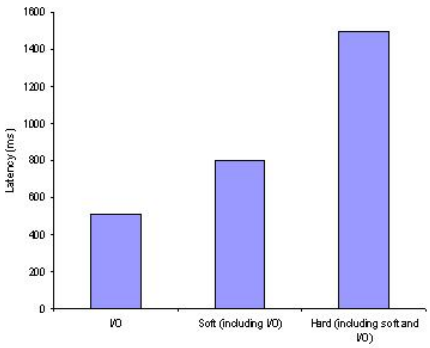

Figure 6. Migration latency

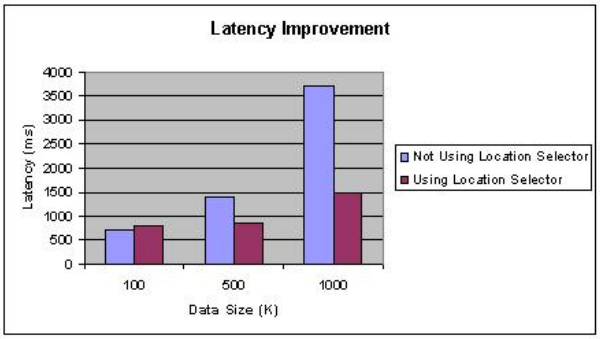

Figure 7. Improvement from location selector

user experience perspective [1].

7.2. Latency improvement by location selector

In this experiment, we measure how the location selector can help reduce the end-to-end latency. We assume that the video is available on the mobile platform as well as at a remote servers <B>. Figure 7 shows the cost of retrieving states using the location selector (light bar) or not using it (dark bar). The link evaluation cost is included in the former case but not the latter. As can be seen from the figure, for small sizes (up to 100 KB), there is no advantage for link evaluation and thus it is better not to use location selector for soft states (typically a few KB). On the other hand, there is upwards of 60% performance improvement for large file sizes (1 MB or more), which suggests we should use location selector for determining the best source for streaming the video (buffer size 512KB-1MB).

7.3. Simulating different network conditions

In this experiment, we will investigate the end-to-end latency of the migration and answer questions like "what if I deployed MobiGo in my home with only 1.5Mbit/sec connection?". The end-to-end latency is the elapsed time between pressing the SWITCH button and the resumption of the migrated service. We divide this latency into three parts: (1) MobiGo overhead to process messages, (2) player overhead to play/pause/seek the movie and (3) network latency for transferring data. The first two are relatively constant while the third one varies in different network conditions. In our current implementation, MobiGo overhead is 100-150ms for each migration and player overhead is 60-80ms for single commands (e.g., play or pause). Every migration needs 4-6 commands depending on the player's capability [2]. To simulate the network latency, we modified the MobiGo implementation to add network delays commensurate with the network bandwidths. Figure 8 shows the end-to-end latency in different bandwidth conditions[3].

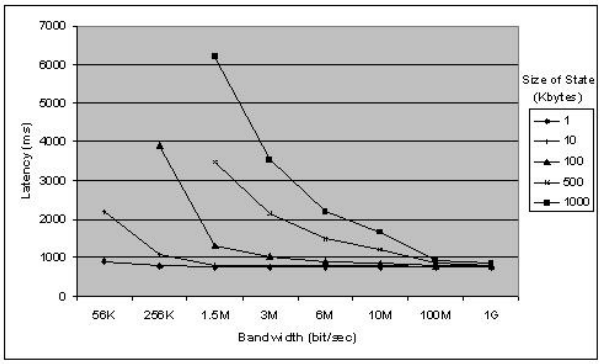

Figure 8. Migration latency affected by network bandwidth (for different sizes)

Since the fixed cost (1) and (2) are incurred in any circumstance, the influence of the network bandwidth is not significant for small downloads (e.g., soft state migration). The larger the size of the state to be migrated, the more the influence of the bandwidth on end-to-end latency. For a 1.5Mbit/sec connection like a home DSL, it may take 6-7 sec to buffer 1MByte video from a remote site. This may be acceptable in some cases but it is better if a closer state provider, perhaps a server in the home with a 100Mbit/sec Ethernet connection, is chosen for better performance. Furthermore, movie players tend to buffer more than 1 MByte for high quality movies (image size 640×480). Therefore, the importance of the location selector becomes more apparent for those cases.

Besides affecting the latency prior to starting the movie, the location selector can also ensure the user experience during the actual movie playing by selecting the source with *average* bandwidth greater than the video bitrate. It can also help instruct the player on the expected amount of buffering to be done to ensure there is no jitter during playback.

[1] We assume a 2 second latency between initiating an action and observing its effect is tolerable from a user experience standpoint

[2] Some players need a play and a pause before a seek

[3] All the latencies more than 7 seconds are removed for better view.

8. Conclusion and Future Work

In this paper, we identified the need for seamless mobility, and constructed the MobiGo system to address this need, which provides the ability to migrate a user's soft state, hard state, and I/O state between environments. By targeting a demanding video service, we show that our system is able to avail the services and resources in the environment to provide a rich user experience. Through performance evaluation, we show that our system is able to find the best way to provide continuity of services to users and the overhead is reasonable. Our future work includes considering multi-user scenarios and interactive services (like gaming) wherein consistency of the shared content becomes an issue. We will also plan to layer MobiGo on top of uMiddle [12] to leverage the device level interoperability with service level interoperability. In addition, we are exploring more sophisticated algorithms for the selection of hard state migration, as well as low latency mechanisms for ensuring security and data integrity.

Acknowledgements

The work has been funded in part by an NSF ITR grant CCR-01-21638, NSF NMI grant CCR-03-30639, NSF CPA grant CCR-05-41079, and the Georgia Tech Broadband Institute. The equipment used in the experimental studies is funded in part by an NSF Research Infrastructure award EIA-99-72872, and Intel Corp. We thank the members of the Embedded Pervasive Lab at Georgia Tech (http://wiki.cc.gatech.edu/epl/) for their helpful feedback.

References

[1] M. Satyanarayanan and et al., "Pervasive Computing: Vision and Challenges", IEEE PCM Aug. 2001, pp.10-17

[2] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh and B. Lyon, "Design and Implementation of the Sun Network Filesystem", Proc. Summer 1985 USENIX Conf.,

[3] S. R. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, T. Winograd. "ICrafter: A service framework for ubiquitous computing environments", UBICOMP 2001

[4] T. Richardson, Q. Stafford-Fraser, K. R. Wood and A. Hopper, "Virtual Network Computing", IEEE Internet Computing, volume 2, pages 33-38, 1998

[5] J. Zhang, A. Helal and J. Hammer, "UbiData: Ubiquitous Mobile File Service", Proceedings of the ACM Symposium on Applied Computing, March 2003

[6] Your Desktop on Your Keychain, `http://research.microsoft.com/research/sv/keychain/`

[7] S. Kalasapur, M. Kumar and B. Shirazi, "Seamless service composition (SeSCo) in pervasive environments", In First ACM Int. workshop on Multimedia service composition, 2005

[8] S. Goyal and J. Carter. "A lightweight secure cyber foraging infrastructure for resource-constrained devices". WMCSA 2004

[9] X. Gu, A. Messer, I. Greenberg, D. Milojicic and K. Nahrstedt, "Adaptive Offloading for Pervasive Computing", IEEE Pervasive Computing, vol.3, num.3, July, 2004

[10] A. Messer, I. Greenberg, P. Bernadat, D. Milojicic, D. Chen, T.J. Giuli and X. Gu. "Towards a Distributed Platform for Resource-Constrained Devices", ICDCS, 2002

[11] E. Solomita, J. Kempf and D. Duchamp, "XMOVE: A Pseudoserver for X Window Movement", The X Resource, volume 11, page 143–170, 1994

[12] J. Nakazawa, H. Tokuda, K. Edwards and U. Ramachandran, "A Bridging Framework for Universal Interoperability in Pervasive Systems", ICDCS 2006

[13] R. Caceres, C. Carter, C. Narayanaswami, M. T. Raghunath, "Reincarnating PCs with Portable SoulPads", Proc of ACM/USENIX MobiSys 2005

[14] HP Cooltown, `http://www.champignon.net/TimKindberg/cooltown.php`

[15] Sun Ray Deployment on Shared Networks, `http://www.sun.com/blueprints/0204/817-5490.pdf`

[16] Bluetooth Technology, `http://www.bluetooth.com`

[17] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham and R. Neugebauer, "Xen and the Art of Virtualization", SOSP 2003

[18] F. Douglis and J. K. Ousterhout, "Transparent Process Migration: Design Alternatives and the Sprite Implementation", Software - Practice and Experience, 1991

[19] Gaia project in UIUC, `http://gaia.cs.uiuc.edu/`

[20] Intel Personal Server, `http://www.intel.com/research/exploratory/personal_server.htm`

[21] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen and H. Yang. "The Case for Cyber Foraging" In the 10th ACM SIGOPS European Workshop, September 2002

[22] D. R. Krishna and et al., "Simplifying Cyber Foraging for Mobile", CMU tech report, CMU-CS-05-157

[23] E. de Lara, R. Kumar, D. S. Wallach and W. Zwaenepoel, 'Collaboration and Multimedia Authoring on Mobile Devices", MobiSys 2003

[24] R. Wolski and N. T. Spring and J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing", Future Generation Computer Systems, volume 15, pages 757-768, 1999

[25] Microsoft Remote Desktop, `http://www.microsoft.com/windowsxp/using/mobility/default.mspx`

[26] Universal Plug and Play Device Architecture, `http://www.upnp.org/`

[27] T. Woo, S. Lam, "Authentication for Distributed Systems", in IEEE Computer (January 1992) pp 39–52