A COMPUTATION PARADIGM FOR MULTIPLE MOBILE AGENT SYSTEMS

By

OGUZ KAAN ONBILGER

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2004

To
my unborn son Alpkan

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor, Dr. Randy Chow, for his guidance and support, which made this study possible and for sharing his wisdom with me. I also would like to thank my co-advisor, Dr. Richard Newman, for fruitful suggestions, and sharing his intelligence and experience. I thank both of my advisors for patiently listening to my presentations during the weekly group meetings and providing their guidance.

I also would like to thank my committee members, Dr. Jih-Kwon Peir, Dr. Abdelsalam Helal and Dr. Suleyman Tufekci, for their efforts and brainstorming to make it clear what I was trying to do and their helpful comments and suggestions. Special thanks go to Dr. Shigang Chen for providing the network simulation software he developed and his collaboration and discussions. Thanks go again to Dr. Randy Chow for encouraging this collaboration.

Very special thanks go to my parents for their continual emotional support from thousands of miles away.

Finally, I express my gratitude to my beloved wife, Derya, for her constant love, encouragement, patience, and support through difficult times. Without her, this study would never have been completed.

TABLE OF CONTENTS

page

ACKNOWLEDGMENTS iv

LIST OF TABLES viii

LIST OF FIGURES ix

ABSTRACT xi

CHAPTER

1 INTRODUCTION 1

2 BACKGROUND 7

2.1 Software Agents and Multi-Agent Systems 7
2.2 The Mobile Code Concept 9
2.3 The Mobile Agent Paradigm 11
2.3.1 Applications of Mobile Agents 13
2.3.2 Mobile Agent Systems 15
2.3.3 What Mobile Agents Offer 19
2.3.4 Drawbacks of the Mobile Agent Paradigm 22
2.4 Security Problem of Mobile Agents 23
2.4.1 Malicious Agents Problem 25
2.4.1.1 General protection mechanisms against possibly malicious mobile code 25
2.4.1.2 Specific protection mechanisms against possibly malicious agents 27
2.4.2 Malicious Hosts Problem 29

3 MULTIPLE COOPERATING MOBILE AGENTS 35

3.1 Introduction 35
3.2 Requirements and Objectives 37
3.3 Definitions 40
3.4 Mission Models 41
3.5 Multi-agent Systems (MAS) vs. Multiple Mobile Agent Systems (MMAS) 44
3.6 Proposed Architecture for MMASs 45
3.7 Example Missions 48
3.8 Related Work 52

4 REMOTE DIGITAL SIGNING WITH MOBILE AGENTS....54

4.1 Introduction....54
4.2 Electronic Commerce and Mobile Agents....54
4.3 Mobile Commerce and Mobile Agents....55
4.4 Mobile Agent Security in Mobile Commerce....55
4.5 Objectives....57
4.6 Background....58
4.7 Multiple Cryptography....61
4.8 Key Splitting and Signature Generation....63
4.8.1 Using the Multiplicative Property of RSA....64
4.8.2 Using the Additive Property of RSA....65
4.8.3 Using El Gamal Public Key Cryptosystem....66
4.8.3.1 Signing in sequence with El Gamal signature scheme....67
4.8.3.2 Signing in parallel with El Gamal signature scheme....69
4.8.3.3 Transition from El Gamal cryptosystem to digital signature algorithm....70
4.9 The Overall System for Remote Digital Signing....71
4.10 Using Limited-liability Keys and Public Key Certificates....74
4.11 Practical Issues of Remote Digital Signing....78
4.11.1 Probabilistic Signature Scheme and its Impact on Practice....79
4.11.2 Performance Considerations and the Big Picture....80

5 A NETWORK POSITIONING ARCHITECTURE FOR MOBILE AGENTS....84

5.1 Introduction....84
5.2 Related Work....85
5.3 TPNP Approach....88
5.3.1 The Algorithm....90
5.3.2 Starting the System....95
5.3.3 Scalability Issues....96
5.3.4 Security Considerations....97
5.4 Experimental Evaluation....97
5.4.1 Simulation Environment....98
5.4.2 Simulation Parameters....99
5.4.3 Simulation Strategy....99
5.4.4 Simulation Results and Analysis....100
5.5 Conclusion....104

6 QUANTITATIVE ANALYSIS OF MULTIPLE MOBILE AGENT SYSTEMS....106

6.1 Introduction....106
6.2 Network-awareness in Mobile Agent Computing....107
6.3 The Traveling Salesperson Problem....108
6.4 Application of TPNP and TSP to Classical MA Model....111
6.4.1 The Data Structure....111
6.4.2 The Heuristics....112

6.5 Experimental Evaluation 113
6.5.1 Simulation Environment and Parameters 113
6.5.2 Simulation Strategy 114
6.5.3 Results and Analysis....114
6.6 Context-awareness in MA Computing 120
6.7 Multiple Cooperating Mobile Agents Model 122
6.7.1 Trust Model 122
6.7.2 Performance Model 124
6.8 Application of TPNP to Multiple Cooperating MAs....126
6.8.1 Problem Formalization 128
6.8.2 Heuristics for MATAP 131
6.8.3 Experimental Evaluation 132
6.8.3.1 Simulation parameters....132
6.8.3.2 Simulation strategy....133
6.8.3.3 Results and analysis 133
6.9 Related Work....140
6.10 Conclusion and Future Work....142

7 SUMMARY AND FUTURE DIRECTIONS....144

REFERENCES 147

BIOGRAPHICAL SKETCH 156

LIST OF TABLES

Table page

3-1. Comparison of MAS and MMAS....44

6-1. Application of NN heuristic with TPNP....118

6-2. Application of RA heuristic with TPNP....118

6-3. Application of FRP heuristic with TPNP....118

6-4. Simulation parameters....133

LIST OF FIGURES

Figure page

2-1. Mobile agent execution model 12
2-2. Mobile agent execution platform....16
3-1. Mobile client/server model....42
3-2. Multiple mobile agents model with two mobile agents....43
3-3. Multiple mobile agent system architecture....46
4-1. A snapshot of a mission using multiple mobile agents 67
4-2. Protocol for sequential signing with multi-agent model 72
4-3. Protocol for parallel signing with multi-agent model....73
4-4. Limited-liability key protocol....77
5-1. Global network positioning model (GNP)....86
5-2. Local network positioning model (TPNP)....88
5-3. Discovery algorithm 91
5-4. Selection algorithm....92
5-5. Distance measurement protocol....93
5-6. Average relative error....101
5-7. Average local relative error 102
5-8. Average outliers relative error....102
5-9. Distribution of average relative error to path latencies 103
6-1. Performance of 2-Opted TSP heuristics across the problem size....115
6-2. Performance of 2-Opted TSP heuristics across the number of TPNP dimensions...116

6-3. Illustration of MATAP 127

6-4. Mission communication time across problem size.....135

6-5. Mission communication time across security level category 136

6-6. Mission communication time across random selection distribution over security levels.....137

6-7. Effect of transmission rates on mission communication time.....138

6-8. Heuristic performance with the available SA hosts in the network 139

6-9. Heuristic performance with the number of clusters.....140

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

A COMPUTATION PARADIGM FOR MULTIPLE MOBILE AGENT SYSTEMS

By

Oguz Kaan Onbilger

December 2004

Chair: Randy Chow
Cochair: Richard Newman
Major Department: Computer and Information Science and Engineering

Mobile agent (MA) systems have been proposed as a potentially useful computing paradigm for distributed applications. An MA is an autonomous computing entity, which migrates from one host to another carrying its code/data/state information, to accomplish a task on behalf of its owner. The concept is attractive since mobility of the agent brings execution closer to the remote resources resulting in a great reduction of communication overhead, and the autonomy property of the agent relieves the attention of the initiating owner, rendering more concurrency for the applications.

The primary hindrance that prevents MA systems from wider acceptance is the concern of MA security. Protection of agents from malicious hosts is considered a difficult problem. Our goals are to prove that sufficient degree of agent security can be achieved to make MA systems a reality, and to propose prototype architecture for *secure multiple MA* systems.

Our approach relies on the assumption that security can be enhanced through the use of multiple collaborating MAs. We conjecture that security can be strengthened through *protection as a whole* meaning that attacks to some of the agents cannot corrupt the entire group of agents performing collaboratively on a common task. One of the problems tackled in this research is the requirement of the ability to *compute with secrets in a public domain.* Our research focuses on a special case of this problem, i.e., remote digital signing with multiple agents. Multiple collaborating MAs are a natural extension (with mobility) of the research on multi-agent systems in artificial intelligence.

Central to the design of mobile systems is the ability to track the locations of migrating agents. In our proposed multiple MA system, agents need contextual information based on location to optimize their traveling itineraries. We propose a general coordinate-based peer-to-peer network-positioning scheme for the system. The design of such a location service opens up many new and interesting variations of the classical traveling salesperson problem. The location service is used in a simulation tool for analyzing the performance and communication overhead of the MA-based network.

CHAPTER 1
INTRODUCTION

The concept of software agents is a natural extension of the notion of processes in traditional operating systems. In the context of distributed systems, software agents are *autonomous* processes deployed at various sites to perform some specific functions or tasks. When *collaborations* among these agents are needed, they form a Multiple Agent System (MAS) to achieve a common goal. The design issues on MASs generally fall into two complementary research areas, distributed systems and artificial intelligence. The distributed systems research community is concerned with the system supports for the agent technologies such as synchronization and communication of interacting agents, while in the AI community, the focus is more on the intelligence and decision support of the cooperating agents. This research adds another dimension, *mobility*, to the multiple agent systems we term, Multiple Mobile Agent Systems (MMAS). In such a MMAS, mobile agents exhibit strong mobility in that they need and are allowed to dynamically migrate from host to host carrying with them their code/data/state information to accomplish their missions.

The concept of code migration or mobile code is not new. Most Internet users have experienced its impacts in both positive and negative ways. On the positive side, we have enjoyed downloaded programs such as Java applets in web browsers for all kinds of operations ranging from banking transactions to animating cartoon characters. This aspect of mobile code technology helps to realize the important goals of transparency and efficiency in distributed systems. On the negative side, we have seen numerous malicious

uses of the technology in the forms of viruses and worms that disrupt our daily computer operations. Similarly, Java servlets, which are the server counterpart of applets, have become commonplace for many Internet applications. Like applets, their code migration is static, meaning that codes are uploaded to servers, run there and cease their existence. The emerging mobile agent technology takes the applet and servlet concepts one step further to allow dynamic and continuous migration of the agent processes autonomously.

Mobile agents have been proposed and used in some limited application domains such as e-commerce, information retrieval, network management, software distribution, distance education, and even as a security mechanism for intrusion detection. It has been argued that using mobile agents as a programming paradigm is a perfect fit for distributed systems since the autonomous characteristics and mobility of agents facilitate naturally the implementation of asynchronous and dynamic interaction of concurrent execution in distributed applications. More precisely, the mobility of agents brings execution closer to the remote resources resulting in great reduction of communication overhead, and the autonomy property of agents relieves the attention of the initiating agent owners rendering more concurrency for the applications. However, the concept has received much skepticism until the current millennium, out of concern that interoperability and security might not be justified if agents are to roam freely in an open and large-scale network. This research addresses the security issue of mobile agents. In particular, it claims that utilizing multiple collaborating mobile agents can actually enhance the security of the applications.

There are two separate categories of threats in mobile agent security, malicious agents and malicious hosts. Protection of hosts against malicious agents is well

understood and has been implemented at least in some degree in most systems. Common solution approaches to the problem are host hardening and agent containment such as the built-in security mechanisms for byte-code execution in Java language and virtual machine. The protection of mobile agents against malicious hosts, however, is less intuitive. First, the assumption that a user would send its agent processes to untrustworthy hosts is less conventional. Second, the secrecy and integrity of the agents and the need to execute the code in public might conflict with each other at times. Total agent protection is considered unsolvable without special hardware by some researchers. The main difficulty comes not only from agent mobility but also autonomy. Especially, their ability to migrate from one host to another poses vulnerabilities, which could easily be exploited by the very host platforms they execute on. This dissertation focuses on the problem of protecting mobile agents against malicious hosts.

There are threats to mobile agents which lead to very simple attacks that could be devised: denial-of-service and replay attacks. A host platform which receives an agent may simply not execute the agent at its own discretion, which leads to denial of service. Similarly, a host platform can reuse the agent by using the credentials delegated to it by its owner (i.e., user) to masquerade the original host and user, which leads to replay attacks. These two types of attacks are not properly addressed by the known or proposed mechanisms. A complete description of attacks possible against mobile agents and a background information about proposed solutions and their classification are given in Chapter 2.

We have identified several requirements for a solution for the malicious hosts problem. These are summarized as follows:

1. A solution proposed to protect hosts from possibly malicious hosts should not jeopardize the protection of hosts from agents,
2. The second important requirement is that solutions should not limit the potential benefits of mobile agents,
3. It is vital to address easier attacks against mobile agents, which are denial-of-service and replay,
4. The concept of the network trusted computing base needs to be extended to be applications and case specific.

Details of these requirements are provided in Chapter 3.

The theme of this dissertation is to address the protection of mobile agents from possibly malicious behavior of host platforms by keeping in mind the requirements pointed out above. The proposed technique is based on an autonomous group of multiple agents designed to accomplish a single well-defined task, which in turn relies on the concept of information dispersal for security. Information dispersal is not a new concept. Both cryptographic and non-cryptographic means of protection via the technique have been proposed and implemented. Cryptographic mechanisms are mostly related with key sharing and threshold cryptography. Non-cryptographic means are directly applied on data to be protected. For example, sensitive information is split up and distributed to several hosts connected with a computer network. Penetration to one or some of the hosts would not reveal the information, and intrusion to several hosts is considered more difficult than intrusion into a single host. Similarly, sensitive information carried by mobile agents (this may also include the code) can be split up and given to multiple agents. Therefore a successful attack against the group of agents is more difficult than attacking a single agent. Agents in the group should be located on different hosts, and furthermore on different administrative domains. Agents in the group should

communicate and cooperate to accomplish the task given. Therefore, tampering with one of the agents could be detected by the others in the group.

The feasibility of the described approach however does not only depend on information dispersal. To apply the concept into dynamic mobile agents, we first need to figure out how to position multiple mobile agents, specifically how to find the hosts to locate the agents in a feasible manner. This opens up very interesting research issues including making mobile agent systems aware of the underlying network topology. While we are focusing on one specific application of information dispersal in Chapter 3, our focus in Chapter 4 is on the network positioning models to be able to apply information dispersal into multiple mobile agents in an open internetwork environment.

This dissertation is organized as follows.

In Chapter 2, we provide a brief description about software agents and multi-agent systems, mobile code concept and its applications, mobile agents, their benefits and drawbacks. Then we provide background information about mobile agent security from both perspectives described above and proposed solutions and their categories in more detail.

In Chapter 3, we explain the concept of group of multiple cooperating agents for security. We provide models applicable to this concept and our proposed model. We define the mission concept and model and distinguish this concept from the multi-agent systems. The proposed multiple mobile agent system architecture is also introduced. Chapter 3 is the foundation for the rest of this dissertation.

In Chapter 4, we address the remote digital signing problem of mobile agents. For the mobile agent paradigm to be widely accepted and used, one must demonstrate that

mobile agents are capable of performing computations that are possible with the client/server model. Digitally signing a contract is one such computation in e-commerce applications that needs to be performed by mobile agents. We demonstrate that it is feasible to perform this computation with multiple cooperating agents using the information dispersal for security concept, as presented in Chapter 3.

In Chapter 5, we examine the network distance measurement approaches and propose a general, highly scalable coordinate-based peer-to-peer network-positioning scheme, one that fits better to the requirements of the mobile agent systems. By using a simulation tool, we analyze and demonstrate the achieved accuracy of the approach.

In Chapter 6, we apply the network positioning system proposed in Chapter 5 together with traveling salesperson problem heuristics to traditional mobile agent computing and multiple mobile agent systems. We are looking for the answers to the questions about whether we can alleviate the performance problem of multiple mobile agent systems and if we can do it efficiently without canceling out the performance benefit achieved. This chapter also deals with network- and context-awareness in mobile agent systems.

Finally, Chapter 7 summarizes our work and provides future research directions.

CHAPTER 2
BACKGROUND

Mobile agent paradigm has two descendants: software agents and mobile code. Mobile agents are software agents with the added feature of mobility. On the other hand what makes them mobile is the general concept of mobile code. So, mobile agents combine these two unrelated concepts, and become a unique technology in distributed systems. In this chapter we provide a brief outline of software agents in general and then have a look at the mobile code concept and its applications. The rest of the chapter is dedicated to mobile agents, their applications, systems, standards, pros and cons. A detailed survey of the security problem of this technology and the proposed solutions ends the chapter.

2.1 Software Agents and Multi-Agent Systems

Software agents are an active research area in Artificial Intelligence (AI) and distributed computing systems. In this section we use the term “software agent” to refer to software entities that are subject to research in these two fields, in general. So, autonomous agents, intelligent agents, interface agents, virtual agents, information agents, and mobile agents are all software agents.

The difficulty with the definition of an agent in the AI community is to distinguish a software agent from a software program (Franklin & Gaesser, 1997). Bradshaw (1997, pp. 7) cites the definition given by Shoham (1997), with the hope that it might be acceptable to most researchers: “A software entity which functions continuously and autonomously in a particular environment, often inhabited by other agents and

processes." Murch and Johnson (1998, pp. 16) provide a survey of definitions and as a result they conclude that "agents are, by consensus, autonomous, goal seeking, persistent, reasoning, productive, and communicative." So, it seems like characterization of agents makes more sense than giving an exact definition. Bradshaw (1997, pp. 8) shares the same point and as the result of his own survey concludes that "consistent with the requirements of a particular problem, each agent might possess to a greater or lesser degree attributes like the ones: reactivity, autonomy, collaborative behavior, communication ability, inferential capability, temporal continuity, personality, adaptivity, and mobility."

From the distributed computing systems perspective, however, our focus is on mobile agents, which is a specialization of generic software agents with the added ability of mobility. The details on mobile agents are given in Section 2.3.

A precise description of multi-agent systems is given by d'Inverno and Luck (2001, pp. 6):

> Multi-agent systems are typically distributed systems in which several distinct components, each of which is an independent problem-solving agent come together to form some coherent whole. There is generally no pre-established architecture or configuration incorporating the agents, and the interactions between them are not pre-defined, as is usually the case with traditional processes in concurrent programs. More importantly, there is no global system goal, the agents being heterogeneous with their own goals and capabilities. In consequence, agents in a multi-agent system need to coordinate their activities and cooperate with each other, in order to avoid duplication of effort, to avoid unwittingly hindering other agents in achieving goals, and to exploit other agents' capabilities.

What is not clearly specified in the above description is that owner or users of the agents participating in a multi-agent system may be different parties.

It is necessary to distinguish the multi-agent systems and the multiple-agent model we use in our research for better understanding. This distinction is made clear in Chapter 3 where we define multiple cooperating agents concept.

2.2 The Mobile Code Concept

The mobile code concept may be best understood by examining the applications. In the following we provide brief descriptions of mobile code applications.

In traditional distributed computing systems, applications are built on the client/server paradigm. Usually, the client process sends a request to the server process over the network. The server processes the request and returns the result to the client. The communication between the clients and servers are handled through message passing or Remote Procedure Calls (RPC). This is a synchronous process in the sense that clients usually block after sending the request until the reply arrives or a certain timeout is observed. Although asynchronous RPC changes the classical synchronous communication nature of the client/server model, the request/reply scheme remains the same. The mobile code concept however makes a paradigm shift in distributed computing.

In contrast to the request/reply scheme, in mobile code systems, not only the data but also the code to be executed is exchanged between the applications. Remote evaluation has been first proposed by Stamos and Gifford (1990). In this scheme, code to be executed is sent to the remote machine. The result of execution is sent back as data only.

As is true for other resources, processor cycles can also be shared in distributed systems. The mechanism necessary for sharing such a static component requires code migration in the form of processes, which is known as process migration. The first

objective of process migration in distributed systems is to redistribute load and improving performance by migrating processes from node to node. The secondary objective is to achieve location and performance transparencies by distributed process scheduling (Chow & Johnson, 1997).

Perhaps the most well-known mobile code mechanism that is widely used today is Java applets. An applet is a Java object that is associated with a web page. When the page is requested by a client browser, applet is downloaded into the client machine along with the web page and executed in the Java Virtual Machine installed on the same machine. Applets are static in the sense that they cannot move anywhere other than the client machine, Moreover, their communication to the outside world is restricted to the originating server machine due to security concerns.

The server counterpart of applets is an object known as a servlet in Java. So, servlets are uploaded from the client to the server to make the server functionality customized according to the specific requirements of the client. Code migration, therefore, follows the opposite path of an applet. Servlets are widely used today by the web servers, which are implemented in Java. However, servlets in use today are created and used by the server host, mostly due to the security concerns. Therefore, the capability of code migration, which was explained above, is not used. Java applets and servlets are said to be examples of code-on-demand paradigm (Lange & Oshima, 1998).

Active networks are an active research area employing the mobile code concept (Hartman et al., 1996; Tennenhouse et al., 1997; Tennenhouse & Wetherall, 1996). The area has its roots in the SOFTNET project, which is a multihop packet radio network developed in the 1980s (Zander & Forchheimer, 1988). In active networks, specialized

packets, which are called capsules (Tennenhouse & Wetherall, 1996), carry not only data but also code. The code is injected into the nodes along the route of the capsule. Subsequent packets which are related to the original capsule trigger the execution of the code previously stored in the nodes. The result is that the routers of the network and therefore the network itself are programmed by its users. Immediate advantages of the described scheme are that customized computation for individual applications can be carried out inside the network and new network protocols can be installed on the routers easily, on-the-fly.

2.3 The Mobile Agent Paradigm

Mobile agents (MAs) are an approach to distributed computing employing the mobile code concept. MAs are autonomous entities, which are composed of code, data and state information. They visit hosts (e.g., servers) possibly using an itinerary, perform some execution on those hosts using their codes and migrate with their state information from host to host. As in the case with stationary agents, they act on behalf of their owners (e.g., senders). They are autonomous in the sense that they have all the knowledge needed to perform the assigned task on behalf of their owners. The traditional mobile agent execution model is illustrated in Figure 2-1. The mobile agent is launched from the originating platform by the owner. It visits the hosts to accomplish its task and return to the owner.

The most important difference between the MA paradigm and the other mobile code systems is the strong mobility of MAs. None of the mobile code systems and mechanisms except the process migration mentioned in the previous section has the strong mobility feature. This means that the execution state of the transmitted mobile code is not carried. Therefore, the code (e.g., program, procedure or method) starts

execution from the beginning and terminates at a certain point. In the case of mobile agents, execution stops but not terminates and resumes at the next platform, where the mobile agent migrates.

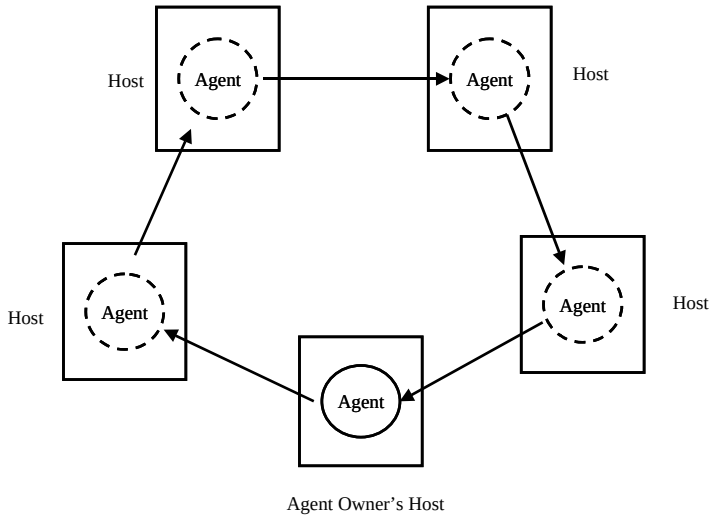


Figure 2-1. Mobile agent execution model

Migrating processes are not allowed to choose where and when to migrate in contrast to mobile agents. While processes are migrated to balance the load transparently, mobile agents autonomously migrate to hosts where the information or services are of interest. Another difference between the two is that processes in general can migrate among homogeneous environments whereas mobile agents are designed to run on the middleware, which makes them platform (machine and operating system) independent.

Active network architecture is mainly proposed for network level tasks, although application level customization is also possible. Mobile agents, in contrast, are mostly to be used at the application level. While the target hosts are the network routers in the former, target of mobile agents can be any host. One last distinguishing feature is that mobile agents have execution state, whereas an active capsule has no such state. However, active networks and mobile agent systems are not mutually exclusive. Active

network architectures and models, which employ mobile agents, have been proposed (Breugst & Magedanz, 1998; Busse et al., 1999).

2.3.1 Applications of Mobile Agents

Mobile agents have been proposed to be useful in a wide range of applications. We classify these applications in the following and give a brief insight about the application areas.

E-commerce, m-commerce. Among many application areas of MAs, e-commerce draws the most attention from both academic and industrial researchers (Busch et al., 1998; Marques et al., 1999; Klusch, 1999; Roth, 2000; Sandholm, 2000). This is mostly due to the fact that, MAs and in general agent systems have the capability of representing users (i.e., customers) in the electronic marketplaces. Agents can effectively profile user preferences, act on behalf their owners, participate in e-auctions, watch stock prices, search for commodities and find the best offer from competing vendors, purchase goods by paying and committing to transactions, communicate and cooperate with other agents of relevant goals.

Network management. Code mobility in the form of mobile agents has been proposed by many researchers as a promising technology in network management. Some examples are Baldi, & Picco, 1998, Papavassiliou et al. 2002, and Meer et al., 2000. Baldi and Picco (1998) model network management activities using client/server model, remote evaluation and mobile agents and summarize advantages of using mobile code in network applications as semantic compression of information, a higher abstraction level to the network manager and autonomy of management agents.

Information retrieval. The information retrieval problem is getting more difficult everyday on the Internet due to the amount of information available to the users. Mobile

agents can search, filter and retrieve vast amount of information locally. Therefore information retrieval has become one of the most important and popular application areas of the mobile agent paradigm. Brewington et al. (1999) provides a qualitative and quantitative analysis of the use of mobile agents in information retrieval applications. Johansen (1998) discusses pros and cons of mobile agents in different application areas and provides quantitative analysis of real applications. One of these applications is StormCast where several servers receive weather data from meteorological censors. Users in turn, by sending their mobile agents can learn about the weather conditions in their area, and even set alarm conditions which are watched by their agents. Agents react by notifying the their users when the condition is occurred.

Software distribution. Mobile agents are capable of customizing server side functionality by moving the client specific code to the server and this capability is one of most important benefits of the mobile agent paradigm. This concept therefore can even be used to deliver software for permanent use of server and client computers. Using this push model, subscribed code consumers may enjoy immediate software installation and updates. In fact, active networks are a good example of using mobile code in the form of specialized packets to install new protocols to the network routers, on-the-fly. An example of software update application using mobile agents is given by Bettini et al. (2002).

Other application areas. Other example specific application areas are distant/remote education (Jamwal & Iyer, 2003), and vehicular traffic management (Schill et al., 1998). Mobile agents have also been proposed as alternatives to provide

security in distributed systems. An example is intrusion detection systems, which utilize mobile agents (Jansen et al., 2000).

2.3.2 Mobile Agent Systems

Informally a mobile agent system is a middleware where mobile agents live. This middleware typically sits on top of a runtime environment. The runtime environment is responsible for running the mobile agents and in some cases also for the execution of the mobile agent system itself as in the case of Java systems. The instances of these middleware systems, which run on hosts are known as places (Lange & Oshima, 1998) or platforms. We prefer the term “platform”. So, the mobile agents are created, executed and terminated in these platforms. When we say that mobile agents migrate between hosts, what we actually mean is that they migrate between the platforms, running on different hosts connected through a network. The mobile agent systems are responsible for all the operations of mobile agents. Creation, migration, communication with other systems and agents, cloning, security and termination are all handled by the underlying platform.

Figure 2-2 illustrates the mobile agent system concept and relationships with the other components of a host, which employs a mobile agent platform. The runtime environment is typically a high-level language interpreter or Java virtual machine, since vast majority of mobile agent systems today are built using the Java programming language. Java has been the common ground for mobile agent systems, mostly due its standard, object-orientedness, support of mechanisms such as serialization and reflection, support of distributed objects as in RMI, and inherent security features. However, because different mobile agent systems themselves and mobile agents are implemented in the same programming language and executed by the same standard Java virtual machine

does not mean that these systems would be interoperable. More details on interoperability problem are given in Section 2.3.4.

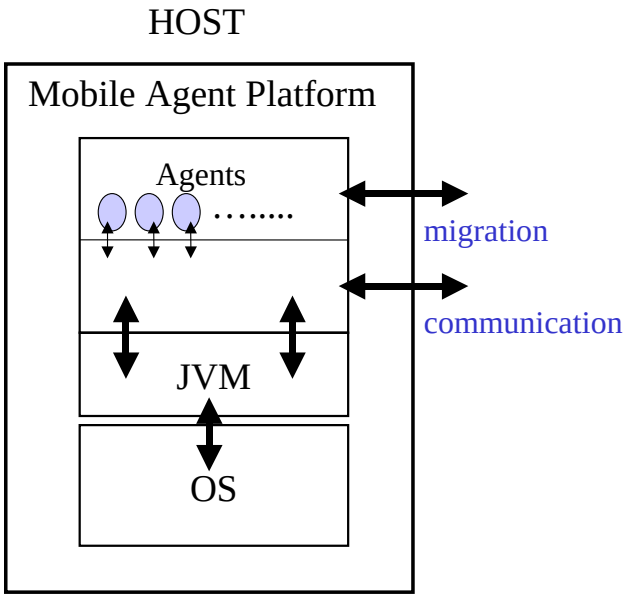


Figure 2-2. Mobile agent execution platform

Today, there are more than fifty mobile agent systems. It is difficult to keep track of all the mobile agent systems exist today around the world. Some of the earlier ones ceased to exist as they become unsuccessful in the market or some of them have been experimental research systems. However, many new projects have been started to develop new systems which usually to overcome the deficiencies of the existing or previous systems.

Kiniry and Zimmerman (1997) provide a survey of both commercial and research based Java mobile agent systems. Brewington et al. (1999) classify the systems as multiple language, Java based or others which are written mostly in script languages, and compare the systems. Lange and Oshima (1998) give a brief description of popular mobile agent systems. Karnik (1998) classify mobile agent system according to several

criteria including the security features provided by these systems. We classify the mobile agent systems as in the following.

Programming language used. Most of the latest mobile agent systems are developed in Java. Hence, mobile agents themselves are implemented in the same language. Earliest systems however have been written in scripting languages or interpreted bytecodes in other high level languages.

Migration state carried by the agents. There are two different approaches for mobile agent migration. Within the first one control state of the mobile agent is captured and send to the target machine. The execution resumes in the target where it left off in the source machine. Within the second approach, the execution state of the mobile agent is not sent to the target machine. The programmer is responsible for the agent state and the execution state resume in the target host by referring to an entry point. The second approach makes it simpler to develop the underlying system but it may be a burden to the application programmer to capture the current state of the agents. But this is not really a problem because migration decision is made solely by the mobile agent itself by executing an instruction such as *go*. Since, migration is not forced by the underlying system, entry points can be easily specified using the data portion of the agent rather than the control state which includes the execution stack. This approach is taken by many of the Java based systems for interoperability since Java does not support inspection of the execution state of the objects due to security concerns. The former approach requires modifications to the JVM to capture the state and hence mobile agent systems and mobile agents developed using these systems can not run on standard JVMs.

Multi-threading of mobile agent systems and mobile agents. Many Java mobile agent systems run the agents as a separate thread of the main process of the mobile agent platform. Others run agents in isolated processes and some of the systems use a combination of the two approaches. Ara is an example of the former approach whereas D'Agents is for the latter.

Communication primitives. Different mobile agent systems use different communication primitives among mobile agents. Some uses message passing, or RMI. Some systems require the agents to be present at the same platform, some other provide remote communications between sites. As examples, Aglets support sending and receiving messages in the form of objects, Ajanta supports RMI using proxies and D'Agents use message passing.

Security provided. This is perhaps the most important classification from our point of view. Unfortunately, vast majority of the mobile agent system have been developed without any security consideration at all. This is perhaps due to the fact that these systems are not intended for large scale, open systems or networks such as the Internet. In a company network or intranet, security may not be that important for mobile agent systems. Karnik (1998) makes the distinction among mobile agent systems in three categories: secure communication, server resource protection and agent protection. Secure communication is about whether the agent migration is protected using encryption and authentication. Server resource protection is usually associated with authorization of agents and access control mechanisms such as ACLs. Agent protection is related with the malicious hosts problem. Agents must be protected against any possible hostile behavior of hosts where they are executed. This is an open research area, which is the subject of

this dissertation and only a limited number of agent systems support mechanisms within limited contexts.

2.3.3 What Mobile Agents Offer

Lange and Oshima (1998) point out that mobile agents: 1. reduce the network load, 2. overcome network latency, 3. encapsulate protocols, 4. execute asynchronously and autonomously, 5. adapt dynamically, 6. are naturally heterogeneous, and 7. are robust and fault-tolerant. Gray et al. (1996) examines the benefits of mobile agents from the mobile computing perspective. They point out the capability of mobile agents which provides a powerful alternative for partially connected computing through mobile computers (i.e., laptops, personal digital assistants, etc.) and home computers, which have intermittent connections to the fixed networks. They also highlight how mobile agents simplifies the development, testing and deployment of distributed applications, and how it is possible to employ a scalable, peer-to-peer architecture for distributed applications instead of the rigid client/server model.

Chess et al. (1995b) in their classical paper in the area collect fourteen claims, which have been given as the advantages of mobile agents over existing client/server paradigm (i.e., message passing, RPC, etc.) and they examine each claim in detail. They conclude that none of the claimed application areas could be uniquely addressed by the mobile agent paradigm (one exception might be the real-time applications which could be impaired with the network latencies between hosts that run client and server components of such a distributed application, especially in wide-area networks). For every application area, they could find an alternative that may well be addresses by the existing technologies. However, as the final conclusion they point out that mobile agent technology is unique in the sense that there is no alternative technology that could

address all of the application areas and benefits together that can be offered easily by the mobile agents.

Johansen (1998) shares their experiences of implementing and using mobile agents in real world applications using the Tacoma mobile agent system and also provides quantitative analysis of the performance aspects of mobile agents. He concludes that 1. agents are a convenient way to install client software at remote hosts, 2. their most successful application would be to build extensible servers, 3. the asynchronous nature of agents is important, 4. agents are shown to be a very convenient structuring technique for the distributed applications.

We briefly summarize the benefits, which are offered by the mobile agent paradigm as follows.

Mobile computing point of view. Mobile devices have limited battery life, and their intermittent connection to the wireline networks have low bandwidth, and high latency. Mobile agents overcome all these problems. They can be launched from mobile devices in a brief connection to the fixed network. Returning results to the users requires only another brief connection. During the computation the device can be disconnected from the network, and can even be turned off.

Networking point of view. Mobile agents are capable of searching, analyzing and filtering huge amounts of information available on the servers across the Internet. As a result they can only return the most relevant information to their interested users. This saves network bandwidth when compared to the existing client/server model of computing where all the analysis and filtering are performed in the client side which requires transferring all irrelevant information along with the most relevant parts.

Service point of view. Services offered by servers can be customized according to the user requirements, on-the-fly. Servers can offer basic primitives to use their services. Using these simple primitives, mobile agents can be programmed to reach the services and information offered by the servers in a customized way. The alternative used today is to provide service packages, which are offered to clients. Offering a new service requires a long implementation and deployment process. Therefore, users have to choose from these offered service packages. Mobile agents make the deployment process immediate.

Application development point of view. Even it is not a real problem theoretically, client/server based application development poses the problem of a design issue of at which component the client or server, the services should be implemented. Mobile agents overcome this problem easily. Client side functionally can easily be moved to the server. Mobile agents, in general, simplifies the implementation, testing and deployment of distributed applications (Brewington et al., 1999; Johansen, 1998). Interestingly enough, they can even be used to develop and deploy prototype distributed applications (Chess et al., 1995a).

Application user point of view. Not only mobile agents, but in general software agents have the capability of representing their users and user preferences, which is an important benefit especially in electronic commerce and mobile commerce applications.

Distributed system point of view. Most distributed applications fit naturally to the mobile agent model (Brewington et al., 1999), mostly because of the mobility of the agents. This is the aggregate advantage and hence becomes the unique property of mobile agents.

As a conclusion we believe that mobile agents possess an enormous potential in building feature distributed systems, once their drawbacks are addressed convincingly, as explained in the following sections.

2.3.4 Drawbacks of the Mobile Agent Paradigm

Two major open problems, which prevent mobile agent paradigm from being a widespread, actively used technology today, are security and interoperability. The security problem of mobile agents is addressed in the next section. Here, we provide brief information about the interoperability problem.

As mentioned in Section 2.3.2 mobile agents require their own mobile agent platforms to be executed. For example, an Aglet cannot be executed in an Ajanta platform even though these two mobile agent platforms and agents themselves were implemented in Java and all they need is a standard Java virtual machine. Interoperability has several aspects such as capability of communication between the different systems via message passing or RPC, capability of running mobile agents from different systems, or communication between agents from different systems.

Recently, Brazier et al. (2002) proposed generative migration of mobile agents among heterogeneous platforms. This approach is based on agent factories and blueprints of agents. Blueprints, which could be describe with a high-level specification language, describe the agent functionality. Agents' state is described by a language such as XML. Agents' blueprints along with their state are migrated between host platforms where agent factories exist. The job of the agent factory is to transform the blueprint and the state into an executable form using libraries designed for this purpose and for the target environment, which exists on the same host. Although the approach seems to be promising, it raises several questions. First one is the security issue. While it is already

difficult to protect agents from malicious hosts, this approach makes it worse. For example, the approach renders the agent protection schemes completely useless which are based on obfuscation methods. The other concern is performance overhead introduced with the approach to transform agents back and forth.

Another approach is to let agents migrate only between their own platforms. If and when they need information available on some host, which does not employ the same platform that the agent needs, the agent migrates to the nearest host to the target one, with a platform needed for the agent. Then, assuming standard client/server interfaces are employed between the hosts, agent becomes a client and communicate with the target host using a request/reply scheme. This scenario is equivalent to the one with a target host, which does not support any mobile agent platform as in (Theilmann & Rothermel, 2000). The major problem with this approach is related with performance, which is to find the nearest available hosts to the target host. This issue is a subject of this dissertation. Moreover, making mobile agents dependant on the client/server model limits their capability to customize server functionality.

2.4 Security Problem of Mobile Agents

In contrast to many other technologies in distributed systems, mobile agent technology will not be accepted and widely deployed before the security problem is solved. Other technologies have been widely deployed before their security requirements have been understood and mechanisms have been implemented and used.

There are two types of security threats introduced by mobile code systems. One comes from potential malicious agents and the other from malicious hosts. The malicious agents problem is rather old and many protection mechanisms have already been proposed and implemented because of the fact that some kind of protection mechanisms

are common in mobile agent systems and other mobile code systems. On the other hand, the malicious hosts problem is more difficult and considered unsolvable without dedicated hardware. The problem is relatively new since the computation in the form of mobile agents has never been defined to take place in a remote environment that could possibly be malicious. We are concerned with the latter problem in this dissertation: protecting the MAs from malicious hosts.

Chess (1998) identifies the assumptions made on the security of computing systems in four categories: identity assumption, origin of programs, origin of attacks, immobility of programs. These assumptions state that programs and their users can be easily identified; programs are obtained from easily identifiable and trusted sources. Significant security threats come from attackers running programs with a clear intent in a restricted environment. Programs rarely cross administrative boundaries and only in controlled ways, programs run entirely on one machine on a specific operating system and operating system is responsible for the security. Unfortunately, these assumptions do not hold for mobile agents and in general, mobile code systems. Therefore new security mechanisms are necessary especially in the mobile agents case.

The protection of hosts from potential malicious agents and the protection of agents against potentially malicious hosts are not mutually exclusive. There are two reasons for this. First one is that, in general, solutions proposed for the former problem implies also the protection of hosts. This is due to the fact that, a successful attack against a mobile agent in a given platform may threaten the subsequent platforms, which the MA visits. For example a brainwashed shopping agent may request for hundred airline tickets instead of the correct value of two. Similarly, an agent responsible for software updates

may introduce Trojan horses to the subsequent hosts it visits, if a former platform could fabricate this code inside the software patches carried by the agent. The second reason is that some protection mechanisms, namely obfuscation techniques may render host security useless, since the intent of the agent might not be easily checked or verified by the platforms, which are to execute the agents.

2.4.1 Malicious Agents Problem

Hosts, which are to accept and execute MAs must be protected against possibly malicious MAs. A malicious agent may access sensitive information on the host, which is not authorized to do. Moreover, MAs may alter, fabricate or delete data or files, and may inject malicious behavior in the form of Trojan horses.

We have a brief look at mechanisms to protect hosts from two perspectives. The first one is the general mechanisms, which apply to almost all mobile code systems hence to MAs. The second category is specific to MA security, which necessary due to strong mobility and multiple execution platforms that a given MA may need to migrate and run.

2.4.1.1 General protection mechanisms against possibly malicious mobile code

Sandboxes and playgrounds. Sandbox is a restricted execution environment where all the foreign code (i.e., downloaded from another machine) is to be executed. In Java, the Java Virtual Machine executes foreign code in a sandbox, which is complemented by a security manager, which checks the instructions of the code to verify if they adheres to the security policies (Oaks, 1999). For example, foreign code has restricted access to the file system, main memory and other assets of the computer system. Moreover, in the case of Java applets for example, their communication to the outside world is also restricted to the host from where they are downloaded. Since the code inside the sandbox needs to be interpreted in order to be checked, code

interpretation is regarded as a part of the sandbox (Tanenbaum & Van Steen, 2002). However, in general, there exist code interpreter systems that do not employ the sandbox concept. Therefore, we consider this mechanism as a separate one in the following.

Malkhi and Reiter (2000) extended the sandbox mechanism into playgrounds. Basically, the functionality is the same, however, playgrounds are to be placed in physically isolated machines. Foreign code is downloaded and executed only in these machines. The local programs can access the migrated code and data through traditional mechanisms. However, foreign code cannot access to the other computers and local assets in them.

Interpreting code. Code interpretation is an important concept in mobile code systems for interoperability of heterogeneous systems. The code, which is not compiled directly into the machine instructions but instead compiled into some intermediate code (i.e., bytecodes in Java) can be run by standard middleware systems on top of the underlying specific operating systems. The best known example is the standard Java Virtual Machine, which can run almost on any operating system exist today.

A nice side effect of code interpretation is the ability to easily check the code before execution during runtime. Therefore, each instruction can be inspected to figure out whether there is any access violation or any behavior, which does not conform to the security policies defined.

Code signing. It is important in mobile code systems to authenticate the source of the code, which is migrated or downloaded. Digital signatures (through public key cryptography) can effectively be used for this purpose. The code is signed by the manufacturer (i.e., programmer) and/or the owner (i.e., user of a MA) by using their

private keys. The receiver uses the corresponding public key to authenticate the source of the code. If the source is considered to be trusted then the foreign code can be assumed safe and executed. Even in theory this works well, in practice it is difficult to assess the trustedness of the source and the level of trust that could be associated with the source.

2.4.1.2 Specific protection mechanisms against possibly malicious agents

Here, we provide an overview of the mechanisms proposed for the host protection against malicious agents. These mechanisms, mostly try to circumvent the issues arise from the nature of strong mobility of MAs which is a direct result and necessity of visiting multiple host platforms to accomplish a task.

Path histories. The idea of path histories (Chess et al., 1995a; Ordille, 1996) is to record and carry authentication information of previously visited hosts, with an MA. Each host platform digitally signs and inserts its own identity and the next platform to the history using digital signatures. Meanwhile, every host, which receives an agent checks the validity of the signatures and decides by looking at the previous hosts, whether it would be safe to accept and execute the MA. If, for example, any of the identities of the previous hosts is considered to be untrusted, the MA is discarded. The drawback of the approach is that the payload of the MA increases at the every platform visited, and it is required to check the whole path of digital signatures carried by the agent.

State appraisal. The state appraisal mechanism is to detect, to a certain extent, whether the current state of a MA, when arrived to a new site, is not harmfully altered. The agent code producer and the user of the agent provide appraisal functions for the state of the agent. This function is carried by the agent along with its code and state. When an agent arrives a new host, it must decide what specific privileges it will need at the host. The state appraisal function is used to compute a set of privileges to request as a

function of the current agent state. In turn, the authorization mechanism employed by the agent platform determines which privileges requested by the agent it is willing to grant and whether the agent is in a safe state (e.g., no harmful modifications have been made) (Farmer et al., 1996). The authors indicate however that, it may not always be possible to detect a deceptive state from a correct one.

Proof carrying code. This is a technique, which is used by the hosts to verify that foreign code provided by an untrusted party adheres to a predefined set of rules, which is known as safety policy. This policy is formed by the hosts to guarantee that the downloaded or migrated foreign code, if complies with the policy, will be safe to execute. Two components are used with the technique: a formal proof and a proof validator. The code producer creates a formal safety proof, which expresses the fact that the code will behave according to the safety policy. The code consumer host uses the proof validator, to check whether the proof is valid and therefore the code is safe to execute (Lee & Necula, 1997). The authors indicate that there are four necessary components for the technique: 1. a formal specification language used to express the safety policy, 2. a formal semantics of the language used by the untrusted code, 3. a language used to express the proofs, and 4. an algorithm to validate the proofs. Although the approach is theoretically sound, there might be practical difficulties in implementing and using the approach including a standard formalism for establishing the safety policy, automated assistance for generation of proofs, and techniques to limit the potentially large size of proofs (Jansen, 2001). In addition, the technique is tied to the hardware and operating environment, which is in conflict with the interoperability requirements of MA systems.

2.4.2 Malicious Hosts Problem

As we have pointed out above, protecting mobile agents from possibly malicious hosts in open environments is one of the most difficult security problems in distributed systems. This problem has even considered unsolvable without dedicated hardware. The obvious reason for this is that an agent in under complete control of the host it is to run on. Hohl (1998a) identified the specific threats to mobile agents:

- Spying out and/or manipulating code,
- Spying out and/or manipulating data,
- Spying out and/or manipulating control flow,
- Incorrect execution of code,
- Masquerading as a host,
- Denial of execution,
- Spying out and manipulation of interaction with other agents,
- Returning wrong results to system calls issued by an agent.

Hohl (1998b) also provided a set of requirements for an attack model against mobile agents. He uses the Random Access Stored Program (RASP) machines to show that the components of the execution process of an agent program can be accessed by an attack program and this program can be executed by another machine (in the abstract sense) to control the execution of the agent program.

There are several approaches proposed in the literature. Different classifications of these approaches can be given. For example, some approaches are aimed only to detect certain attacks whereas others try to prevent them. While some of the approaches are more general, majority of the proposed solutions target specific threats. Although we do not give a precise taxonomy, a classification and a brief description of the proposed solutions are presented below.

Organizational or social trust. Most of the MA systems ignore security by the assumption of organizational or social trust. For example, an individual user may have trust to a reputable well-known company. Therefore he/she may not expect any hostile behavior from the company that operates the MA system, against his/her agent. But this cannot be applied to a virtually unknown company, and such an e-commerce MA system may not be fair to businesses that have not yet established a public trust. On the other hand, social trust does not apply to a business-to-business e-commerce system based on MAs. For example two competing companies could not assume the same trust to one another as in the individual user case, even if these were reputable companies. This potential aspect of lack of trust could severely limit the use of MAs in an open e-commerce environment. However, we believe that together with other technical security mechanisms, this aspect will be helpful in providing better security for MAs.

Solutions based on obfuscation. Blackbox security is an obfuscation scheme, which does not rely on cryptography. It defines the problem as to make an agent's code and data be messed up so that cannot be read or modified at any time (Hohl, 1998a). Because there is no known solution to the problem, the "at any time" requirement is relaxed to "for some known time interval." The idea is to scramble code and data of an agent so that they do not reveal the function of the code. The interesting aspect of this proposal is that it does not rely on cryptography. There are some issues to be solved by the approach such as necessity of synchronized clocks to be able to realize time limitation. Code obfuscation is a well-known method and there are many examples especially for Java. However, there are also tools to defeat known obfuscation methods and this is an arms race as indicated in (Dyer, 1997).

Another approach for mobile agent protection relies on cipherprogram concept (Sander & Tschudin, 1998). It is claimed that mobile agents do not have to rely on cleartext data, program or messages. This, in turn, relies on computing with encrypted functions and computing with encrypted data. Encrypted functions work as follows. Suppose Alice has an algorithm to compute a function *f* on data item *x*. Bob has the computation power and would like to compute *f* for Alice. However Alice does not want Bob to know anything about *f*. If *f* can be encrypted in a way that another function *E(f)* can be computed by a program namely *P(E(f))*, then Alice sends this program to Bob, after execution Bob sends the result back to Alice. Alice, in turn decrypt the result to obtain *f(x)*.

Although it is not claimed to be a general solution to agent protection and the computation is limited to certain functions (e.g., polynomials), this approach is a good example for a software-based solution based on cryptography. However, recently Barak et al. (2001) have shown that this approach is not likely to succeed.

Tracing data state and execution. Partial Result Authentication Code (PRAC) has been proposed by Yee (1999). The goal is the technique is to ensure forward integrity using cryptographic checksums formed using symmetric cryptography similar to Message Authentication Codes (MAC). Forward integrity means that results of the previously visited hosts should not be able to be forged by a malicious host subsequently visited. The technique requires maintaining or generating keys for every host visited, with the assumption that the hosts will destroy the keys after the computation is complete. This raises concerns especially if the agent needs to revisits a previous host. PRACs are aimed to provide integrity rather than confidentiality. Karjoth et al. (1998) improved the

technique using digital signatures to create a chain of results obtained from the hosts visited.

Young and Yung (1997) proposed the sliding encryption technique to deal with the small size data usually obtained from hosts by mobile agents when compared to the size of the encryption keys and resulting ciphertext. The technique is to ensure confidentiality with public key cryptography accumulating the encrypted data in each platform visited.

Another approach is called cryptographic traces (Vigna, 1998), which aims at detecting any kind of tampering with a mobile agent. It relies on code execution verification using traces based on cryptography. This technique requires each host visited by a mobile agent to create a trace of the execution of agent. This trace is both maintained in the host and forwarded to the other hosts subsequently visited by the agent. The major concern is the size of the trace, which needs to be carried with the agents. This is another unauthorized modification detection technique.

Hardware-based solutions. Hardware support is recognized as the only tractable solution since no single software solution proposed so far addresses every possible attack. One example is the Tamper-proof Environment (Wilhelm et al., 1998), which is a regular computer with some specialized OS, manufactured only by authorized well-known parties. Sites that offer services to mobile agents purchase these computers and advertise this. In turn agents execute only in these isolated places by interacting with the hosts on those sites by well-defined secure interfaces. However it might still possible to devise some attacks and the solution does not seem feasible because of its special requirements. A similar approach has also been taken by Yee using secure coprocessors (Yee, 1999).

Environmental key generation. Environmental Key Generation approach (Riordan & Schneier, 1998) relies on some interesting observation that agents can find some keys whose protection is crucial only after they are on the host on which they will execute. A key can be sent to an environment such as a news list or a mailbox. The idea is to not carry the keys but to know how to find them. After agents find their keys in this manner they can do their jobs with some protection. An interesting example is given as a search agent that needs to search remote databases without revealing what it looks for. The solution is nothing more than using a cryptographic hash function that is applied to the items in the database and checking the results against the already hashed value carried with the agent. Because it is still vulnerable to denial-of-service attacks by changing the value carried by agent this approach should be used together with some other solution to prevent the kind of attacks mentioned.

Multiple (cooperating) agents. The idea of using more than a single agent for fault-tolerance and security has also been studied by some authors. However this approach has not been attracted deserved attention. Since the theme of this dissertation is in this category, we provide related previous work in Chapter 3 in more detail.

Other approaches. Another approach protects the computation of agents by relying on trusted third parties (Corradi et al., 1999): some sites that offer a trusted environment for mobile agents to perform secure operations. Agents need to visit such a site after computing in some untrusted host. Another example (Marques et al., 1999) assumes a “neutral trusted” host for e-commerce applications. But it is not clear in these proposals, why and how untrusted hosts guarantee to send the agents to so-called trusted servers to compute with secrets.

Meadows (1997) proposed the detecting objects idea, which is originally proposed for database integrity against unauthorized modification. In this scheme, some dummy objects are inserted into the mobile agents. For example, a shopping agent can be provided by some dummy offers as if they were coming from legitimate merchants. By carefully selecting these objects, the owner platform can check the agent whether these dummy objects have been tampered with. If there is no modification to these dummy objects, then with some degree of confidence it can be said that there have been no modifications to the other parts of the agents. However, this scheme is highly application specific. We extended the idea to detection objects and code (Onbilger et al., 2001). In this scheme, some dummy code fragments can be inserted into the agent code together with dummy objects. Depending on the degree of protection desired, dense of injection can vary for different applications. Since, it is intractable to distinguish dummy code fragments from the original ones, execution results of the dummy code can be checked against previously recorded results and it can be detected whether the code is executed correctly. By combining the technique with multiple cooperating agents, detection can be made immediate, without requiring the agent to return to the originating platform.

General discussion and survey papers for mobile agent security are given by Tschudin (1999), Jansen (2001), Claessens et al. (2003), and Oppliger (1999). Some flaws in the security protocols, which have been proposed before and summarized above are identified by Roth (2001).

CHAPTER 3
MULTIPLE COOPERATING MOBILE AGENTS

3.1 Introduction

Information Dispersal is a technique used for fault- and intrusion-tolerance of information. The study in this area consists of three phases. In the first phase, Shamir (1979) showed how to construct a robust key management scheme for cryptographic systems. In this scheme, a secret *S* is divided into *n* pieces in such a way that *S* can easily be constructed from any of the *k* pieces, but the knowledge of *k-1* or fewer pieces reveals no information about the secret *S*. This is called a *(k, n)* threshold scheme. The remarkable aspect of the scheme is that *k-1* pieces give no information about *S*, so it is applicable to relatively small data such as secrets. However, this scheme is not space efficient, consequently it is not suitable for large data such as a file. Rabin (1989) showed how to disperse a file into pieces and to use a subset of pieces to reconstruct the file later, in a space efficient manner. In this scheme, a file *F* of length *L* is divided into *k* pieces each of length *L/m*. The file can be reconstructed from any *m* pieces. The sum of the lengths of pieces is *L(k/m)*. Because *k/m* can be chosen to be close to 1, the scheme is space efficient.

The second phase includes the techniques of Fragmented Data Processing (FDP) (Fray & Fabre, 1991) and Fragmentation-Redundancy-Scattering (FRS) (Fabre et al., 1994). The common goal in this phase, in addition to the concept of information dispersal for security in the first phase, is the processing of sensitive information. FDP is designed to employ parallel processing techniques to process fragmented and scattered pieces of

data. Sensitive information (both code and data) is fragmented into pieces in the trusted part of a distributed system and the pieces are located on hosts in the untrusted part of the system. The code fragments are constructed using a pre-processor and compiler so that when collaboratively applied by the hosts on the corresponding pieces, the result becomes exactly the same with the result of the original code applied to the original data.

Similarly, the goal of the FRS is to employ several hosts to provide fault-tolerance of the systems and intrusion-tolerance against deliberate faults. The fragmentation process may consist of several iterations. At each iteration, application objects, which consist of both code and data are decomposed into more elementary objects if there is still identifiable confidential information exists. Redundancy is achieved by either applying techniques such as checkpointing and synchronization or relying on detection mechanisms and voting protocols implemented over underlying multicast communication system. Scattering phase consists of assigning the fragmented and redundant elementary objects to the hosts in the system. The goal is to assign the elementary objects in such a way that the objects that are assigned to the same host would not reveal any confidential information.

We consider our work as part of the third phase in this field. There are however important differences between the third phase and the former ones. FRS and FDP utilize a distributed static infrastructure to provide tolerance. In our case the target environment is already distributed and extremely dynamic. The assumption with those techniques that there are fixed available hosts in a close environment does not hold for the MA problem. FRS merely relies on a spatial technique (replication) but we need to consider both spatial and temporal solutions for efficiency. Also, in addition to Shamir and Rabin’s work we

also need mechanisms not only to distribute the secrets but also to be able to compute with them again in a distributed fashion. These differences inevitably add new challenges to the known problems and solutions.

3.2 Requirements and Objectives

There are mainly four requirements to meet in providing security to mobile agents as mentioned in Chapter 1. The first one is that a solution proposed to protect hosts from possibly malicious hosts should not jeopardize the protection of hosts from agents. Some proposed solutions such as obfuscation mechanisms might not be feasible since they may cause hosts to be vulnerable to attacks by hostile mobile agents. The second important requirement is that solutions should not limit the potential benefits of mobile agents, which are otherwise observed and enjoyed. While security is an important requirement for the mobile agent technology to be accepted and widely used, it should not sacrifice the benefits we gain from using them. One such property that is usually ignored by the proposed solutions is the autonomy of mobile agents. Since it is one of the distinguished features of them, sacrificing autonomy would severely limit the their applicability to the wide range of possible applications. For example, an MA may be required to communicate with its owner's host to perform some security sensitive operations. This violates the autonomy property of the agents, which constitute the basis of disconnected operation, which is a highly desirable mode of functioning in m-commerce. Another example is to allow mobile agents to execute only on trusted hosts and then limit the use of mobile agents into the client/server model to access the untrusted hosts. While this model provides security it sacrifice the benefit of customizing the computations on servers, which is another important feature of mobile agents. The third one is related to one of the fundamental principles of security: when there are easier ways of defeating a

system, an attacker would not try to penetrate to the system through well-protected components. So, it is vital to address easier attacks against mobile agents, which are denial-of-service and replay. The last requirement is extending the network trusted computing base concept. This concept has been proposed as a solution without giving clues about how they could be realized. Simply assigning some dedicated hosts for this purpose does not seem practical. At least, these hosts could be single points of failure and targets of attacks. Moreover, locations of these hosts should also be considered.

Under the requirements given above, the goal is to make individual MAs "meaningless" when they are treated as a single entity as much as possible since there is a trade-off between openness and security. This makes them resistant against malicious behavior of the hosts where they execute. A given task is split up into two or more MAs. These MAs are located in different hosts and migrate when necessary. They exchange information and partial results of execution at certain synchronization points. So, the approach presented here is based on information (data and code) dispersal supported by known cryptographic techniques.

In a mobile agent system there are three general security objectives:

1. Data confidentiality,
2. Correct code execution and data integrity,
3. Code confidentiality.

Data confidentiality is the easiest-to-achieve goal in our approach especially if the data needs to be kept confidential from certain hosts that possibly could gain advantage from knowing it. It is possible to place sensitive information (i.e., credit card) in an agent encrypted with a nonce and place the nonce in a cooperating agent. Other more complex means are also possible.

Data integrity can be achieved through the idea of detection objects, which has been proposed by Meadows (1997) to detect possible modifications to MA data. Correct code execution along with data integrity can be provided by extending the detection objects idea and introducing detection code. Predetermined but random code fragments are injected into the original code of the agent (possibly by also introducing dummy data). These code fragments can be produced by a tool and the modified program can be guaranteed to give the exact same result as the original when executed. At certain synchronization points, agents exchange the results of the modified program together with the results of the original program. Since the dummy code results have to be fixed, another agent can easily check them against the precalculated values. Together with realizing time limitations and dense injection (i.e., one line of dummy code for every original line) it can be guaranteed that the original code was executed correctly.

Although a human can detect which portions of code are dummy by analyzing the code, it is an undecidable problem for machines especially if data flow analysis is prevented by mixing the data flow from the original code to the dummy code (e.g., y=y*x; z=y; z=z/x; y=z; y and z are dummy, x is not). Since the concern here is instant correct execution of code and time for that execution can be limited to a few seconds, an analysis by an human is prohibited. The approach presented is equivalent to randomized state information that could be maintained and exchanged among agents.

Code confidentiality is the most difficult of these problems. The difficulty of the problem comes from the fact that it is not possible to limit the time for analysis; therefore human analysis attacks are possible. Due to the difficulty of the problem, one might

restrict the confidentiality of code into certain decision functions (e.g., a shopping agent's purchase decision) and apply the code injection technique to those functions before splitting. In addition to arbitrary code generated by a tool randomly, other mechanisms can be applied. For example, code can be generated which consists of similar statements to the original code. Also, code libraries can be used to provide code that implements some relevant or irrelevant function to the original function.

The three goals above are the generalization of broad range of diverse security requirements of the MAs. There are certain situations where combinations of the three goals above overlap. One such requirement in a mobile e-commerce environment is computing with secrets in an untrusted environment. Any such secret (e.g., a private key) cannot be revealed to any third party since the information here is more sensitive. For example, people reveal their credit card information to buy lunch but not their social security number (even if asked). So, it is crucial to have the ability not only to keep certain data secret but also to compute with that data. Digital contract signing (Sander & Tschudin, 1998) is such an example and Chapter 4 demonstrates how to do this computation remotely by a group of agents without divulging secrets or inventing new cryptographic algorithms.

3.3 Definitions

The problem of protecting MAs against malicious hosts comes from the fact that they are, by definition, autonomous. Since this aspect of MAs is the most important among others and actually it is what make them special, it would not be a good idea to give it up for the sake of making them secure. But if we define "autonomy" not for a single MA but a group of communicating and cooperating MAs, which rely only on each

other to perform a single task to achieve a goal, we will be able to reach autonomous secure MA groups.

Definition. *Autonomous MMAs.* A group of mobile agents is said to be *autonomous* if they together have the knowledge necessary to perform a single task, and they communicate and cooperate to perform that well-defined task.

What follows is the definition of the mission concept.

Definition. *Mission (traditional single mobile agent case).* A mission consists of a mobile agent, a set of hosts to be visited, execution of the agent's code on these hosts and the set of migrations of the agent between any pair of these hosts.

Definition. *Mission (multiple mobile agents case).* A mission consists of an autonomous group of mobile agents, a set of hosts to be visited, execution of the agents' code on these hosts, a set of migrations of the agents between any pair of these hosts and communication among the agents.

The term mission is the counterpart of the term *session* in client/server computing. A mission can represent any session that carries out a computation such as a database search, a network management activity or an e-commerce task, etc., using MAs.

3.4 Mission Models

Single mobile agent model. This basic model is illustrated in Figure 2-1 in Chapter 2. The mission must be accomplished by visiting several hosts, which requires process migration from host to host. The agent computes (e.g., is being executed) in these hosts and returns home at the end of a successful mission in this case. Note that the illustration in Figure 2-1 is a simplified generic case of a mission. It may not be necessary for a mobile agent to return home and a same host might be visited multiple times in the same mission.

Mobile client/server model. Depending on the mission given, one or more of the following reasons might prevent placing an agent on a service provider host:

- The server may not have a mobile agent platform,
- The mobile agent platform supported might be a different one,
- The host may not be trusted or may not provide any measure to appraise trust on.

Therefore in this model the mobile agent is located on a host, which provides the necessary environment and is close to the target host. The model is illustrated in Figure 3-1.

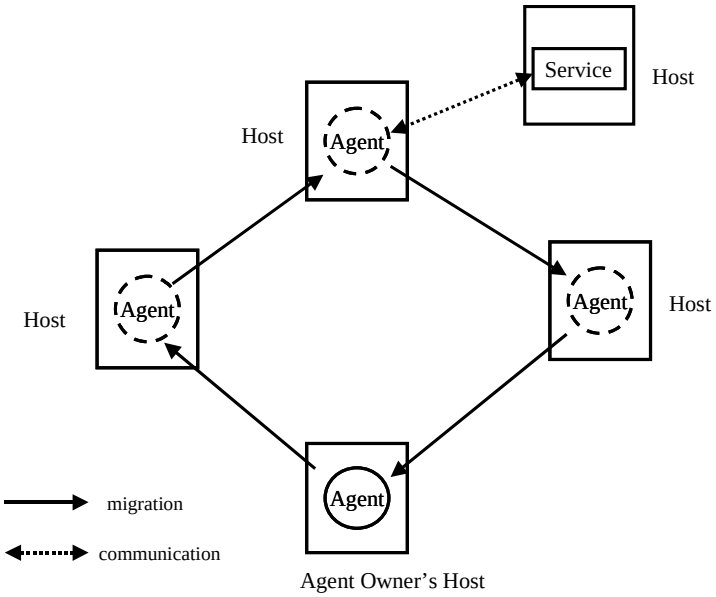


Figure 3-1. Mobile client/server model

This model is not suitable for applications where it is necessary to customize certain server functionality. If the model is used for security purposes, it will not meet the second requirement mentioned in Chapter 1.

Multiple mobile agents model. The multi-agent paradigm fits well with the concept of *protection of an application as a whole*. It is more difficult to compromise a task if the task is split into multiple collaborating agents. In the context of data secrecy, this is also referred to as *information dispersal for security*.

The multi-agent model does not differ from the classical agent model in terms of the definition of the mission. The difference is due to the definition of the autonomy property of MAs. So, in the new model, MAs are autonomous as a group but not as an individual entity. The group of agents carries out a single task by communicating and cooperating.

Figure 3-2 illustrates the model. In the model, we call one of the agents the *master agent* (Alice in the figure), who actually visits the set of hosts that are necessary to complete the mission. This set of hosts is called the itinerary of the mission. The itinerary may or may not specify the order of the hosts to be visited. The other agents are called *support agents* (Bob in the figure), who visit only the hosts outside of the itinerary of the master agent. Note that, the model that we describe here is the most generic multi-agent model.

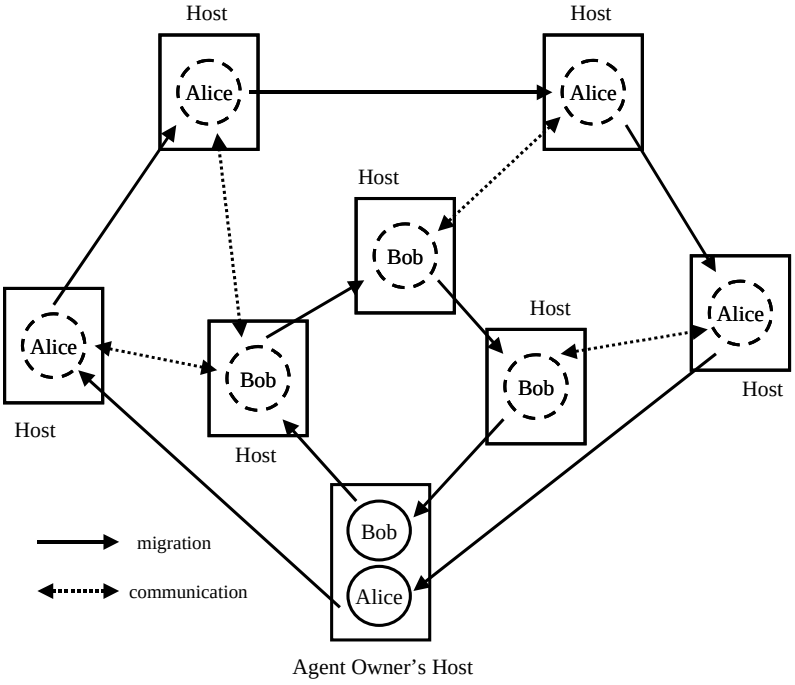


Figure 3-2. Multiple mobile agents model with two mobile agents

Variations of the model admit more than one master agent, or alternatively, a peer-to-peer architecture can be employed. In the figure, Alice and Bob, migrate according to their itineraries and communicate with each other.

As it is clear, the migration of an agent or agents in a group is more complicated in this model and requires support from an underlying system that is aware of the underlying network topology. This issue is addressed in Chapters 5 and 6.

3.5 Multi-agent Systems (MAS) vs. Multiple Mobile Agent Systems (MMAS)

In Chapter 2, we have defined multi-agent systems and said that multiple mobile agent systems differ from those. Following table highlights the differences between the two.

Table 3-1. Comparison of MAS and MMAS

Property	**Multi-agent Systems**	**Multiple Cooperating Mobile Agents**
Interactions	Not pre-defined	Pre-defined
Global system goal	Not exist	Exist
Autonomy w.r.t. goal	Yes	No
Single owner	No	Yes
Main goal	Easy, efficient, intelligent and distributed problem solving	Combine MAS and MA Paradigm MA Protection

However, it should be noted that the above properties of multiple cooperating agents are internal to their group. There is nothing to prevent a group of multiple cooperating agents from participating in any multi-agent system. So, from a broader point of view multiple cooperating agents can perfectly become a part of any multi-agent system.

3.6 Proposed Architecture for MMASs

In this Section we present the proposed system architecture of MMASs. We will explain the architecture by providing the step-by-step workflow of the system as illustrated in Figure 3-3.

On the host side we show the components/modules of an MMAS. Note that, the Mission Planer, Mission Optimizer, Security, and the Context components can be interpreted and implemented as part of the MMAS. However, we show them as separate components in the figure for the sake of clarity.

On the network side the double-line boxes represent systems rather than modules. These systems, except the Directory (a.k.a. Yellow Pages) system are distributed systems. The network positioning system is a peer-to-peer distributed system, which provides relative positions of the participating hosts via coordinates in a geometric system. The proposed architecture for this system is given in Chapter 5. The DNS is the Domain Name System. The anticipated role of the DNS will be clear in Chapter 5. The Directory system can be either a centralized or a distributed system. Although it is not addressed in this dissertation, we believe that it needs to be a distributed system. The primary responsibility of this system is to maintain and provide the information about the hosts, which employ a MA System and the services provided. The workflow and the responsibilities of the components in the architecture are given below.

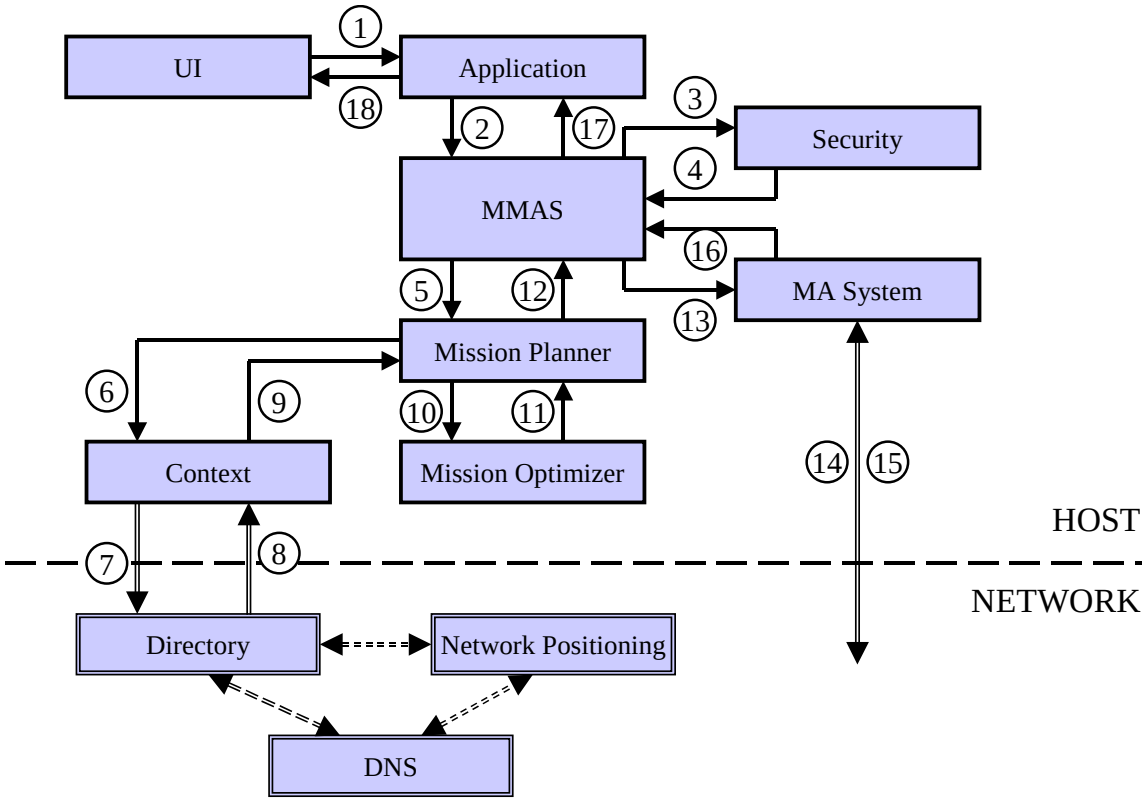


Figure 3-3. Multiple mobile agent system architecture

Step 1: The *Application* receives the information from the user through the *User Interface*. Two example applications are provided in the next Section.

Step 2: The Application presents the information to the *Multiple Mobile Agent System (MMAS).*

Steps 3 & 4: The MMAS gives the related information to the *Security* module. This module is responsible for determining the security level required for this instance of the Application and deciding the number of agents and the necessary communication among them. The module returns this information to the MMAS. Note that, apart from the workflow presented, the MMAS and the Security modules can interact to create the group of mobile agents necessary for the mission by observing the necessary dispersal of information, code and data, at the later steps in the workflow.

Step 5: MMAS presents the information provided by the Application and the Security module to the *Mission Planner*. This module is responsible for planning the mission by coordinating the other support modules.

Steps 6 & 7: The Planner module first asks the *Context* module for information about where and how this particular instance of the application could take place as a mission. The context module in turn communicates with the *Directory* service to obtain contextual and network/topology information.

Steps 8 & 9: The mission planner receives the contextual information.

Steps 10 & 11: The planner then asks the *Mission Optimizer* to compute the best possible itineraries for each agent by taking into account their interactions during the mission, in terms of the total execution time of the mission.

Step 12: The planner presents the optimized mission information to the MMAS.

Step 13: MMAS using all the information provided by the modules creates the mobile agents to accomplish the mission and passes the agents to the *MA System*. Any existing MA System can be used here, however one might expect some modifications on the existing system for integration/interoperation of the MMAS and the MA System.

Steps 14 & 15: The MA System, just as in the single mobile agent applications send out the agents to the network. Mobile agent(s) return to the MA System with the result.

Steps 16, 17 & 18: The result of the mission is presented to the user.

Following chapters of this dissertation deals with the research issues of Security, Mission Optimizer and Context components as well as the Network Positioning System. More precisely, Chapter 4 is related with the Security component. Chapter 6 addresses

the issues with the Mission Optimizer component and also makes it clear how context-awareness is an important concept in MMASs and their security. Some issues with the Context component and a proposed architecture for a network positioning system are given in Chapter 5.

Due to the lack of the Directory component, which is out of scope of this study, we assume its existence, however we make no assumptions about the use of contextual concepts. The TPNP system (Chapter 5) is capable of providing this information as a peer-to-peer system. A directory system will only need to cooperate with the TPNP to obtain contextual (including location) information.

3.7 Example Missions

In this Section we present two real-world examples of multiple mobile agent applications based on the system architecture given in the previous Section.

Example 1. *Shopping Agent.* The shopping agent is a classical example of mobile agent applications in e-commerce (See Chapter 2). The job of the agent or the group of agents is to visit on-line merchants of the product of interest, negotiate and finally purchase the product by making the payment. We assume that the user would like to purchase flowers.

The user presents the details of preferences for this purchase (e.g., a dozen red roses, under $10, with a gift card) to the shopping agent applications using the user-interface provided by this application. The user either indicates the security requirements specific for this mission or the application uses a preference file to use default or existing set of requirements for this kind of missions. The application passes this information to the MMAS possibly in XML format. MMAS consults the Security module to determine the mechanisms needed to apply to be able to meet the security requirements of the user.

For example, the purchase may require a digital signature to be generated and presented to the merchant, which is the subject of Chapter 4. We assume that the Security module decides to use a master/support model with two agents; Alice and Bob from Section 3.4. MMAS passes all the information about the mission to the Mission Planner.

The planner needs first the context information, which consists of the potential merchant hosts to be visited by Alice, the set of hosts of which a subset would form Bob's itinerary, their trustworthiness and locations. This is the job of the Context module in the architecture and this module contacts the Directory system for this purpose. The directory system executes a query given the input as the transaction of purchasing flowers from potential merchants. The result of the query appears to be a list of merchant hosts on the Internet. This part is exactly the same as of the classical single mobile agent applications. The directory service will also provide additional information about these hosts. This information needs to be the level of security provided by these hosts, the hosts that Bob will be visiting, the level of security provided by these support hosts and location information of all the hosts. The network positioning system, TPNP is capable of providing all these information, which is the subject of Chapter 5.

The directory system returns the information about merchants and potential support hosts to the Context module. The mission planner at this point has two issues to consider, which are to find best possible itinerary for Alice to visit merchants and the selection a subset of hosts returned by the Context module for Bob. The mission planner may consult the user for choice of possible merchants or may use a file to determine the preferred merchants. The planner then consult the Mission Optimizer module to provide the order of hosts to be visited to make the mission as short as possible. In addition, the mission

optimizer is responsible for choosing the closest support agent hosts to a subset of merchant hosts. The result of this selection is the itinerary of Bob including the order of hosts. This is the subject of Chapter 6. When the mission planner returns its decisions, the MMAS is ready to create the agents for the flower mission and then pass them to the MA system to be forwarded to the network.

Example 2. *Software Distribution.* Our second example is from a relatively new application area of mobile agent technology. The scenario is as follows. A software vendor has a product running on their customer sites around the world. The company continuously improves its products and provides updated versions to the customers. They also need to provide their customers with software patches for newly found security flaws in their products.

The company figures out that mobile agent technology is a viable alternative for this problem, that is using a push model with mobile agents is rather powerful than the pull model in client-server computing. They also know that security is still a drawback of mobile agent technology. But they also are aware that a group of cooperating multiple agents provide the level of security they need. So, they develop a multiple mobile agent system following the architecture presented in the previous section and ship the MA execution environment part of this product to their customer sites along with their application software.

When a new security patch is ready to be installed for a recently discovered flaw in their application software, they launch agents to the Internet to update their customer systems. They organize their software in well-designed small modules so that when they

need to update some module, their agents will only carry that small module as the payload.

We will not repeat the whole process for this application but we will provide only the differences from the previous example. In software distribution, the number of sites to be visited are expected to be much larger than the e-commerce applications. So, several agents or group of agents may need to be deployed according to their locations. Since the customers are well known by the company, the directory system's main responsibility of providing host information is not useful. However, context and network information is still quite important.

This application may use the master/support agent model as well as the peer-to-peer model. The former may be preferred for small number of customer sites. The latter is more suitable when the number of sites is large. In this case, agents need to act both as a master agent to do the primary job of software distribution and checking the integrity of peers for security. The alternative is to use the former model with multiple master agents, and make the support agent responsible for checking all of them.

We assume that the security component decides to use a master/support agent model with two agents, Alice and Bob. The software module to be installed is split up by the MMAS with the help from security and mission optimizer components. Security component makes sure that the payload of either agent would not reveal essential information about the module. Mission optimizer's responsibility is to balance the load of agents so that the cost of the mission will be minimized in terms of agent migration and communication.

Both Alice and Bob have the credentials to check the integrity of the payload of each other. On every host Alice visits, she contacts Bob and sends the payload integrity check to Bob. Bob verifies the code as well as the identification of host on which Alice is running. If all checks are successful, Bob sends Alice the part of patch he is carrying. Alice after checking the Bob's information installs the new module in the customer system and migrates to the new site in her itinerary. Bob, if necessary also migrates to the next host in his itinerary. Neither agent in this application needs to return home after the mission completes.

3.8 Related Work

Multiple agents have first been used by Minsky et al. (1996) for fault-tolerant distributed computing with MAs. The proposed scheme is deploying clones of an MA to identical servers at each stage of the computation and then comparing the results. In this scheme MAs do not communicate or cooperate. The assumptions that the identical servers would be available and that they would be under different administration domains, so that they would behave independently, are not realistic.

Yee (1999) proposed a fault-tolerant approach, where replicated agents are sent to the same set of hosts to figure out airfare prices for a flight ticket purchase. Agents traverse the hosts in the reverse order and at the end the minimum prices they come up with are checked. In the one malicious server case it is possible to find out the actual best price. This is one of the earliest proposals using multiple mobile agents which is application and case specific.

Ng (2000) used multiple agents for security purposes. In this scheme, again the agents do not cooperate; instead the task is split into multiple agents so that any agent alone would not reveal any useful information. In contrast to the multiple agent model we

use, agents in this scheme visits the same hosts, therefore they need to be completely anonymous to be able to defeat attacks.

Cooperating multiple agents have been first used by Roth (1998). It is shown that two cooperating agents, under certain assumptions, can verify the path each agent takes and whether the migration patterns adhere to the itinerary of the agents.

CHAPTER 4
REMOTE DIGITAL SIGNING WITH MOBILE AGENTS

4.1 Introduction

Although the MA paradigm opens many interesting applications, to validate it as an alternative to traditional client/server computing, one must address its security issues. In particular, it should demonstrate the ability to compute with secrets in remote public domains. A good example of the need for this is digitally signing a contract for m-commerce (and in general e-commerce) applications with MAs as shown by Sander and Tschudin (1998). We call this problem *remote digital signing*. In this chapter, a multi-agent architecture is used and a solution to this problem is presented. The techniques we explore and analyze are based on information dispersal in distributed system security terms as well as multisignatures and secret splitting in cryptographic terms. The idea is to devise a secure way of sharing secret keys among members of a multi-agent group and signing with shares.

4.2 Electronic Commerce and Mobile Agents

Among many application areas of MAs (such as information retrieval, e-commerce, network management, network/site security, distance education, and software distribution) e-commerce draws the most attention from both academic and industrial researchers, for example see Busch et al. (1998) and Klusch (1999). This is mostly due to the fact that, MAs and in general agent systems have the capability of representing users (i.e., customers) in the cyberspace. Agents can effectively profile user preferences, act on behalf their owners, participate in e-auctions, watch stock prices, search for commodities

and find the best offer from competing vendors, purchase goods by paying and committing to transactions, communicate and cooperate with other agents of relevant goals. Although it is now agreed that MAs are not a new enabling technology, they offer many technical capabilities together (i.e., all-in-one) over the traditional client/server computing (Chess et al. 1995a; Chess et al. 1995b; Lange & Oshima 1998). Mobile commerce (m-commerce), which is a rapidly growing field in e-commerce, is especially a suitable application area of MAs.

4.3 Mobile Commerce and Mobile Agents

Mobility of agents brings unique advantages to m-commerce. Mobile devices such as PDAs and laptop computers have limited battery life, intermittent and low-bandwidth connections to the fixed network. Traditional client/server computing which was originally designed for and very well fit into the fixed wireline networks is not suitable for m-commerce due to these limitations. MA paradigm enables disconnected operation (Chess et al., 1995a; Gray et al., 1996), where a brief connection to the fixed network from a mobile device through the wireless network is sufficient to launch an MA (or MAs) to engage in a mobile commerce activity. For example, a laptop owner, which has a wireless connection to the Internet, through a cell phone, may launch an agent to search for the best offer for an airline ticket and make a purchase. While the agent working towards the goal of purchase, the owner can (or may be forced to) disconnect from the network. When the MA accomplishes the goal, it takes another brief connection to receive the agent with the results.

4.4 Mobile Agent Security in Mobile Commerce

There are two aspects of the security issues in MA technology which are known as the malicious agents problem and the malicious hosts problem. In the former case, the

hosts that are to accept and execute the agents should be protected against any possible hostile behavior of agents. There are known mechanisms such as sandboxes proposed and implemented. The latter case is considered to be much more challenging due to the remote nature of the platforms where the MAs are to be run. Since these platforms are owned and operated by other parties, it is difficult to establish trust. Classical security mechanisms designed for distributed systems, including the cryptographic ones come short for threats against the MAs due to the assumptions, which do not hold for MAs (Chess, 1998). So, protection mechanisms are needed to make MAs safe in possibly hostile environments.

E-commerce is the most security demanding application of the MA paradigm. This is not different for m-commerce, which is a special case of the broader topic of e-commerce. In fact, it can be argued that if all the security requirements of e-commerce applications are met, then the general MA security problem is solved altogether. The shopping agent application where a MA is deployed to find the best possible price for some good such as an airline ticket, flowers or CDs, and make a purchase, has become the classical problem for discussing the requirements of MA security and proposing solutions to certain aspects of the requirements (see for example, Berkowitz et al. (1998), Hohl (1998), and (Yee 1999)). Hohl (1998) provides an extensive list of attacks using a shopping agent application example that could be launched against an agent.

The focus of this chapter is the remote digital signing problem for shopping agents. In any trade, principals engaged in the activity need to authenticate each other. A merchant would like to know whether the credit card presented by the buyer really does belong to the party or whether a check provided is legitimate and authentic. Customers

would like to make sure they present their confidential information such as a credit card to the merchant of their choice, but not anybody else. Similarly, merchants need to authenticate the MAs and their owners. This is necessary to prevent repudiation, which could be a very simple attack to devise using MAs. Even honest users may change their minds well after the transaction took place and deny that they didn't send any MAs to buy any such product. On the other hand, a hostile MA could masquerade a legitimate MA and hence its owner, to engage in fake trading to harm either or both of the principals of the transaction. Therefore a MA should be capable of digitally signing a contract agreed on by both parties to authenticate themselves and their owners, remotely and publicly, meaning that on the hosts they execute.

4.5 Objectives

Our objective in general, is to meet the requirements of solutions proposed for any aspect of MA security problem. These requirements were identified in Chapter 1.

Our specific goals in this chapter are to achieve a solution to the remote digital signing problem which should be as simple, realistic, flexible and ubiquitous as possible. With simplicity, we mean that the solution will be easy to understand and implement. By the use of already established and standardized digital signature schemes, such as RSA and El Gamal algorithms, if the original signing and verification functions can be used then specific implementation may not even be needed for our problem. To be realistic, it is meant that a proposed solution should fit into the real world environments, where they are to be used. For example, in theory, threshold signature schemes seem to fit very well in the MA paradigm when a multi-agent model is used. However, in practice it is necessary to identify the hosts where these MAs are going to be executed. The number and location of these hosts are to be restrictive as explained in detail later. Flexibility is

related to the autonomy of MAs from another perspective. Unlike some other solutions proposed, it is important to distinguish what can be done (i.e., signed) by a MA and what actually has been accomplished. Ubiquity is again related with the cryptographic functions used. Widely implemented cryptographic signature schemes improve the scalability in terms of number of hosts where MAs may need to find and use these schemes.

4.6 Background

Multiple agents have first been used by Minsky et al. (1996) for fault-tolerant distributed computing with MAs. The proposed scheme is deploying clones of an MA to identical servers at each stage of the computation and then comparing the results. In this scheme MAs do not communicate or cooperate. The assumptions that the identical servers would be available and that they would be under different administration domains, so that they would behave independently, are not realistic. Ng (2000) used multiple agents for security purposes. In this scheme, again the agents do not cooperate; instead the task is split into multiple agents so that any agent alone would not reveal any useful information. In contrast to the multiple agent model we use, agents in this scheme visit the same hosts, therefore they need to be completely anonymous to be able to defeat attacks. Cooperating multiple agents have been first used by Roth (1998). It is shown that two cooperating agents, under certain assumptions, can verify the path each agent takes and whether the migration patterns adhere to the itinerary of the agents.

Sander and Tschudin (1998) introduced the concept of Mobile Cryptography. The idea is to encrypt agents as a whole and apply computing with encrypted functions and data. Although it is limited to polynomial and rational functions, this is a good example of a software solution to the MA security problem that is based solely on cryptography.

In the same paper they also introduced the concept of “undetachable digital signatures,” which is based on the concept of computing with encrypted functions. They point out that this is a possible realization of “… an agent would like to use the secret in public e.g., to compute the digital signature of an order form but without disclosing the secret needed to do so.” In this approach, user constraints are “glued” together with the general purpose signature function to enforce them to be a part of the signed contract; hence the term “undetachable signatures.” Nevertheless, they also point out that the scheme on which their proposal is based has been successfully attacked.

Kotzanikolaou et al. (2000) proposed a solution to the problem introduced by Sander and Tschudin (1998). They use RSA (Rivest et al., 1978), which is based on exponential functions rather than rational functions. However, as the term “undetachable digital signatures” implies, the solution given by Kotzanikolaou et al. (2000) requires that the signature be generated by the user (i.e., owner of the agent) and given to the agent *before the mission takes place*. This contradicts MA autonomy. This is due to the fact that the purchase decision has to be made strictly before negotiation with the sellers. User constraints, which have to be signed before these negotiations, therefore need to be pure data. However, it is desirable that a decision function be executed after or during the negotiation or bargaining process. This means that agents should be capable of deciding what to buy, where to buy, under what conditions, price, type of payment, delivery options, etc. For example, the user demand should be able to be stated as flexibly as possible with “I would like to purchase as many blank rewritable CDs as possible and I’ve got $100.” The result of the decision function will have a direct effect of the contract to be signed. Therefore what we need is to make the agents capable of computing with

secrets in public as the quoted sentence in the previous paragraph implies. Without this capability, either the user must have a perfect knowledge of market conditions that might change rapidly, or user interaction during the mission is necessary. In the former case it is highly possible that the mission may fail, in the latter, autonomy is sacrificed. The problem arises from the fact that a malicious host should not be able to manipulate or directly use the agent in order to sign “arbitrary” documents. On the other hand, agents should also be capable of preparing the documents to be signed. The challenge is to resolve this contradiction.

A threshold signature scheme in conjunction with the use of multiple agents and an undetachable threshold signature scheme, which combines undetachable signatures with threshold signature schemes, have been proposed (see Borselius et al. (2001) and further references). While the former is vulnerable to attacks when used alone the latter still carries the concerns with threshold signatures.

First concern is that threshold signature schemes come in great variety. They are neither standardized nor widely accepted. This means that MAs may face problems in finding the hosts to execute, which would have standardized implementations of these schemes. The second concern is threshold signature schemes tacitly assume that there would be sufficient number of shareholders to sign a document. Even small values of this “sufficient number” may not be feasible for MAs since in practice, existence of hosts for MAs to execute on, finding those hosts and location of them in the underlying network are problems as will be explained later. So, threshold schemes are reduced to multisignature schemes by these restrictions. Nevertheless, their complexity remains.

4.7 Multiple Cryptography

Multiple cryptography, as the name implies, covers the cryptosystems that deal with more than two parties as opposed to the classical cryptography where there are only two parties: the one who encrypts or signs and the other who decrypts or verifies. In fact, there are many real-world applications that have multiple parties involved. For example, in a banking application, electronic fund transfers require approvals of bank officials of different ranks. Usually, at least two people are involved for a single transaction. Ironically, this is such an application that, in the digital world, a more realistic abstraction of the real world than the real world itself may be possible. Using the same example, a bank cannot have a signature. Only people who work for the bank have signatures, and they sign on behalf of the bank. But with multiple cryptography, it is possible to assign a key to the bank, and shares of this key with the individuals who work for the bank. When these officials sign a document like a check, the signature they generate perfectly represents the bank itself, but not the individuals who really signed the document. So as the example implies, it is possible, with multiple cryptography not only to share the secrets but also compute with them without regenerating the secrets.

Our focus in this chapter is on multiple digital signatures or *multisignatures*, which are a special case of multiple cryptography. The term refers to digital signature schemes, which enable multiple parties to sign documents or messages cooperatively but independently of each other using keys or shares of a key generated for this purpose. Boyd (1988) shows the generalization of RSA and use it as a multisignature scheme. A brief explanation of Boyd's work, which is related to our application, will be given later.

Multiple cryptography, in general, is intended for classical applications (e.g., a banking application). In these applications, shareholders are usually individuals who

represent an organization or a company. There is an important distinction between these applications and the MA applications. MAs are nothing but software entities. They are neither organizations nor individuals. Moreover, they are owned by individuals or organizations. In fact, it can be argued that, the MA owner and a bank have some kind of resemblance. So, the officials of the bank and the MA perform similar operations when signing a document. While this is not wrong, the actual difference comes from the fact that, MA owners are active players, while organizations or companies like a bank are not. A bank is actually an abstraction and cannot for example sign a document. But in the case of a human MA owner, this individual can equally sign documents him/herself. Also, the shareholders do not always exist. They are created when necessary and after they complete their work they cease to exist.

In addition to the threshold schemes like Shamir's (1979), techniques which are known as *threshold cryptography* for the purpose of not only sharing keys but also being able to compute with them without a central authority, have been proposed. A survey of research in this area was provided by Desmedt (1997). A threshold multisignature scheme has been given by Frankel and Desmedt (1992). In this scheme, authors combine RSA signature scheme by Rivest et al. (1978) together with Shamir's (1979) threshold scheme to distribute and to sign documents with shares. It is also possible to generate the shares of a secret in a distributed fashion, which enables the shareholders to compute their own shares without the necessity of a central authority. An example using RSA is given by Boneh and Franklin (1997). The threshold multisignature schemes are in fact the generalization of the multisignature schemes. While key sharing is *k-out-of-k* in the latter,

it is *t-out-of-k* in the former, which means that *t* of the total of *k* shares are enough to generate signatures.

Nevertheless, threshold schemes do not have specific advantages over simple secret splitting techniques we are using, with MAs. While it is feasible to use these secret splitting schemes to both share the keys and compute with them, it is also feasible to come up with very simple techniques to create multiple combinations of keys for providing fault-tolerance, as demonstrated by Wu et al. (1999). On the other hand, in our application, secret splitting has two important advantages: simplicity and ubiquity. We use very simple secret splitting schemes, which use only addition and multiplication. The secret splitting schemes we use require nothing but the implementations of the standard public key cryptosystems, namely, de facto industry standard RSA and the official Digital Signature Standard (DSS), which is based on El Gamal public key scheme. So, these schemes do not need any new algorithms or an implementation of those algorithms. It should also be noted that what makes it possible to use these simple secret splitting schemes with the El Gamal cryptosystem and DSS is the unique property of the application that the whole secrets and all of the shares are to be computed and known by the MA owner; therefore it is possible to perform computations in advance to be used later when the complete signature is computed. Details of these computations are given in the next section.

4.8 Key Splitting and Signature Generation

Boyd (1988), by using the multiplicative property of RSA, showed that the classical RSA is actually a specialization of a general multisignature scheme. For an RSA implementation for sequential signing, we will use this property. Boyd (1989) also mentions about a similar technique, which enables to perform signature generation in

parallel. This is the RSA part of the techniques we will use in parallel signing. Note that, both techniques use nothing but original RSA signing and verification algorithms. The RSA implementations, which are standardized and used in practice, and their implications on remote digital signing will be discussed later in the chapter. In the following, we present techniques to do the same with El Gamal Cryptosystem, which again use original signing and verification procedures, by using a property that is unique to MAs, as explained.

4.8.1 Using the Multiplicative Property of RSA

Classical public key cyptosystems use two keys, one is the public key and the other is the private key. It was shown by Boyd (1988) that this is only a specialization of a general multisignature scheme in the case of RSA. This is possible since RSA has the multiplicative property, namely, with the same modulus and two decryption keys d_1 and d_2,

$$S\left(S\left(m_h, d_1\right), d_2\right) = S\left(m_h, d_1 d_2\right)$$

where S is the signing function and $m_h = h(m)$. Here, m is the document to be signed, h is an appropriate cryptographic hash function (i.e., SHA-1) and m_h is the message digest. Note that, for clarity we defer the discussion of recent advances in theory and practice of computing the message digest m_h for better security of RSA signature scheme until Section 8.

The modulus n is still the product of two large primes p and q. However, unlike classical RSA, which has two keys e and d that are encryption and decryption keys respectively, in this scheme $k-1$ keys are chosen randomly and then the k^{th} key is chosen to satisfy the property

$$d_1 \cdot d_2 \cdot \cdots \cdot d_{k-1} \cdot e = 1 \bmod \Phi .$$

The encryption key is *e* and when the d_i's multiplied together the result is the decryption key *d*.

So, with three partial keys d_1, d_2, and d_3 three agents can sign a document with

$$X = \left(\left(m_h^{\,d_1}\right)^{d_2}\right)^{d_3} = m_h^{\,d} \mod n ,$$

which can be verified by

$$X^e = m_h^{\,d \cdot e} \bmod n .$$

4.8.2 Using the Additive Property of RSA

Boyd (1989) mentions an alternative scheme to overcome the difficulties of multisignatures with more than two signatories (e.g., different users). In the context of multisignatures it is possible to assign different keys to different users, and signing process can be done in parallel. Resulting signatures are then combined. Using the same scheme, it is also possible for our purposes to split up a key by addition instead of multiplication assuming three signatories as follows

$$d = d_1 + d_2 + d_3 .$$

Each agent is provided with one of the partial keys d_1, d_2 and d_3. When it is time to sign a contract, they are either provided a copy of the original contract or they prepare the same contract depending on the application (see Section 4.9).

Then they sign by

$$S_1 = m_h^{\,d_1} \bmod n \qquad S_2 = m_h^{\,d_2} \bmod n \qquad S_3 = m_h^{\,d_3} \bmod n .$$

Note that, the message digest m_h above is computed as mentioned in the previous section.

Signatures S_1, S_2, and S_3 are sent back to the server where the signatures are required. The server computes

$$S_1 \cdot S_2 \cdot S_3 = m_h^{d_1} \cdot m_h^{d_2} \cdot m_h^{d_3} \bmod n = m_h^{(d_1+d_2+d_3)} \bmod n = m_h^{d} \bmod n .$$

Therefore, the server can also verify the signature by

$$(S_1 \cdot S_2 \cdot S_3)^e = m_h^{d \cdot e} \bmod n .$$

The scheme can be generalized for *n* signatories in the obvious way.

4.8.3 Using El Gamal Public Key Cryptosystem

Here we use a variant of the original El Gamal signature scheme as given by Kaufman et al. (1995). There are two reasons to do this. First is, this scheme is simpler and it is easy to compute with partial keys. The other reason is that, the scheme is actually the El Gamal version used in DSS, which in turn is based on the original idea introduced by El Gamal (1985). So this scheme will enable us an easy transition from El Gamal to DSS. The El Gamal variant that we use is summarized below (Kaufman et al., 1995):

- Long term public key: $<g, p, T>$, secret key: S, where $g^S \bmod p = T$
- For a message m choose random number r, compute $g^r \bmod p = T_m$, and message digest d_m (digest of $m \mid T_m$)
- Sign with $X = r + d_m \ S \bmod (p - 1)$
- Verify by $g^X = T_m \ T^{d_m} \bmod p$

In the following sections, we use Figure 4-1 to illustrate the signature generation scheme. There are three mobile agents, Alice, Bob and Carol. Alice is the master agent and the others are support agents. Figure uses the same mission model developed in Chapter 3, however, for clarity only a snapshot of the mission is shown.

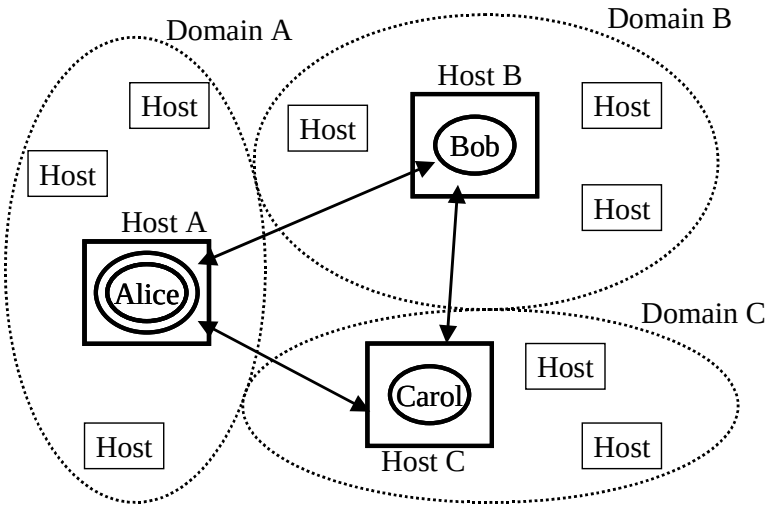


Figure 4-1. A snapshot of a mission using multiple mobile agents

4.8.3.1 Signing in sequence with El Gamal signature scheme

The El Gamal cryptosystem uses a pair of private keys as opposed to RSA's single key. The first one is the long term key as in the RSA. The second is a short-term, per session, private key for each message to be signed with the long-term key. We split up both of these keys as follows:

Long-term key: $S = S_a \cdot S_b \cdot S_c$ (1)

Short-term key: $r = r_a + r_b + r_c$. (2)

Here we assume again that our MA group consists of three agents, namely, Alice, Bob and Carol. Alice is the master and the others are support agents.

To sign a message, Alice computes the message digest d_m and signs with

$$X_a = r_a + d_m \, S_a \bmod (p-1) \qquad (3)$$

Note that, the message digest d_m here is the result of an appropriate cryptographic hash function H applied to the contract m concatenated with T_m:

$$d_m = H(m \mid T_m) \qquad (4)$$

where $T_m = g^r \bmod p$ as given above. In sequential signing, Alice is the only agent, who needs T_m and it is assumed here that she is provided by this value before the mission takes place.

Then, she sends her partial signature to Bob. Bob, upon receiving Alice's signature X_a, further signs it with his portions of partial keys as

$$\begin{aligned} X_b &= r_b + X_a \; S_b \bmod (p-1) \\ &= r_b + S_b \; r_a + d_m \; S_a \; S_b \bmod (p-1) \end{aligned} \qquad (5)$$

Carol does the same on X_b, which represent the partial signature generated by Alice and Bob,

$$\begin{aligned} X_c &= r_c + X_b \; S_c \bmod (p-1) \\ &= r_c + r_b \; S_c + r_a \; S_b \; S_c + d_m \; S_a \; S_b \; S_c \bmod (p-1) \end{aligned} \qquad (6)$$

Unfortunately, this last equation above, unlike the RSA counterpart, is not equal to the signature *X*, although the last term of the equation is nothing but the last term of the original signing equation:

$$d_m \; S_a \; S_b \; S_c = d_m \; S \, . \qquad (7)$$

This leads us to the observation that the difference between the target signature X and the signature generated by the three agents X_c is

$$X_c - X = (S_c - 1) \, r_b + (S_b \; S_c - 1) \, r_a \bmod (p-1) \qquad (8)$$

Since the right hand side of the equation consists only of constants and partial private keys, it can easily be computed and given to agents before they are sent out to the network. This ability is *unique* to the application that we consider in this chapter. In the classical applications of digital multisignatures and threshold signatures, it is not possible to perform the same computation since the signatories are distinct parties and the secrets

they share cannot exist in a single site as a whole. So the last agent in the row, Carol, sends the partial signature X_c to Alice. Alice computes X, the target complete signature by using the equation above. The general difference equation for n signatories is

$$X_n - X = \sum_{t=1}^{n-1} \left(r_t \left(\prod_{k=t+1}^{n} S_k - 1 \right) \right) \bmod (p-1) \qquad (9)$$

4.8.3.2 Signing in parallel with El Gamal signature scheme

Signing in parallel with the same variant of El Gamal cryptosystem is also possible and even easier. For this purpose we split up the keys as follows:

$$\text{Long-term key: } S = S_a + S_b + S_c \qquad (10)$$

$$\text{Short-term key: } r = r_a + r_b + r_c \qquad (11)$$

Then, each agent is given the partial keys as well as $T_m = g^r \bmod p$ since all of the agents will need it to compute the message digest $d_m = H(m \mid T_m)$ where m is the contract to be signed and H is an appropriate cryptographic hash function. They sign independently of each other as

$$\begin{aligned} X_a &= r_a + d_m\ S_a \bmod (p-1), \\ X_b &= r_b + d_m\ S_b \bmod (p-1), \\ X_c &= r_c + d_m\ S_c \bmod (p-1) \end{aligned} \qquad (12)$$

and the support agents send their partial signatures to the master agent. Together with the master agent's signature, the server combines the partial signatures and obtains the complete signature as follows

$$\begin{aligned} X &= X_a + X_b + X_c \\ &= r_a + r_b + r_c + d_m\ S_a + d_m\ S_b + d_m\ S_c \\ &= r + d_m\ (S_a + S_b + S_c) = r + d_m\ S \end{aligned} \qquad (13)$$

The scheme can be generalized to n signatories in the obvious way.

4.8.3.3 Transition from El Gamal cryptosystem to digital signature algorithm

While RSA is the *de facto* industry standard of public key cryptography, the Digital Signature Algorithm has been proposed as the official standard as Digital Signature Standard (DSS) by US National Institute of Standards and Technology. It is based on the original idea of the El Gamal public key scheme and is very similar to the variant of the El Gamal cryptosystem.

We will neither provide the details of DSS nor the details of the signature generation by partial keys. However, we will give the differences between the El Gamal scheme presented in previous sections and DSS.

The signing equation in DSS is given by

$$X = r^{-1}\left(d_m + S\,T_m\right) \bmod q \qquad (14)$$

where r^{-1} is the multiplicative inverse of $r \bmod q$. Therefore it can be calculated in advance and instead of splitting up *r* we can just as easily split up r^{-1}.

Signing in sequence with DSS. The key splitting is performed as follows:

$$\text{Long-term key: } S = S_a + S_b + S_c \qquad (15)$$

$$\text{Short-term key (inverse): } r^{-1} = r_a \cdot r_b \cdot r_c . \qquad (16)$$

The signing process is very similar to El Gamal scheme. However the difference in this case is given by

$$X - X_c = \left(\left(r_a - 1\right) r_c\; r_b\; S_b + \left(r_b\; r_a - 1\right) r_c\; S_c\right) T_m \qquad (17)$$

For *n* signatories (i.e., agents), the general difference equation is

$$X - X_n = T_m \sum_{t=1}^{n-1} \left(\prod_{k=1}^{t} r_k - 1 \right) \left(\prod_{w=t+1}^{n} r_w \right) S_{t+1} \qquad (18)$$

Signing in parallel with DSS. The key splitting, signing and signature combination calculations here are very similar to the El Gamal scheme. However, the combined value is not equal to the target complete signature *X*. Therefore we call this value *X'* and the difference is given by

$$X - X' = T_m \left(r_a \left(S_b + S_c \right) + r_b \left(S_a + S_c \right) + r_c \left(S_a + S_b \right) \right) \quad (19)$$

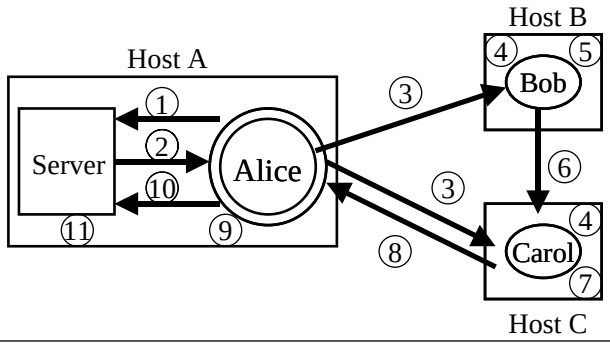
For *n* signatories (i.e., agents), the general difference equation is

$$X - X_n = T_m \sum_{t=1}^{n} \sum_{\substack{k=1 \\ k \neq t}}^{n} r_t S_k . \quad (20)$$

4.9 The Overall System for Remote Digital Signing

As presented in the previous section, there are two major multi-signature schemes: sequential and parallel. Figures 4-2 and 4-3 provide the overall system of signing and verification processes as part of an MA mission, for sequential and parallel signature generation schemes, respectively. In this section, we discuss these protocols, compare them with respect to assumptions made and the analysis of attacks possible against each scheme.

Please note that, in this chapter we do not address fully the overall security issues necessary for a whole MA mission. Signature generation is actually an integral part of the whole mission. However we provide the protection mechanisms when necessary in addition to the signature generation process to make this process more meaningful.

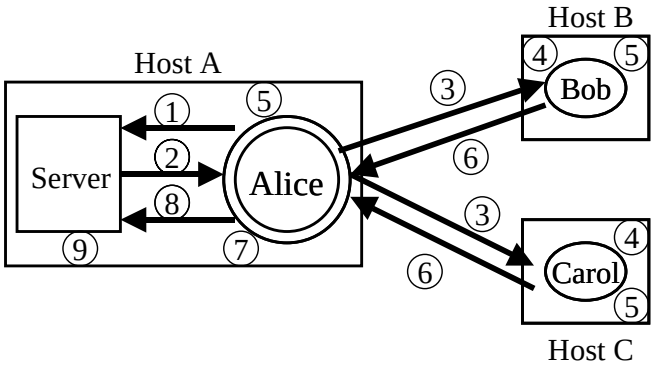


1. Alice asks the Server for a bid for her demand
2. Server provides Alice with its offer along with its signature for the contract
3. Alice sends the server's bid along with its signature to Bob and Carol
4. Bob and Carol verify the server's signature
5. Bob prepares the contract and signs it with his part of private key
6. Bob sends the partially signed contract C_B to Carol
7. Carol signs the contract, partially signed by Bob, further with her partial key
8. Carol sends the partially signed contract C_{BC} to Alice
9. Alice, with her partial key, further signs the contract and obtains the fully signed contract C_{ABC}
10. Alice delivers the signed contract C_{ABC} to the server
11. Server verifies the contract using the public key of the user

Figure 4-2. Protocol for sequential signing with multi-agent model

In both of the protocols, the first part is to obtain the bid of the server along with the signature for this offer and then verify this signature in steps 1 through 4. After Alice obtains server's signature, she sends it to both Bob and Carol. Signature verification is carried out by both Bob and Carol independently on different hosts in different domains. It would not make sense to let Alice verify the signature since Alice is under complete control of the very same host that made the offer. Step 5 in Figure 4-2 contains the contract preparation process performed by Bob. The protocol assumes that Host B does not have any interest in forging the computation as does Host A. Although the chances are low, this assumption is not enough for a convincing level of security, since a denial-of-service attack by Host B is possible. This is due to the drawback of this protocol that there is no way to check whether the contract signed by the first agent in the row is correct. This is true regardless of the fact that the decision function is executed in cooperation with all the members of the group. However, this drawback does not make

the protocol totally inappropriate because even an attack cannot be prevented, it would be detected in Step 11 when the host attempts to verify the signature. Nevertheless, it is not possible to tell which host is cheating. The parallel signature generation scheme does not have this drawback as we see next.



1-4. Same as Figure 4-2
5. Alice, Bob, and Carol prepare the contract and sign it with their part of private key and each obtain C_A, C_B, and C_C respectively
6. Bob sends the partially signed contract C_B to Alice, Carol does the same with C_C
7. Alice, combines C_A, C_B, and C_C, and obtains the fully signed contract C_{ABC}
8. Alice delivers the signed contract C_{ABC} to the server
9. Server verifies the contract using the public key of the user

Figure 4-3. Protocol for parallel signing with multi-agent model

In Figure 4-3, Step 5 states that all agents prepare the contract to be signed. This is possible since Bob and Carol receive the server's offer from Alice. Furthermore, any decision function that needs to be executed to reach the decision for actual purchase is shared by three agents and executed in cooperation. However, it is also possible for any one agent to prepare the contract and communicate it to the others. Once the contract is obtained either way, agents sign it with their shares of the private key for the chosen algorithm as described previously. Step 6 says that Bob and Carol send their signatures back to Alice, and in Step 7, Alice combines them to obtain the fully signed contract. This means that combining the partial signatures takes place in the host where Alice resides. This is because there is no trusted authority to ask for this computation. But for

the application under consideration, there is no need for it either, since there is no known attack possible.

4.10 Using Limited-liability Keys and Public Key Certificates

Our multi-agent model provides a high level of security by making it difficult to compromise all the agents, which together form an autonomous group. However, the secret, which is a private key of the agent owner, is an extremely sensitive piece of information. Revealing user's shopping preferences or losing electronic cash is certainly undesirable, but in essence a part of our daily lives since we have just enough security to protect these valuables. But a forged signature under a document may mean "anything" and may not be tolerable or acceptable. So even a very small chance of the whole group of agents being compromised may not be acceptable since the consequence of this malicious action is revealing the private key of the user. Therefore we define two types of public/private key pairs as follows.

Long-term public/private key pairs: These are the long-term keys, which are created, registered and assigned to the user (who becomes the owner of the keys) by a Certificate Authority (CA). The lifetime of these keys are again decided by the same authority.

Limited-liability public/private key pairs: These are the short-term keys, which are created by the very same user (owner of the keys). Since the creators of the keys are the users themselves they also are the authority to decide the lifetime of the keys.

There are two limitations that can be imposed on the limited-liability keys. The first one is the lifetime of the key and the second one is what can be done (i.e., signed) with these keys. Both of these limitations can be flexible or restrictive and is up to the owner of the keys. For example, the lifetime of a key may vary from a single MA mission,

which may be limited to a couple of minutes, to a total of a couple of days that spans several different missions. Liability definition says what exactly can be signed with these keys. For example, a typical definition may say that these keys can only be used in an m-commerce transaction and the amount that could be involved in such a transaction cannot exceed $500. The other type of limitation that can be imposed is the type of the transaction. For example, it can be stated that, with these keys only one mobile phone, one color printer or one DVD player purchase contract can be signed.

The problem with this scheme is the authenticity of this new pair of keys. One possible solution would be to register this pair of new keys with a CA and obtain a certificate, or ask the CA directly for a pair of new keys. Nevertheless, this solution does not make much sense, since this introduces an overhead of obtaining a new key for every single transaction from a CA, which could easily become unmanageable. Instead, the user creates this key and a certificate for this key by using his/her long-term key. The idea is that the user obtains only a single public key and a certificate for it from a CA. Using this certificate and public key, it is possible for the very same user to create and use unlimited number of other public keys. The key point here is that these new keys are certified by their owners. That is, the user in essence becomes a CA for the agents he/she sends on missions.

It may seem at first that the scheme explained in this section provides enough security for the limited-liability private key. This is due to the fact that what can be signed by this key is restricted by the certificate provided for it. However, it is not so for two reasons. First problem is the same problem that the undetachable signature schemes have as explained previously, which is the difference between the limitation of what can

be signed and what actually is signed. A certificate can only enforce what can be signed with the key, for which it is issued. But in fact, it is necessary for agents to be able to engage in transactions with the most favorable choices. For example, a certificate may allow for a purchase of up to $500. However, this should not mean that the agent would accept an offer of this amount. If the server's original bid is $300, then the server should not be able to enforce the agent to involve in an agreement for the allowed full amount of $500. The second problem is that an attack is possible by any host visited. Any such host for example might learn the complete limited-liability private key from agent Alice, and then can sign a contract to sell something. Also, even if the support agents are involved in the decision function to be executed, this is not enough since a single agent who possesses the complete key could be manipulated. In short, this scheme when used alone could open a can of worms. Therefore, the limited-liability keys are protected by splitting them up and giving them to members of a group of MAs. Their usage, on the other hand, is protected by the certificate created by the user using the long-term key.

The complete protocol for using the certificates and limited-liability keys for agents is given in Figure 4-4. In the protocol, *P* and *S* represent the long-term public and private keys respectively and *p* and *s* represent the limited-liability keys. The protocol assumes that user already has *P* and *S*. Note that *P*, *S*, *p* and *s*, are symbols for the private and public keys, for example in the case of RSA, *P* and *p* indicate (*E*, *N*) and (*e*, *n*) respectively. The certificate contains *p* and the description of the liability. Any document, which is signed with *s* and therefore can be verified by *p*, should conform to this description to be valid.

Note that a certificate chain that would also include CA's certificate for (*P*, *S*) rules out the necessity for the server to connect the CA and ask for verification.

1. User creates (*p*, *s*) or uses a pre-computed pair of keys
2. User splits up *s*, using one of the techniques mentioned
3. User prepares a certificate and signs with *S*
4. Server uses *P* to verify the certificate and obtains the identity of the user and *p*
5. Agents sign the contract with their shares of *s*
6. Server verifies this signature using *p*

Figure 4-4. Limited-liability key protocol

Now let's look at what can and cannot be accomplished with these limited time and liability keys and what can go wrong. The analysis involves three parties that can act maliciously; agent owner, the host to which a signature is provided, and third parties, which could presumably acquire the limited-liability private key. Since the private/public key pair is arbitrary, meaning that they are neither created nor registered by a CA, the host on which the transaction took place may sign a contract that states that the agent purchased 500 mobile phones for the price of $100 each. Since the host also knows the credit card information of the user, it can perform a transaction and bill the user's credit card for this transaction. The solution to the dispute requires the proof of the user's demand for this kind of purchase. But the certificate which is signed by user's long-term private key pair states that the transaction amount may not exceed $500 and it is only valid for a purchase of a single mobile phone. Therefore there is no way for the host to prove such claim.

Now, suppose that a third party was able to compromise all the agents in a group and therefore acquire the limited-liability key. Then this party could sign an arbitrary contract on behalf of the user by masquerading as the user. In this case, however, it is not possible to bind the user to this contract since there is no certificate signed by the user

using the long-term private key. So, such signed document is invalid because the user can deny it rightfully.

Third attack involves a malicious user, and is known as non-repudiation problem. The user cannot deny a contract that states a purchase demand that conforms what is stated in the certificate signed by the user's long-term private key. This is what we expect since this is the aim of the certificate and is completely legitimate. Also, the user cannot deny the legitimacy of a contract by claiming that the contract was signed after the limited-liability key had been expired. That is because agents are supposed to check whether the expiration date and time has passed. On the other hand, as explained in the previous paragraph, the user can deny any illegitimate contract signed by the limited-liability key that does not conform to the certificate. Another similar attack might be possible, if an attacker creates and uses a limited-liability key by also creating a fake long-term key without registering this long-term key by any CA. Then the attacker prepares a certificate for the limited-liability key using this fake long-term key. The last step for the attacker is to let the agents engage in transactions and then deny their existence. These transactions may or may not conform to the certificate prepared. So, the merchants/sellers should always check the legitimacy of the short-term keys by checking the certificates prepared for them, as well as the legitimacy of the certificates by checking the long-term keys used to prepare them possibly consulting the CA, if not provided.

4.11 Practical Issues of Remote Digital Signing

In this section we examine the issues related with using the Remote Digital Signing in practice. First we look at the issues related with the security level and robustness of the multisignature schemes by discussing the recent theoretical advances in digital signatures, their impact on implementations in practice and particularly on remote digital

signing. Then, we look at the broader picture of multiple MAs paradigm and discuss the performance issues that might arise in practice and how to address them.

4.11.1 Probabilistic Signature Scheme and its Impact on Practice

Probabilistic Signature Scheme (PSS) has been proposed by Bellare and Rogaway (1996). It is applied to the hashing but the signature generation is the same. The idea is to mix the document to be signed with a random salt in a specific way and the result is a better security proof. This is due to the fact that security of PSS can be tightly related to the difficulty of inverting RSA. We refer the interested reader to Bellare and Rogaway (1996) for details. As the name implies, PSS is probabilistic rather than deterministic, which is the case for classical RSA commonly in use today. The scheme has practical importance and has been adopted by the RSA Corporation as a standard (PKCS#1, 2002).

Unfortunately, the scheme is not applicable in environments where deterministic signature generation is necessary. In our case, it is applicable to sequential signature generation using RSA since signature generation is initiated by a single agent and appropriate hashing is only performed by the same agent. The other agents (i.e., support agents) therefore need not know the random salt value used and can apply their keys to the signature function. However, parallel signature generation with RSA requires a deterministic scheme since all the agents need to know or be able to generate the same random salt. The issue is important since our main concern is to use standardized and widely implemented and used schemes like RSA. The random salt value in PSS enhances the security by providing a tighter security proof. However, in practice, randomness is not critical to security (PKCS#1, 2002). In our case, parallel RSA signature generation is still feasible by providing a fixed value to the agents or have them compute the same value and use it when it is time to generate the underlying message digest to be signed.

Furthermore in our scheme, the lifetime of the keys and their applicability can be extremely restricted due to the use of certificates issued for them. So, the probability of forging the signatures is extremely low and it is difficult to justify the efforts to do so.

Some theoretical research results however lead us to question the probabilistic schemes like PSS. Recently, Katz and Wang (2003) showed that an equally tight security proof of PSS could be constructed in a deterministic way. The idea is to use a single bit (0 or 1) instead of a random salt value when generating the hash value. This variant of PSS may render the current proposed standardization of the original PSS obsolete, which is not convenient in environments where random generation is not possible or is not preferred as in the case with parallel signature generation presented.

4.11.2 Performance Considerations and the Big Picture

Remote digital signing is in fact a part a new computation paradigm of multiple MA systems. Among many issues of this big picture, performance considerations deserve specific attention since multiple agents introduces additional communication to the classical single agent model. The immediate result follows as more communication overhead to the underlying network and degraded performance observed by the user. Network-awareness in general and specifically network distance estimation are hot research topics and aims at assisting applications, which would benefit from being aware of the underlying network characteristics and conditions. Multiple MAs are such an application area where network-awareness as well as context-awareness, which also addresses security considerations, is important. For example, choosing hosts to send the agents randomly increases the security risk of conspiracies against them. So, hosts to be chosen need to be in different administrative domains and should not have common interests to alleviate this risk. The goal, therefore, is to find the best possible combination

of available hosts under the restrictions mentioned above, which are (in terms of network distances) close to each other. This not only addresses the performance but also the security since vulnerability of the agent communications over the network reduces. These issues are the subject of following chapters.

One last remark on performance issue is the subtle difference between the client/server paradigm and the MA paradigm in general, regarding user expectations. The applications of the MA paradigm, which is in fact a special case of the multiple MAs paradigm are not user-interactive as is the case for client/server systems. Users do not expect immediate results from their MAs and it is in that aspect that constitutes the foundation for enabling disconnected operation. Therefore, performance penalties should be observed in a more relaxed fashion than it should be in the traditional client/server computing.

4.12 Conclusion

Mobile commerce is a rapidly developing field of electronic commerce. Mobile devices and wireless networks, which make the mobile commerce a reality, however are not developing at the same pace. It can be expected that the limitations of these technologies will be with us several decades from now. MAs are distinguished candidates to tackle this problem. Although benefits offered by this technology are well understood by now, especially in the mobile commerce field, security and interoperability are two main concerns of this technology. It is also well understood that without proper security mechanisms in place, MAs will not be accepted. So, advances in MA security area will directly affect the future of mobile commerce applications.

This chapter carefully examines the implementation of remote digital signing, which is a special case of computing with secrets in the public domain, within a larger picture of supporting multi-agent computing. The solution approach is based on secret sharing and the concept of protection as a whole. In addition to well-known multiplicative and additive properties of the RSA, similar techniques with El Gamal public-key cryptosystem are demonstrated to show their applicability in the MA systems. Although threshold multisignature schemes fit well into the application, it may not be feasible and even reasonable to provide the implementations of these schemes to the MAs. Moreover, the advantage of the techniques we presented in this chapter is that they use nothing but original signing and verification algorithms of RSA and DSS, which are well known and widely used. In this sense, these simple schemes address the ubiquity in a large scale MA execution environment like the Internet.

The MA security problem is not limited to signature generation, however. In fact, signing a document is supposed to be a part of an MA mission. A complete solution to the problem of protection against malicious hosts is still to be addressed in our future work. However, as we have presented in this chapter, even the most challenging part of the problem, computing with secrets by MAs, is feasible using the multi-agent model.

It is shown that both parallel and sequential signature generation schemes are possible. There are important properties of each scheme. The former one provides data integrity since each agent in the group signs the original document, therefore can check its integrity. However, in this scheme there is no confidentiality for the document to be signed. The latter scheme provides data confidentiality only if signing process begins at the server, where the master agent is being executed. The other hosts, where the support

agents are running, can not see the document in clear, assuming the partially signed documents have enough strength against cryptanalysis. The schemes presented in this chapter therefore address authenticity rather than data confidentiality.

In this chapter, while applying information dispersal to mobile agents to accomplish digitally signing contracts in the public, our tacit assumption was that we could position these agents in the hosts over the network; this positioning as well as the mission as a result of it would be feasible. In the next chapter we examine this positioning problem to show that it is feasible to identify the best host locations to realize the information dispersal with multiple mobile agents.

CHAPTER 5
A NETWORK POSITIONING ARCHITECTURE FOR MOBILE AGENTS

5.1 Introduction

Applications that can benefit from being aware of the underlying topology and the network distance information are numerous. For example, in a web caching system, clients of the system may choose the best location among the alternatives to obtain the cache copies. Similarly, the location information is obviously useful in placing WWW/FTP mirror sites in different areas on the network. In such a replica system, clients may need an automated mechanism to choose the best mirror site based on topology information. The problem of addressing this requirement for such applications is known as the *server selection* problem. Another important application area is in the design of *content addressable networks* (Ratnasamy et al., 2001) and *overlay networks* (Stoica et al., 2001). These systems require transformation of logical network topologies from an underlying physical network. In general, any *content provider* or *peer-to-peer* (p2p) file sharing facility (i.e., Napster and Gnutella) requires topology information for the best performance, which otherwise may not be observed. Many other topology-aware applications have been explored in literature (Ratnasamy et al., 2002).

Different from the applications described above, our interest in developing a network positioning model came from the need for supporting a *multiple mobile agent system*.

The proposed positioning model is a p2p coordinate-based system, which maps a physical host location to logical coordinates for efficient on-the-fly logical distance (e.g., communication delay) estimation between any two hosts in the system.

5.2 Related Work

Much like time services in distributed systems, distance services can be provided by some servers or obtained collaboratively by some participating hosts. The IDMaps (Francis et al., 2002) is an example of the former case. This approach uses triangulation heuristics combined with a centralized, client/server architecture.

Examples of the latter case are the coordinate-based Global Network Positioning (GNP) (Ng & Zhang, 2002), the Lighthouse (Pias et al., 2003) and the binning (Ratnasamy et al., 2002) approaches. The GNP and binning strategies use the concept of *landmarks*, which are introduced by Hotz (1994). In GNP, coordinates are computed by modeling the Internet as a geometric space. Some hosts in the system are identified as landmarks. These landmarks first measure their distances to each other and then compute their coordinates by minimizing the error between the measured and the computed distances (see Step 4 in Section 5.3.1 for details).

This results in approximate distances as represented by the coordinates among landmarks. An ordinary host can measure its distance to these landmarks and use the landmarks' coordinates to compute its own with the same minimization technique. Figure 5-1 illustrates two hosts *A* and *B*, which compute their coordinates using the global landmarks *L1*, *L2* and *L3*. From there on, *A* only needs to ask *B* for *B*'s coordinates to obtain the distance to *B*. The GNP is a p2p architecture as oppose to the IDMaps' central servers approach.

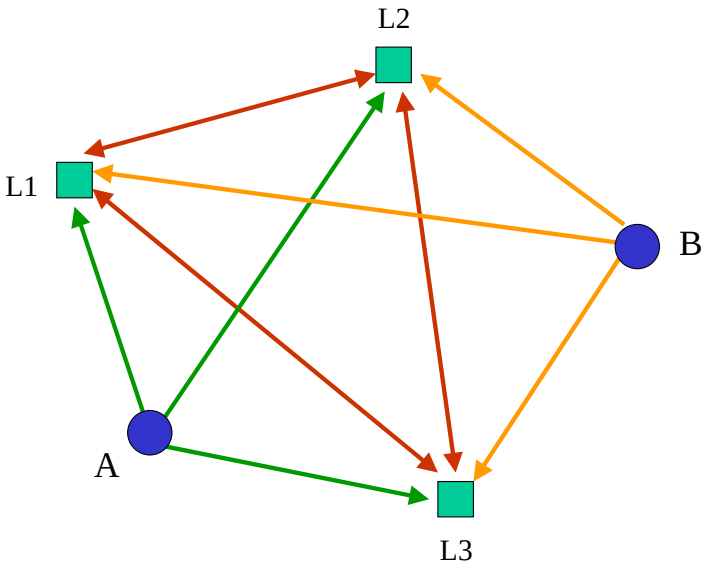


Figure 5-1. Global network positioning model (GNP)

In the binning strategy, rather than utilizing coordinates, hosts measure their actual distances to the landmarks and place themselves in a bin based on the sequence of sorted distances to the landmarks. The idea is that hosts, which are close to each other relative to the other hosts, will have similar sequences and therefore will be placed in the same bin or in a similar one. The binning strategy assumes that fine-grained reduced errors between measured and predicted distances do not have a considerable effect on the application performance. As a result it is possible to make distance comparisons by using less information namely, only a sequence of estimated distances to a set of landmarks. However, the approach is not as flexible as the coordinate-based approaches. This is due to the fact that in the binning strategy bins are tightly bound to the landmarks, whereas in coordinate-based approaches the coordinates and the landmarks are relatively loosely coupled.

The Lighthouse approach (Pias et al., 2003) shares the same idea of eliminating the landmarks to achieve scalability, as the proposed approach in this chapter. The scheme uses a transformation matrix maintained by the hosts for making conversions between the

global and the local bases. However, the approach does not address the practical implications, namely, it is not clear how the global basis is formed and the definition of the local basis is not consistent with the positions of the hosts since the hosts are picked arbitrarily. Moreover, every host that wants to participate needs to create a local basis. While it is also not clear how to find a host that is currently participating in the system, it does not bring any advantage over GNP in terms of accuracy of the distances.

The results of GNP study (Ng & Zhang, 2002) show that, the coordinate-based approach provides better estimation of distances than the IDMaps. It is also more accurate and robust due to its p2p architecture when compared to IDMaps. However, this approach has its own drawbacks. First, the landmarks must be globally distributed to the entire Internet to accurately measure the distances between two arbitrary distant hosts. But the distance estimation between two local hosts is biased with the coordinates of the widespread landmarks, which are not local to the given two hosts. Second, these landmarks are central points of failure and may become target of attacks. Third, where to locate the landmarks, how many landmarks there should be and how/where to move the landmarks or add new ones as the Internet topology changes and grows are important open scalablity problems. Finally, there is a security concern of the system. The authors point out that (Ng & Zhang, 2002) this system may not be suitable in an uncooperative environment since there is nothing to prevent a host from lying about its coordinates for being chosen or not chosen depending on the application. To address these shortcomings we propose the coordinate-based *Pure Peer-to-Peer Network Positioning* (Triple-P NP or *TPNP*) architecture, which completely eliminates the landmarks and the problems they introduce.

5.3 TPNP Approach

The idea of TPNP is similar to the basic principle of dynamic distance vector routing, except that hosts do not need a global picture (i.e., a routing table) of the network. A host that is interested in participating the TPNP system first discovers the nearby hosts that are already in the system to use them as *reference hosts*, which replace the landmarks for the host in question. Then, the host contacts these references to compute its own coordinates (see Section 5.3.1). Figure 5-2 illustrates the same hosts *A* and *B*'s (from Figure 5-1) coordinate computation process. A uses three other ordinary hosts *C*, *D*, and *E* already in the system, and *B* uses *F*, *G*, and *H* locally. These hosts, which are already in the system, are not shown in Figure 5.1 for clarity. As shown host *C* may have used three other locally positioned ordinary hosts for computing its own coordinates.

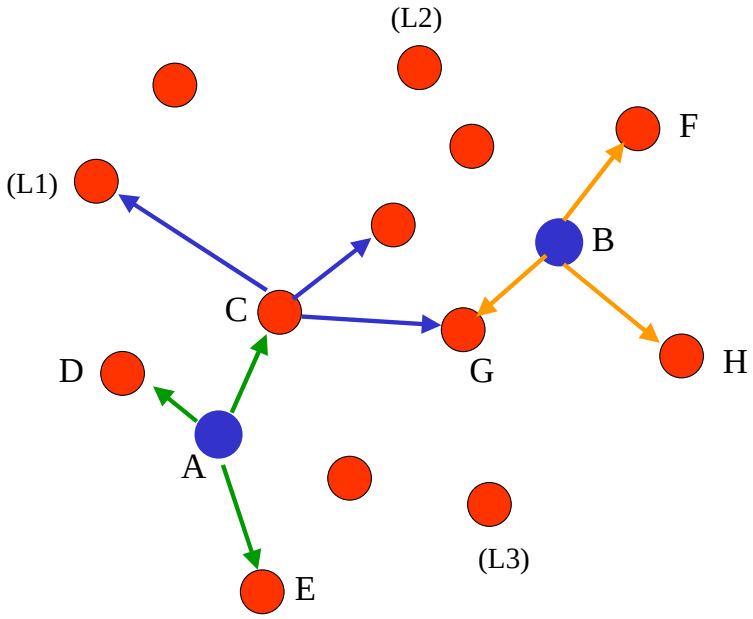


Figure 5-2. Local network positioning model (TPNP)

Unlike GNP or any other global positioning system local measurements as of TPNP are likely to give more accurate results between local hosts. There are two

important facts that support this argument. The first one is that the network traffic of hosts that are close to each other in terms of network distances tend to be routed the same way in most cases. For example, suppose we are interested in measuring the distances between two hosts in Asia. Landmarks, in Europe or North America would not contribute to the accuracy of the estimation. For instance, congestion happened in North America will not affect the end-to-end delay in Asia. In fact, those landmarks at distant locations may have a negative effect on the accuracy. In the global case, we show that the local measurements will have at least, comparable accuracy to that of GNP. For the other fact suppose there are two hosts in the same region (i.e., a country) and the network traffic originated from these two hosts to outside of the region follow different paths, which might have different characteristics (e.g., bandwidth, delay, average load). So, the coordinates of these two hosts that are computed relative to the globally distributed landmarks may seem totally unrelated (i.e., far away). However the hosts may be connected with a reasonably fast metropolitan area network, which makes the distance between the two very small.

Majority of the positioning systems including the ones, which are based on coordinates, do not require on-line distance estimations. In fact, this is one of the major advantages of coordinate-based systems. To figure out the distance between any two hosts in the network, all we need to know is the coordinates of these hosts. However, the Internet is an extremely dynamic ever-changing environment. Therefore, the coordinates should be recomputed by the hosts periodically in order to reflect the changes to provide up-to-date distance/delay information. The next subsection presents the algorithm for a

host to join the system. Recomputing the coordinates of an already participating host is very similar and therefore is not given.

5.3.1 The Algorithm

The steps of the basic algorithm to join in the TPNP system by a host are:

1. Discover nearby hosts that are already in the system.
2. Select a subset of the hosts discovered.
3. Measure distances (i.e., delay) to every selected host.
4. Compute own coordinates.
5. Store and advertise own coordinates.

Step 1: Discovery. The discovery algorithm is given in Figure 5-3. The algorithm uses IP-multicast. Two types of messages are used:

Multicast message: an IP packet indicating a TPNP inquiry with a specified Time-To-Live (TTL) value.

Response message: an IP packet, with a payload of: 1- coordinates of the responding host, 2- coordinates and IP addresses of the reference hosts of the responding host.

The algorithm gradually expands the multicast ring to find necessary number of reference hosts. The first phase of inquiry process takes place in lines 01 through 10. If the predetermined number of hosts cannot be found and the threshold value for the multicast depth is reached, lines 14 through 18 are executed to check whether at least one reference host was found. If this is the case further references of the reference hosts are also to be used by the current host. If the current host found that there was no candidate reference hosts then the multicast ring is expanded by using multicast depth values between d_{thresh} and d_{max}.

```
DEFINE
Hr : set of hosts that replied to the current  TPNP inquiry
Hn: set of newly discovered hosts responded to the current TPNP inquiry
N_TPNP : number of reference hosts to be used globally in TPNP
d : multicast depth
d_init : predetermined initial value of d
d_thresh : predetermined threshold  value of d
d_max : predetermined maximum value of d
t_inq : response wait time for the inquiries

01      Hr = φ;
02      d = d_init;
03      REPEAT
04              Hn = φ;
05              multicast an inquiry message with TTL = d;
06              wait(t_inq);
07              check responses;
08              Hr = Hr ∪ Hn;
09              increment(d);
10      UNTIL   |Hr| ≥ N_TPNP   OR   d > d_thresh;
11      IF      |Hr| ≥ N_TPNP
12              RETURN(Hr);
13      ELSE
14              FOR EACH (i) responding host in Hr
15                      FOR EACH (j) reference host used by Hr_i
16                              IF   Hr_ij ∉ Hr
17                                      Contact Hr_ij and obtain reference host info;
18                                      Hr = Hr ∪ {Hr_ij}
19              IF   Hr == φ
20                      REPEAT
21                              Hn = φ;
22                              multicast an inquiry message with TTL = d;
23                              wait(t_inq);
24                              check responses;
25                              Hr = Hr ∪ Hn;
26                              increment(d);
27                      UNTIL   |Hr| ≥ 1   OR   d > d_max;
28                      IF   |Hr| ≥ 1
29                              EXECUTE 14 through 18
30                              RETURN(Hr);
31                      ELSE
32                              // No participating hosts found in the vicinity,
33                              // Contact DNS for global host information.
```

Figure 5-3. Discovery algorithm

The same procedure is then repeated in lines 20 through 27. If there is still no response from any hosts, then the algorithm gives up. In this case, the DNS needs to be contacted (see Step 5) to figure out some global hosts, which are participating in the system, since there is no local participating host in the vicinity.

Step 2: Selection. The selection of reference hosts from the candidates determined by the discovery procedure above is presented in Figure 5-4. The algorithm uses a heuristics to perform its job as explained below.

IF $|Hr| = N_{TPNP}$

Hr is the final set of hosts to be used as references;

IF $|Hr| \geq N_{TPNP}$

apply the selection heuristics:

A) Pick the hosts, which reside in outside domains or use greatest number of hosts in different outside domains as their own reference hosts,

B) Pick the hosts, which are closest to origin in Euclidean space (i.e., the ones with absolute smaller coordinate values),

C) Pick the closest hosts (i.e., quick responding hosts),

and prepare the final set of hosts as references.

Figure 5-4. Selection algorithm

(A) in Figure 5-4 tries to ensure that the coordinates converge not only locally but also globally. (B) tries to ensure that the system/coordinates converge(s) toward a fixed point to prevent chaos. Coordinates may go out of control over time due to absence of fixed hosts (e.g., landmarks), accumulated error terms and periodic recalculations. Similar to the concept of combining GMT and atomic clock for providing global time and preserving accuracy of time dissemination, we use a single point that never changes its coordinates to solve the similar problem. For example, this point can be set to *O(0, 0)* as the origin, assuming a two-dimensional space. A computer engineering department of a university, for example, may configure all its hosts to serve this purpose. Except hosts closest to the origin no other host needs to contact to this origin. When computing the

coordinates for the fist time, a host can choose the reference points that are closer to the origin as compared with the others. In recomputing coordinates, they may also consider their own coordinates to select the reference points with respect to the origin.

(C) tries to minimize the overhead into the network due to distance measurements, since sending ping packets (see Step 3 below) to closest reference hosts keeps the process local and therefore only a small part of the network is affected. The discovery part of the algorithm (Step 1) guarantees (C) and therefore it is already taken into account in the first two. So, it is applied when the first two cannot make a decision.

Step 3: Measuring distances. The distances (delay) are measured using ICMP ping. However, it is well known that ping can easily be abused. The protocol in Figure 5-5 was designed to use ICMP ping in a secure manner for TPNP. The s*ource* in the protocol is the host to join in the TPNP system. The *target* is one of the reference hosts (nodes) used by the source.

1. Source connects to Target and asks for ping permission.
2. If the Target grants the permission, it returns a random ephemeral port number to be used for ping along with a one-time password to Source. If Target is busy serving other requests, it may simply respond with a “contact later” message.
3. Target opens the ping port by starting the ping server listening on the ping port.
4. Source pings the Target on the ping port. The ping packets include the one-time password.
5. Target checks the password and if valid responds to the request, otherwise the packet is discarded.
6. After the predetermined number of ping requests received or time-out occurred, Target closes the ping port.

Figure 5-5. Distance measurement protocol

Step 4: Computing coordinates. A host, which is computing its own coordinates, needs the coordinates of its reference hosts and the distances to these hosts. In this step, this information is ready as the result of the previous steps. TPNP uses the coordinate computation process of GNP (Ng & Zhang, 2002). This process is a non-linear

computation that involves a minimization function. It is subject to future research to use linear functions similar to the ones proposed by Lim et al. (2003) and Tang and Crovella (2003).

The coordinate computation process tries to find a solution to a multi-dimensional minimization problem. It is formally the minimization of the objective function

$$f(.) = \sum_{i,l} \varepsilon(m_{ij}, p_{ij})$$

where ε is an error measurement function, m_{ij} and p_{ij} are measured and predicted distances among the hosts, respectively. Informally, the error measurement function computes the difference between the measured and the predicted distances. The one we use is the normalized error measure, which was rated the best by Ng and Zhang (2002):

$$\varepsilon(m_{ij}, p_{ij}) = \left((m_{ij} - p_{ij}) / m_{ij}\right)^2$$

The inputs to the process are measured distances obtained in the previous step in the form of a matrix and coordinates of the hosts obtained in Step 1. At each iteration the minimization function is computed and new computed coordinates are assigned to the host as the host's coordinates and used by the following iteration. Iterations continue until the error is reduced below a predetermined minimal value. Final coordinate values minimize the overall error between the computed and measured distances from the current host to the chosen reference hosts.

Step 5: Storing and advertising coordinates. A host, which completed the previous steps should store its coordinates and make them available to the outside world. The coordinates may be required by the other hosts for two purposes. First one is for applications, which would benefit and therefore use distance information between the hosts in the Internet. The second one is that other hosts that would like to join in TPNP

may request coordinate information from the current host. The same is true for hosts that may need to recompute their coordinates.

In addition to making the coordinates available for application purposes at each host, DNS can be used to provide host-name/location or IP-address/location resolution that would eliminate the need to contact hosts for coordinate information.

Lastly, the current host should join the IP-multicast group reserved for TPNP to receive and reply TPNP requests from other hosts, which might like to join in TPNP afterwards. The host should also prepare for handling ICMP ping requests as explained in Step 3.

5.3.2 Starting the System

One of the major issues in realizing TPNP is how to start up the system. In most server-based distributed systems, servers are first configured and then start running. Problems such as load balancing, scalability, proximity and fault-tolerance arise later and continue throughout the lifetime of the system. Within TPNP however, the opposite is true. Once the system is started the problems mentioned diminishes and even disappears over time with the increase in the number of participating hosts. There are two ways to start up the system, namely, sequential joining from the beginning and participation of a set of hosts in parallel. The former scheme is used in our simulations and explained in Section 5.4. The latter can be achieved by simultaneous participation from fixed sites (such as universities) with centrally and statically computed coordinates as in the GNP approach. However, in TPNP this is to be done only once, and these hosts might even cease to exist over time with no effect on the system. Furthermore, the number of such hosts needs not be greater than the number of reference points desired.

One way to figure out the participating hosts in the system is to use IP multicast as used by the algorithm given in the previous section. It would be safe to assume that the first responses come from the closest hosts. This scheme fits nicely into the local distance estimation in TPNP because it is very easy to find the closest hosts using multicast. Otherwise, it would not make sense to talk about “closest hosts” when we are already discussing a distance estimation scheme. One nice side effect of the peer finding process in TPNP is the fact that every host discovered by a new participant reveals more hosts giving more than enough nearby hosts for references.

The alternative is to use DNS to explore the participating nearby hosts in the system by exploiting the Local and Authoritative Name Servers. However, although DNS hierarchy gives clues, it is not easy to find the distant hosts especially during the early stages of the system. Further research is necessary to determine how best DNS could be as an alternative with minimal changes and overhead to the existing system.

5.3.3 Scalability Issues

Contrast to the common scalability problem in distributed systems, system growth has a positive affect on the TPNP system due to its p2p architecture. The GNP approach is clearly more scalable when compared to a centralized client/server model like that of IDMaps. However landmarks, which are cornerstones in GNP introduces other important scalability problems as explained in Section 5.2. Since the TPNP eliminates the landmarks completely, these problems disappear altogether.

However, use of multicast as the discovery mechanism introduces load into the underlying network. But this overhead diminishes and even disappears as the number of hosts participated in the system increases. In fact, in most cases, the overhead of multicast is observed only when a new host wants to join in the system. Even then, if the

number of the nearby hosts, which are already in the system, is no less than the required number of hosts, the multicast overhead will be minimal. For the same reason, hosts do not have to measure their distances to distant landmarks which will reduce the overhead introduced into the network. Therefore, it can be said that the approach is *positively scalable*, in terms of the load introduced into the network for both multicast and delay measurements.

5.3.4 Security Considerations

The TPNP approach also addresses the security of the system. The security aspect of TPNP is two-fold. First, as pointed out before, GNP cannot prevent hosts from lying about their coordinates. For TPNP, because hosts compute their coordinates relative to nearby hosts, and these nearby hosts have to provide their reference points along with the coordinates, it is possible to check using this information whether a host's coordinates are correct. Heuristics can be devised to verify the correctness of the host coordinates by asking the coordinates of the reference hosts of the target host. Several levels of security can be provided at the expense of communication overhead by increasing or decreasing the number of hosts to check.

The second security problem in landmarks is that they may become target of attacks since they must answer each ping request directed to them. The problem is more prevalent for TPNP since each participating host would have to answer any ping request coming from any host in the Internet. The solution we propose is to customize the use of ping. The distance measurement protocol given as Figure 5-5 serves this purpose.

5.4 Experimental Evaluation

We have conducted simulation experiments to determine how TPNP performs, that is, how accurate the predicted distances among the hosts will be in the system.

5.4.1 Simulation Environment

Our simulation system employs a hierarchical structure, which models individual Internet domains (i.e., autonomous systems) either as an Inet or a Waxman topology. Every domain in a given topology has an equal probability of having either a Waxman or Inet model, and has any number of nodes ranging from 20 to 200.

In the Waxman model, the nodes are placed randomly in a two dimensional grid. Links are added pairwise between all the nodes in the network by considering whether a link should exist according to a probability function, which takes into account how close the nodes are to each other and how many links are expected to exist in the network.

The borders between domains use the Inet, which provides a hierarchical, power-law topology. Earlier studies have shown that the Internet follows a power-law topology (Faloutsos et al., 1999; Tangmunarunkit et al., 2001). These studies report on several power-law relationships observed on domains' connectivity degree, degree frequencies, and the neighborhood size within any given hop count from a domain. For example, the first power-law given by Faloutsos et al. (1999) states that the outdegree of a node is proportional to the rank of the node to the power of a constant where the rank is the node's index in the order of decreasing outdegree.

A more recent study by Ratnasamy et al., (2002) related with ours also confirms that results obtained with power-law topologies better match with the actual Internet traces.

To apply GNP in our simulation environment and to compare with TPNP we have integrated the GNP software by Ng and Zhang (2002). We have used the software for GNP computations and for TPNP where possible, without modifications other than the ones necessary for integration.

5.4.2 Simulation Parameters

In our experiments, we used 10 landmarks for GNP and 10 reference nodes (peers) in TPNP (except the first 10 nodes to join). The number of dimensions is two for both. It is important to note that the choice of ten landmarks and two-dimensional coordinate system is for simulation simplicity and efficiency. Our aim is to compare the two approaches and therefore our first priority in selecting parameters is to ensure fairness. Using only two dimensions has the advantage of much less computation overhead. When the number of landmarks and reference points increases, both approaches benefit and GNP benefits more since the results with two-dimensional space are not very accurate and there is more room for improvement. As reported by Ng and Zhang (2002), there is a saturation point for both number of dimensions and reference nodes. We confirm the results for GNP and report that the same is true for TPNP. For prediction accuracy, we use the same performance metric by Ng and Zhang (2002), which is the following:

$$\text{Relative Error} = \frac{|\ measured - estimated\ |}{\min(measured, estimated)}$$

Our choice of the metric as opposed to the simple percentage or simple ratio metrics, which are used by Gummadi et al. (2002), Ratnasamy et al. (2002), and Pias et al. (2003) is due to the fact that while percentage error metric hides underestimates, ratio error is not suitable to compute average undirectional error due to the magnitude difference of over and underestimates.

5.4.3 Simulation Strategy

Our simulation strategy is to obtain the coordinates of all the hosts in the topology and therefore estimations for all the shortest path distances among every pair of nodes.

For TPNP, we have used the downhill-simplex method (Ng & Zhang, 2002), which used for GNP for minimizations.

To select the best possible nodes to be landmarks for GNP, we implemented the N-cluster-medians technique, which was rated the best (Ng & Zhang, 2002). We applied the technique to the whole network to find the nodes, which represent all the nodes with an aggregation based on proximity. It should be noted that this technique is only feasible in a simulation environment and in favor of GNP.

All of the results presented in this chapter were obtained by using three different topologies of roughly the same size and average values of the three simulation runs are reported. This is to make sure that the results are repeatable which is especially important for TPNP's stochastic behavior due to random node join order.

To simulate the actual formation of the TPNP system, we pick a random node on the whole network and assign the coordinates of origin *(0,0)* of the Euclidean system for convenience. Since we are using a 2D Euclidean system, only the second and the third nodes' coordinate computation process differs from the rest. For these two nodes, we measure distance of shortest paths between them and previously joined nodes. We place the second node on the x-axis and pick the positive value for the third node among the alternatives for convenience. For the rest of the nodes, the same minimization technique of GNP is used, however, 4^{th} through the 10^{th} nodes use the available number of reference points while the rest use exactly 10 of them.

5.4.4 Simulation Results and Analysis

As seen in Figure 5-6, when the number of domains increases, prediction performance of GNP is negatively affected. This is the case even though the landmarks are chosen using the N-cluster-medians of the entire network.

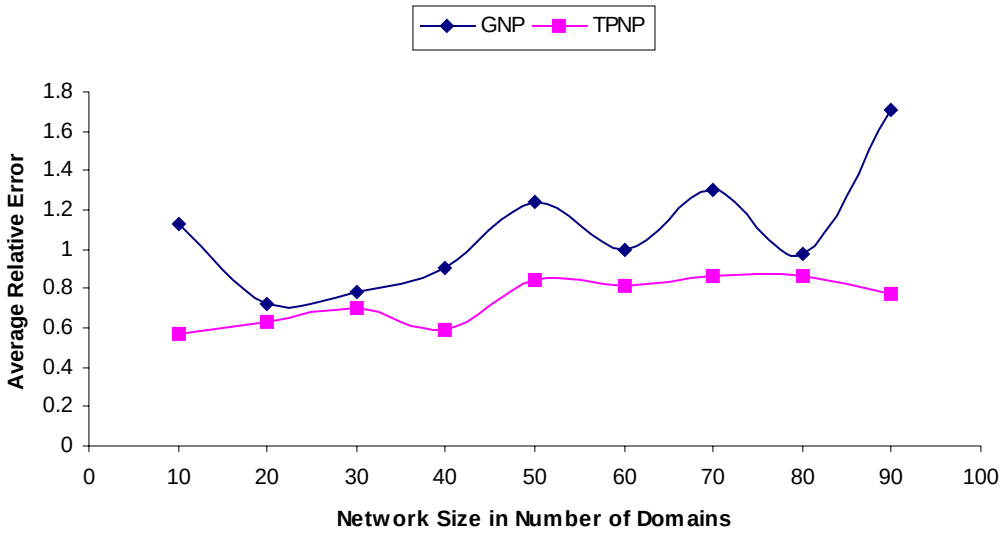


Figure 5-6. Average relative error

It is clear from the figure that using a pure p2p approach without landmarks, better accuracy can be achieved. This is again due to the fact that global landmarks are not adequate for local distance estimations.

In Figure 5-7 we have plotted the intra-domain distance predictions of the both approaches with increasing network sizes. The figure shows clearly that the local (intra-domain) relative error remains constant in TPNP regardless of network size and the number of domains. However GNP's intra-domain relative error is correlated with network size. Moreover, even with a fairly small network size of only one thousand nodes, which is the first point in Figure 5-7 with 10 domains, GNP's accuracy is still not as good.

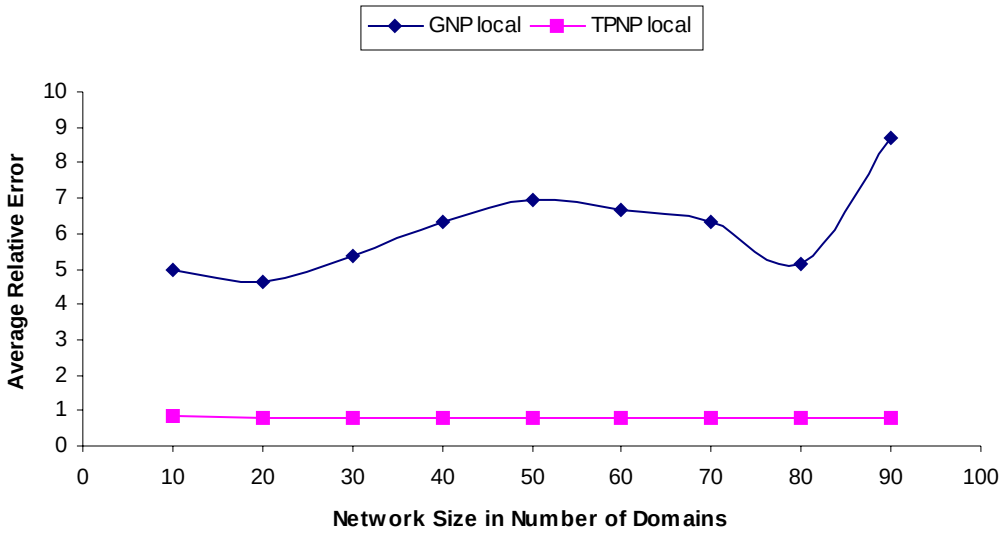


Figure 5-7. Average local relative error

Although excluding outliers in statistical data has merit, in practice it is desirable for a system to be as fair as possible to its users or applications. Since we expect some form of uniformity from TPNP, we also plotted the graphs for average relative error of values greater than 1, which we call *anomaly* in Figure 5-8. As expected, TPNP provided the uniformity of anomalies with also smaller error terms when compared to GNP. As it is clear from the figure, GNP has larger anomalies, which scale with the network size.

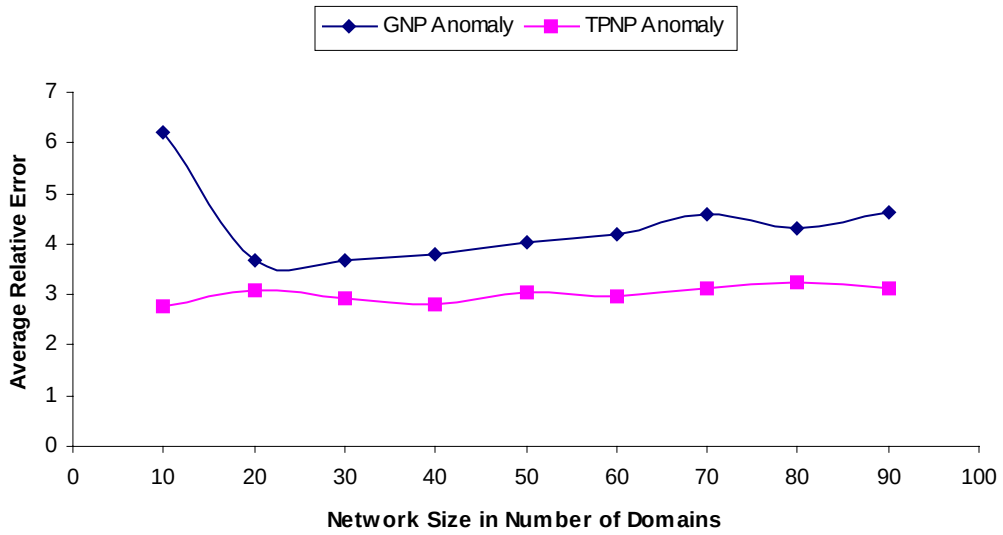
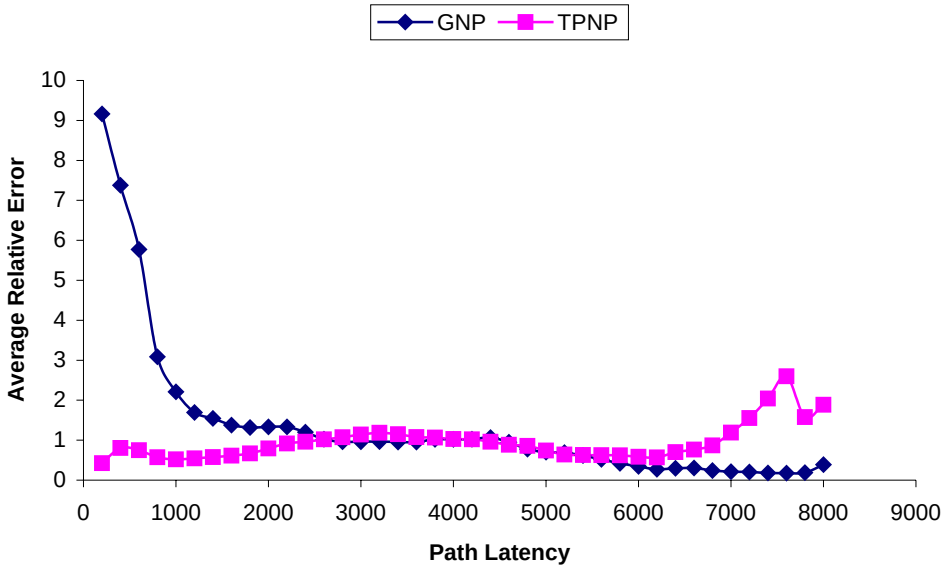
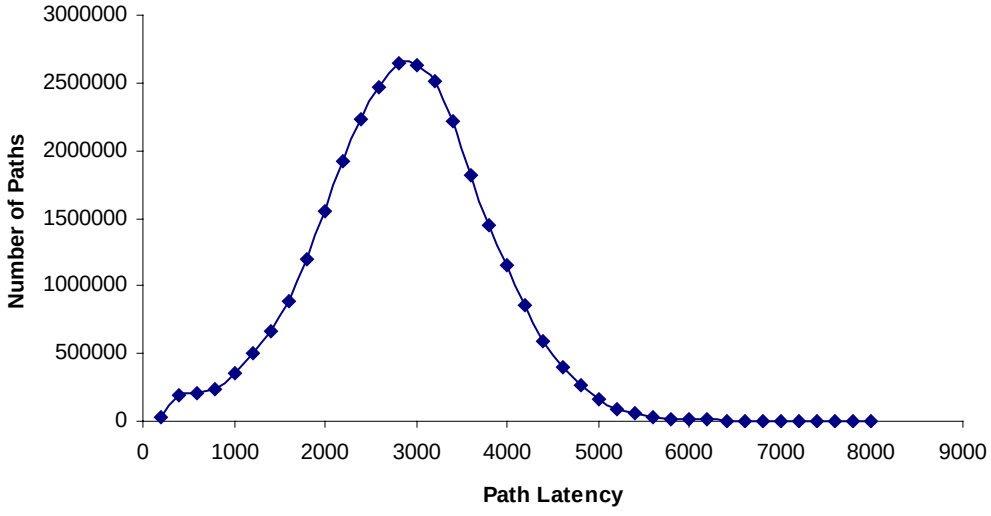


Figure 5-8. Average outliers relative error

Within TPNP, due to cascading coordinate computation, one might expect that the predicted distances would be less accurate due to accumulated errors when the number of hops (peers) increases between any given two hosts. To provide an insight, we plotted the graphs in Figure 5-9. These graphs have been obtained from 7000-node topologies.



A



B

Figure 5-9. Distribution of average relative error to path latencies

Figure 5-9.A shows relative error distribution to the path latency intervals. We created 200 ms intervals of path latency starting from zero and plotted the prediction errors of all the paths whose measured distances fall in that interval. Figure 5-9.B shows the distribution of paths into these latency intervals in the network. With an average of 100 ms of hop latencies in the network a slight increase arises at 7000 ms point for TPNP. This means that error accumulation due to many predictions across the network due to the p2p nature of TPNP, may not be significant. However, until the 2500 ms point, GNP predictions of shorter paths are not as good. GNP's accuracy clearly increases when the path latency increases due to very well-positioned landmarks in the simulation environment, but the number of paths in this upper range is fairly small as seen in Figure 5-9.B.

5.5 Conclusion

In this chapter we presented a p2p coordinate-based network position estimation scheme that does not rely on landmarks for coordinate computation as in previously proposed approaches. The goal of eliminating the landmarks is to achieve greater scalability. The overall architecture is presented and the goal justified. The simulation analysis also shows that the system performance in terms of local distance prediction accuracy can be significantly improved over GNP. Even with a two-dimensional Euclidean space its overall performance is within an acceptable range. The pure p2p nature of the architecture lends itself very well with the robustness of the system. Some security issues are also addressed.

In the next chapter, TPNP is used for the quantitative analysis of multiple mobile agent systems. It is also made clear how TPNP provides context-awareness for these

systems. Moreover, we show in the next chapter that a positioning system is also proved useful for traditional single mobile agent systems.

CHAPTER 6
QUANTITATIVE ANALYSIS OF MULTIPLE MOBILE AGENT SYSTEMS

6.1 Introduction

In Chapter 5, we have proposed a peer-to-peer network positioning architecture. We have said that a general-purpose system such as TPNP, which is based on this architecture could provide coordinates of the hosts in a network and simple off-line computations could use this coordinates to estimate the inter-host distances. This chapter shows how the knowledge of inter-host distances could be useful for mobile agents and what would be the mechanism to do this. We are basically looking for the answers to two important questions: can we alleviate the performance problem introduced with multiple agents by making them network-aware? and can the overhead of off-line computation be justified? In addition to answering these questions, we will explore more details on the network-awareness and context-awareness concepts in the context of multiple mobile agent systems.

In this study we do not target a specific mobile agent application. Any application, which does not impose an ordering of hosts to be visited, is a potential application. Examples are information search and retrieval, software distribution/update, remote education and e-commerce.

Note that in this chapter, we use the master/support agent model, which is given in Chapter 3. We use terminology specific for the chapter's purposes and this needs a notation change from previous chapters. The *MA* stands for *Master Agent* rather than

Mobile Agent, and *SA* is similarly the acronym for *Support Agent*. We also refer to the itineraries for both types of agents as *master itinerary* and *support itinerary* for short.

6.2 Network-awareness in Mobile Agent Computing

Network-awareness is a broad concept. What we mean with being network-aware in the context of mobile agent systems is to be able to obtain distance (i.e., delay) information among the hosts. A common use of distance information is to locate the closest server or object. The problem could be generalized and formalized as finding the nearest neighbor in a given finite set of nodes in some domain. In mobile agent computing however, we add the concept of *itinerary*, which is a list of nodes to be visited by an MA(s). The concept is rather vague in the mobile agent literature. Usually, itinerary in this context means that a mobile agent has an ordered set of hosts to visit. If the ordering is not important or defined, the mobile agent is said to be "itinerary-less." Our view is however different, we define an itinerary for a mobile agent as *the set of hosts to be visited for a given mission*. This means every mobile agent has one. However the order of the visits may or may not be enforced by the itinerary. Moreover, the itinerary may be fixed as well as flexible meaning that it may change during the mission depending on the application. The flexible itinerary concept raises some interesting issues for future research.

One of the most important characteristics of the software agents in the field of artificial intelligence is, without a doubt, their being intelligent entities. In mobile agent research the intelligence of agents is mostly assumed by referring to the general field of software agents. However, artificial intelligence research cannot address the intelligence in mobile agent migration because software agents are static entities. Even if the mobile agents used, for example, learning algorithms in some specific application, we still could

not say much about their intelligence, if they still were “wandering around” the network to accomplish some distributed task. We need ways of making agents intelligent using network-awareness concept in terms of distance information. Not surprisingly, given an itinerary and the distances among the hosts in the itinerary, the problem becomes an instance of the well-known Traveling Salesperson Problem (TSP).

6.3 The Traveling Salesperson Problem

The TSP is a classical problem in combinatorial optimization. The general classical TSP is defined as follows: A salesperson starting from the home city is required to visit *n* cities without repetition before returning home. The objective is to find out the minimum-cost tour given the inter-city distances. The input to any classical TSP instance is a distance/cost matrix. The output is a permutation of cities, which represents an optimal tour. There are many variants of TSP such as *symmetric TSP* where the distances of edges in a graph are the same in either directions or *asymmetric TSP* otherwise (Lawler et al., 1985).

The definition of the classical TSP can be directly applied to mobile agent computing and simply be called the Traveling Agent Problem (TAP). However, just as the original TSP, different mobile agent applications are expected to require variants of the TAP. For example, an application may employ several clones of an agent, which would correspond to the multiple salesman or vehicle routing problem (VRP) (Lawler et al., 1985).

The classical TSP however, is not directly applicable in our case since we do not have inter-host distances but only coordinates of hosts in the Euclidean space. For a given instance of the problem, these distances can be computed easily using the coordinates, however, this computation is not efficient. Therefore what we need is a variant of TSP,

which is known as the geometric (or Euclidean) TSP. In geometric TSP the input is *n* points and their coordinates in k-dimensional space and the output is a permutation that corresponds to an optimal TSP tour. The length of the tour is the sum of the distances between adjacent points under a specified metric such as the Euclidean metric. The geometric TSP is known to be NP-hard (Lawler et al., 1985).

TSP is the most attacked problem in combinatorial optimization. Since the problem is NP-complete, most of the research is focused on heuristics to find near- or sub-optimal solutions and their analysis. The TSP literature has a vast amount of material on the heuristics and the consequence is that there is a large number of variant of the problem and heuristics to choose from. The heuristics are classified as follows (Bentley, 1992; Lawler et al., 1985):

Heuristics that grow fragments. The algorithms in this category maintain fragment(s) of tours. The main body of the algorithms consists of a loop. While the tour is being built in the loop, the fragment represents a path of the subset of points. At each iteration of the loop a new point is added to the fragment. Finally, the two ends of the fragment, which corresponds to a complete path of all the points, are connected together to yield a complete tour. Depending on the heuristic one or more fragments can be maintained simultaneously.

Heuristics that grow tours. As the name implies, these are the algorithms that maintain complete cycles of points as tours, instead of fragments. There are three selection rules (nearest, farthest and random) and two expansion rules (addition and insertion). Thus, the heuristics are cross product of these rules. The selection rules are explained in Section 6.4.2. In the expansion rules only the definition of point expansion

differs. The algorithms' main body consists of a loop, which maintains a complete cycle of a subset of points as a partial tour. Starting with a single point, at each iteration of the loop the tour is expanded by choosing a new point by any of the selection rules and then choosing where to add it by using one of the expansion rules. Note that the terminology is a little confusing since addition and insertion deal with where to add the new point but not how to add/insert it. A new point is added/inserted by deleting an edge and adding two new edges.

Heuristics based on trees. These are the heuristics, which are based on a structure in a graph such as a Minimum Spanning Tree (MST) or directly on an underlying data structure such as a K-d tree (see Section 6.4.1). The algorithms that are based on an MST first build the tree. Then they traverse the tree to yield a tour. According to the algorithm chosen, the operations performed during tree traversal differ. The second category of the algorithms traverses directly the physical tree structure to build a tour of the points stored in the tree.

Local optimization heuristics. The three groups of heuristics above start with a set of points and actually build the tour. Thus, they are referred to as starting heuristics. This last group of heuristics however makes local improvements to the given tours already built by any of the algorithms mentioned above. Thus, the input to the algorithms is a complete tour and the output is a shorter/better tour. The algorithms in this group are identified by using the notation λ-Opt. λ indicates how many edges are removed and added at each step of the algorithm. 2-Opt, 2H-Opt and 3-Opt algorithms are used to perform local optimizations. However 4-Opt algorithm is no longer a local optimization one. The 2H-Opt algorithm deals with 5 points instead of 4 as in 2-Opt or 6 as in 3-Opt.

6.4 Application of TPNP and TSP to Classical MA Model

Bentley (1992) provides a detailed analysis of the heuristics and reports results of the fast algorithms and their implementations. We picked one heuristic from each of the four categories above and implemented them. These are: Nearest Neighbor (NN), Random Addition (RA), Fast Recursive Partitioning (FRP) and 2-Opt, respectively. Our selection criteria is as follows:

Performance. Algorithm's ability to produce shorter tours, which is analyzed theoretically or measured by experiments.

Efficiency. Algorithm's ability to produce the tour quickly. In addition to asymptotic complexity we are also interested in actual running times based on experimental results.

Robustness. A heuristic is said to be robust if its performance with different input distributions is not far from its performance for uniform distribution. Clearly it is a relative metric. Bentley (1992) and Bentley (1990b) provide ten such different distributions and analyze the robustness of heuristics mentioned above.

Simplicity. Heuristic's understandability and ease of implementation.

6.4.1 The Data Structure

Our implementation is based on a data structure known as K-d tree proposed by Bentley (1975). K-d tree stands for K-dimensional binary tree, which generalizes single dimensional ordinary binary trees. We use a more efficient variant of the K-d trees, which is semidynamic K-d tree (Bentley, 1990b). A semidynamic K-d tree allows deletions and undeletions of nodes but new nodes cannot be added once the tree has been built. However, it is sufficient for our purposes with added advantage of efficiency, due to the assumption that the itinerary of an agent (set of hosts to be visited) is known before the

mission starts at the home. A K-d tree supports several operations such as nearest neighbor searching and fixed-radius near neighbor searching (Bentley, 1990b). The nearest neighbor procedure accepts as input a point and returns the nearest point in the tree to the input. The fixed-radius near neighbor procedure accepts two points. The first point is the target and the second one is to define a ball, whose radius is the distance between the two points and the center is the target. The procedure returns the closest point in the tree to the target which lies in the ball. The basic operations on K-d trees are used to solve several closeness problems such as minimum spanning trees and TSP heuristics including the local optimizations (Bentley, 1992).

The K-d tree can be built in $O(KN \log N)$ time (Bentley, 1990a). Sampling techniques reduces the complexity to $O(KN + N \log N)$ (Bentley, 1990b). Semidynamic K-d trees allow bottom-up operations to be performed in $O(1)$ time which would take $O(N)$ time for a procedure that starts at the root (Bentley, 1990b). All the implementations of the algorithms in this chapter are based on K-d trees and therefore bounded by their complexity of the operations used. These are building the tree, deletion, undeletion, nearest neighbor searching and fixed-radius near neighbor searching.

6.4.2 The Heuristics

The NN heuristic is a simple greedy algorithm. It starts with a point and connects the nearest point to the previous one until the cycle completes all points in the list. A variant of the algorithm enlarges the path by adding points to each end of the fragment.

The RA heuristic (Bentley, 1992) is a variant of Nearest Addition (NA) and Farthest Addition (FA) algorithms. The NA adds the next point as the point that is not in the tour and is closest to any of the points that are already in the tour. The FA does the opposite by choosing the farthest point. The RA however, picks the next point randomly.

Surprisingly, due to random selection it runs much faster than the NA and FA with a little degradation on performance, which is the quality of the tours produced.

The FRP algorithm is based on the underlying K-d tree. It first creates the K-d tree from the set of input points, traverses the tree, and finds short fragments of the points in the buckets of the tree by performing an NN search and connect the bucket fragments together which yields a tour.

The 2-Opt heuristic is a local optimization algorithm. It takes a complete tour as input and produces a shorter tour as output. The 2-Opt means that two edges are removed and two new smaller cost edges are inserted. This procedure is repeated on the tour until no possible optimization remains at which point the tour is said to have the 2-Opt property.

All the heuristics use the nearest neighbor procedure while 2-Opt also uses the fixed-radius near neighbor search procedure.

6.5 Experimental Evaluation

In this section, we present the details of applying TSP heuristics to the single agent model via simulation experiments.

6.5.1 Simulation Environment and Parameters

We have run the code that implements these heuristics in the same simulation environment we used in Chapter 5 where TPNP was also implemented and simulated. The programming language used is C++.

All simulation experiments in this chapter have been done on a laptop computer with a 2-GHz Intel Pentium 4 M processor and 512 MB of main memory, running Windows XP OS. This processor is reported to have a 512KB L2 cache.

Simulation parameters for TPNP are the same as the ones used in Chapter 5. We use 10 reference nodes and the same performance metric, which is the relative error. Unless indicated otherwise, the number of dimensions used in the experiments is 4. We have used networks with roughly 5000 nodes in the simulation runs with the number of domains being 50 and the average number of nodes per domain of 100. The number of hosts in any mission is 100 unless indicated otherwise.

On the TSP side, K-d tree implementations need the bucket size parameter, for which we use 5. This means that the maximum number of nodes in a single bucket may not exceed this value.

6.5.2 Simulation Strategy

The strategy is to pick the hosts in the itinerary randomly from all the hosts in the network. The same is true for the home of the agents. To ensure repeatability, for the random tour computations, we pick 100 different sets of hosts and report their averages. For all the experiments, we pick 3 different host sets for the same reason and report the averages. We do not consider different distributions of nodes because we use the TSP heuristics with known robustness ratios for several different distributions as explained before.

In this part of the study, we measure and report only the length of the tours for a given itinerary. The goal is to figure out how good a tour we can compute using predicted distances in the network. Thus, we use predicted distances in our computations but we analyze the results by using the measured distances.

6.5.3 Results and Analysis

Figure 6.1 shows the computed tour lengths for varying number of hosts. The tours, of which lengths are displayed, all have the 2-Opt property. The scalability of computed

tours are remarkable when compared with the average of random/arbitrary tours. Results of the NN and RA algorithms are so close to each other that they overprint in the graph. The results are consistent with Bentley (1992)'s original results in that FRP performs worse than the other two heuristics. It is important to note here again that we compute the tours with predicted distances, but the values in the figure are the measured distances.

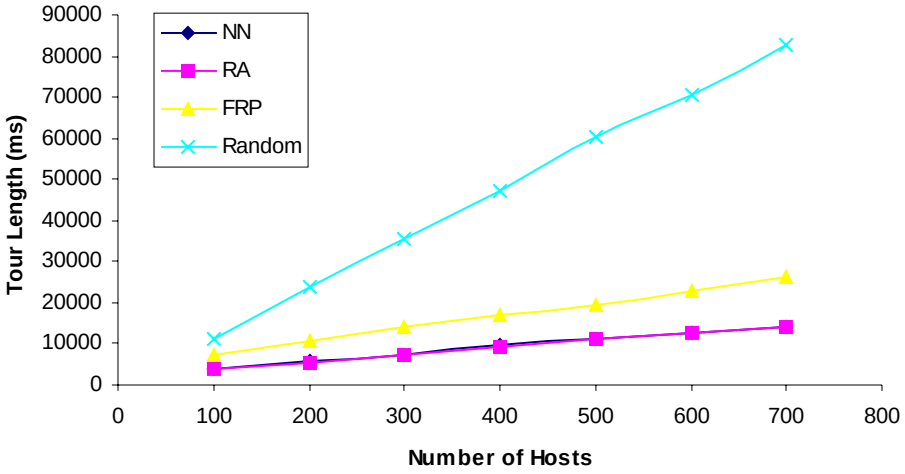


Figure 6-1. Performance of 2-Opted TSP heuristics across the problem size

Figure 6.2 displays the tour lengths with 2-Opt property of the same heuristics with the input of different node coordinate values of varying dimensions in the TPNP system. The Random line is shown as a baseline for comparison. The NN and RA values again overprint each other. It is interesting to observe that as opposed to the theoretical results of distance prediction schemes like GNP (Ng & Zhang, 2002) and TPNP (Chapter 5), increase in number of dimensions after the value of 4 does not reflect any significant improvement or any improvement at all. Closer analysis with varying number of dimensions is presented below.

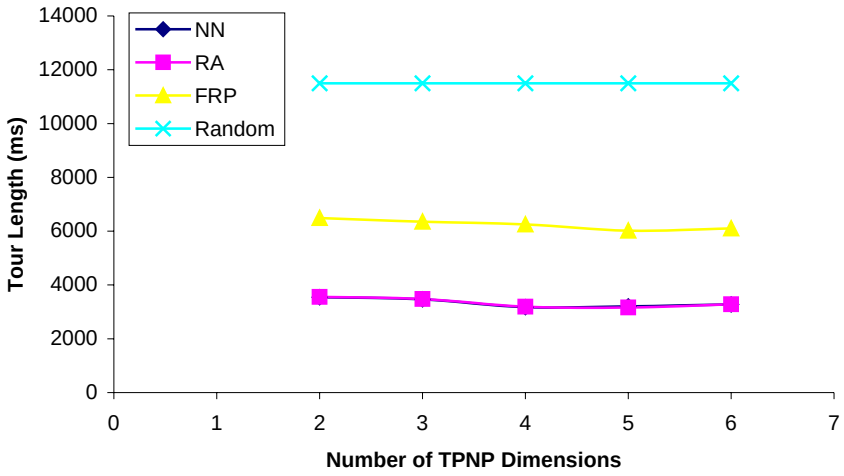


Figure 6-2. Performance of 2-Opted TSP heuristics across the number of TPNP dimensions

Tables 6-1, 6-2, and 6-3 present the results of running the implementations of NN, RA, and FRP heuristics, respectively, including the 2-Opt optimizations applied to all tours generated by each heuristic. The first column of tables indicates the number of dimensions used in TPNP. The next column is the average of random tour lengths created from the same set of nodes. The next set of columns gives the predicted and corresponding measured distances of the tours created by each heuristic along with their relative errors between these distances. The following set of columns shows the application of 2-Opt local optimization heuristic to the computed tours of starting heuristics. The last set of columns displays the percentage improvement of the 2-Opt optimization.

Although the execution times of the algorithms are of great importance, we do not display them since surprisingly each run of any of the algorithms we implemented completes on the average in less than 1 millisecond with default simulation parameter values. This is true also for the extreme cases, for example computing a 2-Opted NN tour with either 700 hosts or 6-dimensional space. The results justifies that employing TPNP

and TSP for mobile agent computing does not impose much off-line computation overhead that would offset the benefit achieved by improved tour lengths. Clearly the off-line computation time is negligible when compared to the improvements on actual measured tour lengths. We should note here that in our simulations we did not use any compiler or code optimizations other than using most efficient implementation of K-d trees. Although there are known code optimization tricks for K-d trees (Bentley, 1990b) for further efficiency improvements, we did not use them in our code to keep it more understandable. The results also indicate that several performance improvements can also be made with small degradation of efficiency since we have a lot of room for total running times. For example, heuristics, which are known to perform better but takes more CPU time could also be employed. For example, instead of using FRP, the original heuristic (Lawler et al., 1985) could be used which takes more time to complete but provides better performance. Depending on the number of hosts in a given mobile agent application, execution times of single digit milliseconds may still be acceptable. Caution however is needed since mobile agents are likely to be employed in mobile computing where mobile devices might not have sufficient processing power and memory as the one we have used in our experiments. Also, the time needed for gathering the network information should be taken into account.

Table 6-1. Application of NN heuristic with TPNP

# Dim	Average Random Tour Length	NN Tour Length			NN + 2-Opt Tour Length			Improvement	
		Pred.	*Msrd.*	*Err.*	*Pred.*	*Msrd.*	*Err.*	*Pred.*	*Msrd.*
2	11494.48	2408.13	4109.37	0.71	1821.03	3401.32	0.87	0.24	0.17
3	"	2701.83	4186.77	0.55	1913.98	3467.35	0.81	0.29	0.17
4	"	2727.46	3731.45	0.37	1932.40	3177.63	0.64	0.29	0.15
5	"	3151.12	3992.68	0.27	2191.03	3205.98	0.46	0.30	0.20
6	"	3220.87	3881.28	0.21	2419.65	3274.62	0.35	0.25	0.16

Table 6-2. Application of RA heuristic with TPNP

# Dim	Average Random Tour Length	RA Tour Length			RA + 2-Opt Tour Length			Improvement	
		Pred.	*Msrd.*	*Err.*	*Pred.*	*Msrd.*	*Err.*	*Pred.*	*Msrd.*
2	11494.48	2017.51	3698.99	0.83	1809.57	3413.57	0.89	0.10	0.08
3	"	2264.22	3804.00	0.68	1911.46	3479.81	0.82	0.16	0.09
4	"	2310.22	3395.87	0.47	1961.92	3196.09	0.63	0.15	0.06
5	"	2460.88	3340.74	0.36	2173.13	3161.17	0.45	0.12	0.05
6	"	2762.21	3492.40	0.26	2406.50	3281.76	0.36	0.13	0.06

Table 6-3. Application of FRP heuristic with TPNP

# Dim	Average Random Tour Length	FRP Tour Length			FRP + 2-Opt Tour Length			Improvement	
		Pred.	*Msrd.*	*Err.*	*Pred.*	*Msrd.*	*Err.*	*Pred.*	*Msrd.*
2	11494.48	5998.78	10012.23	0.67	3503.33	6490.34	0.85	0.42	0.35
3	"	6201.88	9840.16	0.59	3592.91	6350.11	0.77	0.42	0.35
4	"	6500.16	9495.04	0.46	3889.43	6251.32	0.60	0.40	0.34
5	"	6992.51	9300.32	0.33	4112.55	6020.00	0.46	0.41	0.35
6	"	7312.72	9288.71	0.27	4311.45	6102.02	0.42	0.41	0.34

One can argue that comparing random tours with the tours achieved by using heuristics would be analyzing something that could easily be guessed. Clearly, creating intelligent tours with the help of heuristics should outperform the random ones. However, it should be noted that in reality we do not know the actual distances and cannot obtain them under the efficiency constraint. Therefore we have to run the implementation of algorithms with the input of predicted distances. However, we measure the performance

using measured distances. Application of TSP heuristics with predicted input is not necessarily intuitive as explained in the following.

It seems we have two contradictory results. The first one is that predicted tour lengths (*Pred* columns) are increasing with increase in dimensions. The reality is that due to underestimations, the distances seem shorter when the accuracy of the estimations is lower. But while the accuracy is getting better with the number of dimensions, which is clear from the relative error (*Err* columns), the predicted distances grow closer to the actual distance values. So, even we cannot see the direct effect of better tours achieved using heuristics in predicted distances, we actually see the clear improvement in measured distances that we are interested in.

The second is that in some cases in the tables, although the prediction accuracy is better with more dimensions (*Err* columns), the tour lengths are not shorter as expected (*Msrd* columns). It is easy to explain this behavior. When the number of dimensions changes, the nodes are assigned new coordinates in the more dimensional space. The direct consequence is that the predicted distances among the nodes also change, which yields different tours for the same set of nodes.

Increasing number of dimensions in coordinate-based approaches is expected to yield more accurate predictions to some extent. The GNP study (Ng & Zhang, 2002) shows that improvement is gained until 7 dimensions of Euclidean space. The TPNP study (Chapter 5) showed that it is possible to improve the results in lower dimensional values such as 2 and 3. The results in the tables indicate that for our application 4 dimensions are sufficient. In most part, after 4 dimensions results are not improved, improvements are negligible or they are even degraded. Of course the MA computing is

not the only application area of distance estimation schemes. We note that a careful analysis of all potential applications should be taken into account before deciding the number of dimensions that would be used in a real system. The cost of using more dimensions may not justify the benefit that could possibly be gained.

6.6 Context-awareness in MA Computing

Context-awareness is an important characteristic for multiple mobile agent systems and more so for autonomous group of multiple mobile agent systems. This concept is subtle in classical mobile agent model since majority of the studies on mobile agents assumed that an itinerary, which is specific to a mission, could easily be obtained from a so-called directory service. Others assumed a closed network where the hosts to be visited are fixed or known in advance. For example, in a closed network, security is not much of a concern and therefore as long as the agents are within a pre-specified environment no context change occurs. Once an itinerary formed in either way, context information becomes “out of context.”

For an autonomous group of agents the first type of contextual information is the available and suitable mobile agent systems where the agents could be executed. Recall from Chapter 2 that mobile agents require their own system to be loaded and executed. This information applies also to the single mobile agent model.

As the second type of contextual information, location/proximity becomes necessary. Knowledge of some suitable system’s existence does not necessarily imply that we could use it in a specific mission. Those hosts may simply be located too far from our target environment which makes them out of context in terms of location.

The last type of contextual information is about the security requirements of the missions. It is necessary to find out the level of security provided by each host to be

considered as a part of a mission. This aspect is detailed in Section 6.7.1 where we discuss trust model and requirements.

We have mentioned two alternative architectures in Chapter 3 for the network part of our overall framework of multiple mobile agent systems. One uses an additional distributed directory component while the other uses the TPNP system. We consider the latter in this study because of its *self-formation* property. In this architecture, a directory system still exists but its function is identical to the one which is sufficient for classical single mobile agent systems. Although the TPNP is basically a network distance prediction scheme, we call it a network positioning system to distinguish it from other systems since it is capable of providing contextual information.

Step 1 in Section 5.3.1 details the discovery phase of the basic TPNP algorithm. To gather contextual information we extend both the inquiry Multicast and the Response messages by adding an application identification part. The inquiry messages include a field to identify the application that uses TPNP. The response messages consist of two parts. The first part is the same as of Chapter 5, which is related to coordinate specific information. The application identification part consists of the following:

- The application using TPNP that is running on the host,
- Parameters that are necessary for this application.

The first field uniquely identifies the multiple mobile agent or any application that could benefit from TPNP. The second field can contain sub-fields. For our application these are:

- Which mobile agent system(s) is running on this hosts,
- Type and level of security provided by the system,
- Other hosts known that have similar services provided.

6.7 Multiple Cooperating Mobile Agents Model

The overall system framework includes three models: a mission model, a trust model and a performance model. The trust and performance models are based on the mission model. Mission models for multiple autonomous groups of mobile agents have been introduced in Chapter 3. In this chapter we assume the simple master/support mission model for two autonomous mobile agents. One is the master agent with an itinerary and the second is the support agent, which needs an itinerary to be computed according to the itinerary of the master agent. Following sub-sections present the trust and performance models.

6.7.1 Trust Model

To the mobile agent security problem, there is no single solution with either non-technical trust establishment mechanisms (note that some of these mechanisms may be based on technical countermeasures) or technical countermeasures. Even combination of a few technical proposals cannot address every attack possible against agents. Therefore, a combination of technical solutions with non-technical trust establishment mechanisms and relationships are necessary.

Trust establishment mechanisms can be summarized as follows:

- Hosts employ and advertise trusted hardware as explained in Section 2.4.2
- Social or organizational trust which includes trust by reputation as explained in Section 2.4.2.
- Hosts employ some agreed-upon security evaluation criteria and advertise this.
- Trust establishment system: hosts advertise mobile agent transactions served by themselves via third parties that evaluate and certify claimed (number of) transactions similar to eBay “ratings on sellers.”

However, level of trust established via the above mechanisms differs, depending on the applications and end-user. The assumption we make here is that trusted hosts exist in the assumed environment; usually these are not the ones that offer services sought by agents.

The autonomous group of multiple mobile agents scheme is complementary to the trust establishment mechanisms above. Therefore we define three levels of security in the system based on the trust establishment mechanisms above:

Untrusted master itinerary, untrusted support itinerary. This is the most difficult situation to address. Agents do not have anything other than mechanisms they employ to check each other's computations since no trust establishment can be recognized by any of the mechanisms above.

Untrusted master itinerary, trusted support itinerary. This case use hosts for the support agents, that have one or more of the trust establishment mechanisms above in place.

Trusted master itinerary, trusted support itinerary. This case may not seem to need any additional security measures or mechanisms. However, as we stated above, the techniques we employ are complementary to the trust establishment mechanisms. Depending on the application and user requirements, those mechanisms alone may not necessarily be sufficient to put trust into the system.

The security in the mission model given in the previous section is centered on the communication or more precisely on the messages exchanged between the agents. Therefore, we define the trust levels in the system around these messages. We define mainly three categories of trust levels as follows.

Number of messages. Indicates how many messages are to be exchanged between two migrations; in other words when the agents are still executing on the same hosts. One can argue that agents may report to each other in a single message, however, an application may require a sophisticated checking mechanism or require agents to execute a challenge-response protocol. We will call the levels of trust in this category as *primary trust level*s. Each level is represented with an integer as the number of messages exchanged. Thus the lowest level is 1.

Frequency of communication. The agents may not need to communicate on every host they visit. For example, it may be sufficient for the agents of some less security demanding application to exchange messages on every other host visited. We define three levels as *A*, *B*, and *C*, *A* being the highest level. In level *A*, agents are required to communicate on each host, level *B* requires communication on every other host and *C* means every third host will be sufficient. We will call the levels of trust in this category as *secondary trust level*s.

Size of messages. The larger the message the better the security since message size is a direct indication of how much intermediate computation result and/or state information are exchanged among the agents.

In Section 6.8.3 we will analyze the communication overhead of the first two categories of trust levels. The notation, i.e., *A3* means that the primary trust level is *3* and the secondary trust level is *A*.

6.7.2 Performance Model

Strasser and Schwehm (1997) introduced a performance model for mobile agents. Although aimed to be a generic model, their tacit assumption was that mobile agents examine and return vast amount of data to their home site, which is valid for information

search and retrieval applications. Their work uses a selectivity factor μ, which is defined as the mobile agent's ability to refine the search results to be returned to the user. This factor is the key point in their model and suitable for comparisons between client-server and mobile agent computing.

Our aim however is not to compare these two distributed computation models. Our main concern is to investigate the communication overhead introduced by multiple agents for a single mission in addition to the time it takes for a single agent to perform the same mission. Therefore we use a flexible, simple, and basic performance model which can easily be extended to more sophisticated models required for more specific applications. So when we remove the selectivity factor given by Strasser and Schwehm (1997), we are left with the basic communication time equation:

Communication Execution Time (CET) = Propagation Delay + Transmission Delay

The propagation delays are predicted and provided by TPNP. It is observed for every migration and message communication regardless of their load or size, respectively.

Transmission delays are related to the transmission rate of the link used and size of the messages exchanged. Unfortunately there is no known way of measuring transmission rates of the links in a large scale internetwork as opposed to the propagation delay predictions provided by TPNP and other similar systems. In addition, transmission rates can always be improved. However, within propagation delay we are physically limited with the speed of light. So, any optimizations in terms of propagation delay will be useful and more meaningful towards future networks that might possibly have more and more transmission rates.

So for mobile agents we have

Mission Communication Time (MCT) = Agent Migration CET + Inter-Agent CET

In the single agent model only the agent migration CET is considered.

6.8 Application of TPNP to Multiple Cooperating MAs

In Section 6.5 we presented the results of performance analysis of applying TPNP and TSP to the classical single mobile agent model. In this section we consider multiple cooperating mobile agents by taking the communication between them into account. For simplicity we are using two agents, one master and one support agent. This problem is significantly more complex than the single agent model. Here we are given two sets of hosts. The first one contains the itinerary of the master agent and the second one is the available hosts for the support agent. There are many possibilities for deciding the tours of each set. One possibility is the pair wise matching, which finds the best possible tour for the MA and then to decide the individual SA host for each host in the master itinerary. This should give us the best performance in terms of inter-agent communication time since we are capable of finding the best SA host for each host in the master itinerary. However, this is not a practical solution for several reasons. First, the available SA hosts might be limited. Second, this complicates the itinerary of the SA and therefore the application. Third, we would need to take into account the time for the support agent's migrations. The last one is that the support agent's tour may need to repeat some hosts meaning that the support agent would have to travel among the hosts back and forth to be near the MA hosts, not to mention the increased network load.

For the reasons above, we are taking a clustering approach that is more manageable and secure. We do this by clustering the hosts in the master itinerary and select the best possible SA host for each cluster. The approach is more manageable because the number

of hosts reduces to the number of clusters. It is more secure since it requires fewer number of SA migrations. Therefore, the number of clusters is equal to the number of hosts in the support itinerary. Once the master agent completes visiting the hosts in one cluster, it migrates to the first host in the next cluster. At the same time the support agent migrates to the next host in its own itinerary, which is the closest host to the new cluster. We call this problem Multiple Autonomous Traveling Agents Problem (MATAP).

Figure 6-3 illustrates an instance of the problem. The double-line arrows indicate the tour of the MA, while the single-line arrows show the tour of the SA. The clusters created out of the MA tour are shown as circles. For clarity, only the SA hosts that are selected from all available ones are shown. The dashed lines indicate the distance between the hosts in the cluster of the MA tour and the selected SA host for that cluster. The small rectangles represent the hosts in the master itinerary while the small circles represent the hosts in the support itinerary. The labels in the figure will be clear in the following subsection.

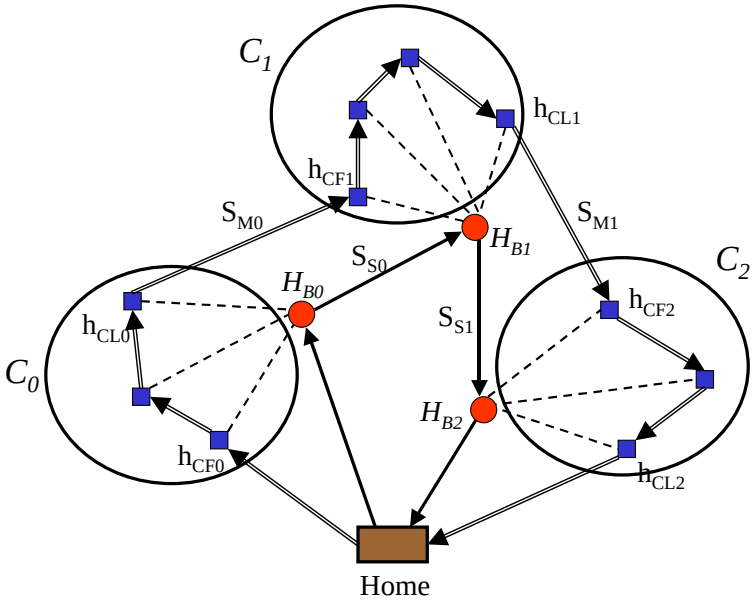


Figure 6-3. Illustration of MATAP

An alternative approach could be to start from the set of available SA hosts, pick the subset of best hosts in the master itinerary for each chosen host for the support itinerary. However the selection of best SA hosts seems like a difficult problem. The only way of doing this is a brute-force computation of distances between each SA host and all the hosts in the master itinerary. Therefore this alternative does not seem like a viable solution.

So, the MATAP is to minimize both the master agent's tour and the inter-agent communication between the master and the support agents. The formalization of the problem is given in the next sub-section.

6.8.1 Problem Formalization

We define the following parameters:

$H_M = \{h_i \mid 0 \le i < n_m\}$; set of hosts in the master itinerary

$C = \{C_i \mid 0 \le i < n_c\}$; set of clusters

$C_i = \{h_{cij} \mid 0 \le j < n_{hi}\}$; set of hosts in the cluster

$C_i \cap C_j = \phi$ for all $i, j \quad i \neq j$

therefore

$n_c = \mid C \mid$

$n_{hi} = \mid C_i \mid$

$n_m = \mid H_M \mid$

$H_S = \{h_{si} \mid 0 \le i < n_s\}$; set of available hosts for the support itinerary

$H_B = \{h_{bi} \mid 0 \le i < n_c\}$; set of support agent hosts selected from H_S

$H_B \subset H_S$ and $\mid H_B \mid = n_c$

$h_{Bi} \leftrightarrow C_i$; h_{Bi} is the support agent host corresponds to/selected for C_i

T_H : TSP tour of H_M

T_C : corresponding TSP tour of C (not used, helps understand the T_S)

T_S : corresponding TSP tour of H_B

Given the parameters we can define the MATAP as follows:

The problem consists of

- finding the optimal TSP tour T_H
- finding the optimal TSP tour T_S
- selecting $H_B \subset H_S$
  where the sum of the distances between h_{Bi} and h_{cij} is minimum.

The corresponding minimization problem is to minimize the expression

$$\sum_{i=0}^{n_m-1} \Delta(h_i, h_{i+1}) + \sum_{i=0}^{n_c-1} \sum_{j=0}^{n_{hi}-1} \Delta(h_{bi}, h_{cij}) + \sum_{i=0}^{n_c-1} \Delta(h_{bi}, h_{bi+1})$$

where $\Delta(a,b)$ is the function that defines the distance between two hosts a and b in Euclidean metric.

The first term in the expression denotes the TSP tour of T_H, the last one is for TSP tour of T_S, and in between is the inter-agent distances of selected support agent hosts and the corresponding hosts in the clusters.

However, the expression above is not exactly what we are to deal with since it denotes the whole distance (delay) of a mission. The first and the third terms actually overlap in time, and the former dominates the latter and is the one which is important. So, considering the mission execution time we need to refine the expression by replacing the last term with the total difference between the inter-cluster migration times of MA and the corresponding migration times of SA. We first need to define two more parameters:

$H_{CF} = \{h_{CFi} \mid 0 \le i < n_c\}$; set of hosts in the clusters to be visited by the MA *First* in the cluster

$H_{CL} = \{h_{CLi} \mid 0 \le i < n_c\}$; set of hosts in the clusters to be visited by the MA *Last* in the cluster

where $h_{CLi}, h_{CFi} \in C_i$ and $H_{CF}, H_{CL} \subset H_M$

and we define

$S_{Mi} = \Delta(h_{CLi}, h_{CFi+1})$
$S_{Si} = \Delta(h_{bi}, h_{bi+1})$

where $0 \le i < n_c - 1$

so the SA wait time is given as

$$SA_{wi} = \begin{cases} S_{Si} - S_{Mi}\ ; & S_{Si} > S_{Mi} \\ 0\ ; & otherwise \end{cases} \quad \text{where } 0 \le i < n_c - 1$$

So, when the last term in the above expression replaced the refined MATAP becomes the minimization of the expression

$$\sum_{i=0}^{n_m - 1} \Delta(h_i, h_{i+1}) + \sum_{i=0}^{n_c - 1} \sum_{j=0}^{n_{hi} - 1} \Delta(h_{bi}, h_{c\,ij}) + \sum_{i=0}^{n_c - 2} SA_{wi} \ .$$

The MATAP is NP-hard since a deterministic polynomial time solution to this problem would also solve the clustering problem and TSP.

6.8.2 Heuristics for MATAP

We consider two heuristics for MATAP. The idea of the first algorithm follows the FRP heuristic given in Section 6.4.2. FRP is based on the patching heuristic (Lawler et al., 1985). They use the assignment problem heuristic to start with and create clusters/tours for smaller parts of the given input set of nodes. Then they use the patching heuristic to patch the tours together to obtain a complete tour of all nodes. It is possible to obtain good tours at the expense of computation time (Bentley, 1992). So, FRP provides a compromise between performance and efficiency. Using FRP it is easy to obtain clusters while building the tour. This is due to the fact that, FRP uses a K-d tree and its buckets as clusters and connects the clusters of hosts in buckets to obtain a complete tour. So, the idea is to use the clusters in FRP to provide a sub-optimal solution to the TSP, and to compute fragments in the master agent tour and use them to figure out the nearest support agent locations. As presented in Section 6.5.3, FRP's performance is poor compared to the other heuristics. Therefore we also introduce another simple heuristic with much better performance and efficiency as follows.

We start with an NN tour, which has the 2-Opt property (note that a 2-Opted RR would also work), and form the clusters by considering the longest fragments between each two consecutive nodes in the tour. For example, if we need 5 clusters, we find the longest 4 such fragments in the tour and consider the nodes between each two consecutive fragments as one cluster. If we refer to Figure 6-3, the S_{M0} and S_{M1} are chosen as the longest fragments. In our implementation the only constraint is that the longest fragments chosen cannot be adjacent. This requires linear time on the number of nodes. Other constraints may be introduced; for example number of nodes in clusters can be made more balanced in polynomial time. The advantage of the heuristic is that we

already have a near-optimal tour provided by 2-Opted NN for the master agent. By creating clusters out of this tour we just need to figure out the nearest hosts in the set of available support agent hosts, to these clusters. One way of doing this is to build a K-d tree of available SA hosts, compute the averages of coordinates in each cluster and then perform a nearest neighbor search on the tree with the input of cluster averages.

Heuristic arguments can be summarized as follows:

- We need to compute a near optimal tour for the master agent anyway, and efficient implementations of algorithms are already in place.
- Because we form clusters as fragments in the master TSP tour, intra-cluster distances are already small.
- Because the inter-cluster fragments are the longest ones as possible (due to constraints) it can be expected that the possibility of SA's having a longer path to migrate between trusted hosts while the MA migrates to leave a cluster and enter the next one, is low. Therefore wait times for MA due to SA migrations can be ignored for more efficient computation.
- Because we form clusters as TSP tour fragments and then choose the nearest hosts to these clusters from the host set of support agent, it is highly likely that the tour of the support agent will be near-optimal under the other constraints the algorithm observes.

6.8.3 Experimental Evaluation

We conducted simulation experiments for both of the heuristics from the previous subsection. In this subsection the details of our simulation experiments are given.

6.8.3.1 Simulation parameters

The simulation parameters defined in Section 6.4.1 for single agent computations are valid here. In addition to them new parameters are defined as given in Table 6-4.

Table 6-4. Simulation parameters

Parameter	Default value
Number of hosts in the master itinerary	100
Transmission rate of all the links in the network	1 MB/s
Primary security level	2
Secondary security level	A
Number of SA hosts	7
Number of available SA hosts	20
Size of an agent	40 KB
Size of inter-agent messages	1 KB
Size of messages in client-server communication	1 KB

6.8.3.2 Simulation strategy

We follow generally the same strategy in Section 6.4.2 of single agent model. So, we choose the hosts in the master and support itineraries randomly from all the hosts in the network. The same is true for the home of the agents. These three sets of hosts are distinct. To ensure repeatability, for the random tour computations, we pick 100 different sets of hosts and report their averages. For all the experiments, we pick 3 different host sets and report the averages for the same reason. That is, for each of the three runs, we choose a set for MA itinerary, SA itinerary and the home.

In this study, our main goal is to measure the mission communication time, which is the time observed by the end-user. So, we look at the problem from the application point of view rather than the network load overhead that is introduced by multiple agents.

6.8.3.3 Results and analysis

Figure 6-4 shows the mission communication times for each selected mission strategy. The Single-Random and Single-Computed represent single agent missions. The former uses a random tour of an itinerary while the latter has a computed near-optimal tour of the same itinerary. The Client-Server line shows the time necessary to exchange two messages between each host in the itinerary and the home. Our aim is not to compare mobile agent computing with client-server computing which has been done before by

many researchers, since the comparison yields many different conclusions depending on the application and the direct effect of this to the size of messages transferred between the two hosts. Our aim however is to give an idea about the size of the missions in our simulation environment and to be able to associate the values with real-life cases by using the client-server paradigm as the connection between the simulation environment and the actual Internet environment.

The rest of the lines show the results of multiple-agent missions. In this category, the values in the graph includes both the MA tour and the inter-agent communication times. Both-Random means that both the tour of the MA and the selection of SA hosts are performed randomly. This is clearly the situation where we expect the worst results. FRP-Random and FRP-Selected both correspond to the MA tours computed using FRP, which correspond to the first heuristic of Section 6.8.2; the former picks the SA hosts randomly, while the latter selects the nearest SA hosts. The selection process uses a K-d tree where we build a tree of all available SA hosts and then use the nearest neighbor search routine to find the closest host to the given cluster. The input to the procedure is the simple average of the coordinate values of all the hosts in the cluster. The output of the procedure is the host that is closest to the given cluster average which represents all the nodes in the cluster.

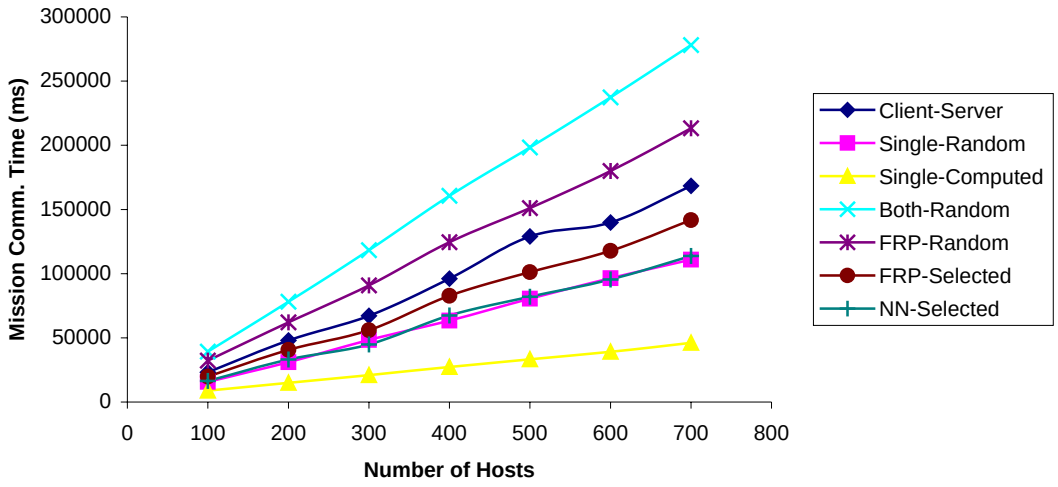


Figure 6-4. Mission communication time across problem size

NN-Selected uses the NN heuristic for the MA tour and selects the SA hosts with the same procedure as explained above. This corresponds to the second heuristic introduced in Section 6.8.2. Since the result of random selection behavior is already observed with the FRP heuristic above, we do not include the NN-Random strategy.

It is interesting to see in the graph that the Single-Random and NN-Selected values are so close to each other that they cross each other at several points and overprint at some segments. The Single-Random represents the case where we have a single agent for the mission with no optimizations; that is the current situation in which we do not have TPNP and do not use TSP at all. The latter represents two agents as well as the communication between them, which uses two messages at each MA host as the default security level values. The conclusion is that the inter-agent communication overhead is compensated by the optimization of the tour length.

Figure 6-5 displays a bar chart of MCTs versus categories of combined security levels defined in Section 5.2. The bar labeled as Average corresponds to Both-Random in the previous figure. As explained before security levels are defined as the number of

messages exchanged between the agents. So, migration CETs are the same for each heuristic category in the figure. What changes is the number of messages (1,2, 3, or 4) exchanged and at what frequency (A, B, or C) the communication takes place. Some categories are almost the same in terms of the MCTs as can easily be observed from the figure. For example categories A2 and B4 have very similar MCT values since the total number of messages exchanged is the same. They are not exactly the same because the inter-host distances are not exactly the same. However, the trust that we can place on each category is different and may be interpreted as completely different depending on the application. This interpretation is subject to further research.

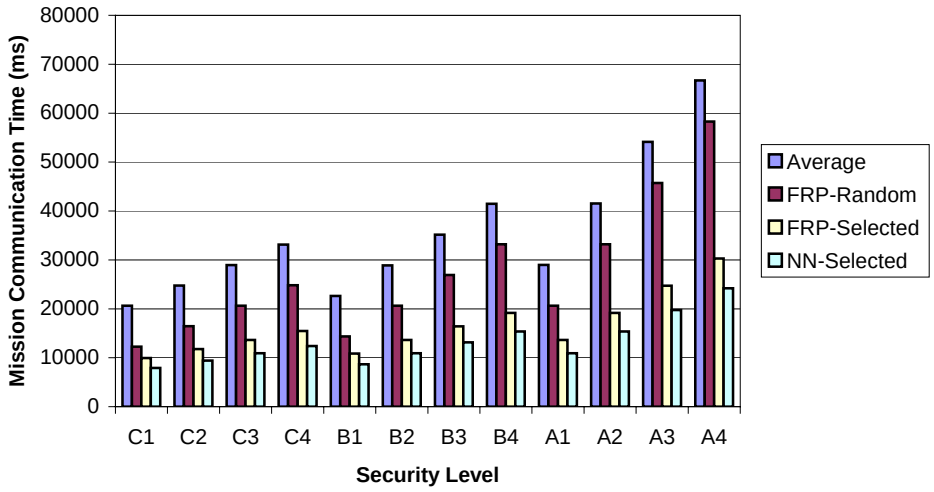


Figure 6-5. Mission communication time across security level category

In Figure 6-6 we show the results of another experiment to illustrate whether we can by chance create a near-optimal tour for the master itinerary and select good SA locations so that the results would be close to the computed configurations of the tour and the hosts. For this purpose we plotted for each category, average, minimum and maximum values of the Both-Random configuration. As it is seen in the figure, NN-Selected configuration outperforms even the minimum values of Both-Random category

out of 100 different random configurations. We note here that the results are based on experimental evaluation and statistical analysis of the heuristic is subject to future research.

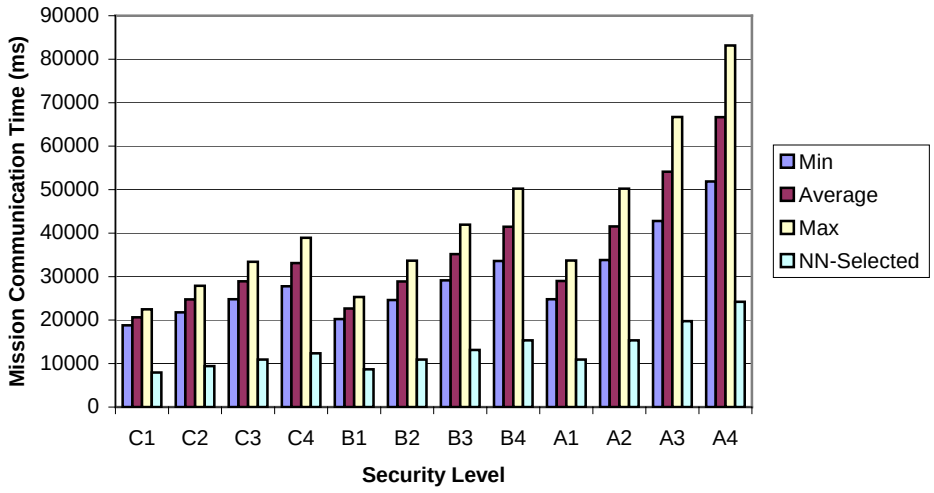


Figure 6-6. Mission communication time across random selection distribution over security levels

Since there is no known way of predicting transmission rates of links in a large scale internetwork, in our experiments we have used a fixed value of 1 MB/s for all the paths in the network. Figure 6-7 displays how the configurations used in the experiments behave as the transmission rates increase. In the client-server case, the difference between points is so small that the line appears almost to be straight. This is due to our assumption of 1 KB message sizes. The two interesting configurations NN-Selected and Single-Random here shows another interesting behavior. Single-Random performs slightly better than the NN-Selected when the transmission rates increases. The reason is that there is no inter-agent communication in the Single-Random case meaning that there is no message exchange of 1 KB size. The MCT is consist only of 40 KB agent migrations, which explains why it is more prone to transmission rate increases. The summary of the figure is

that transmission rate increase in the network has limited effect on the overall application performance, even with larger mobile agent sizes. We are still limited with the propagation delays and this validates our results which depend on network propagation delay estimations.

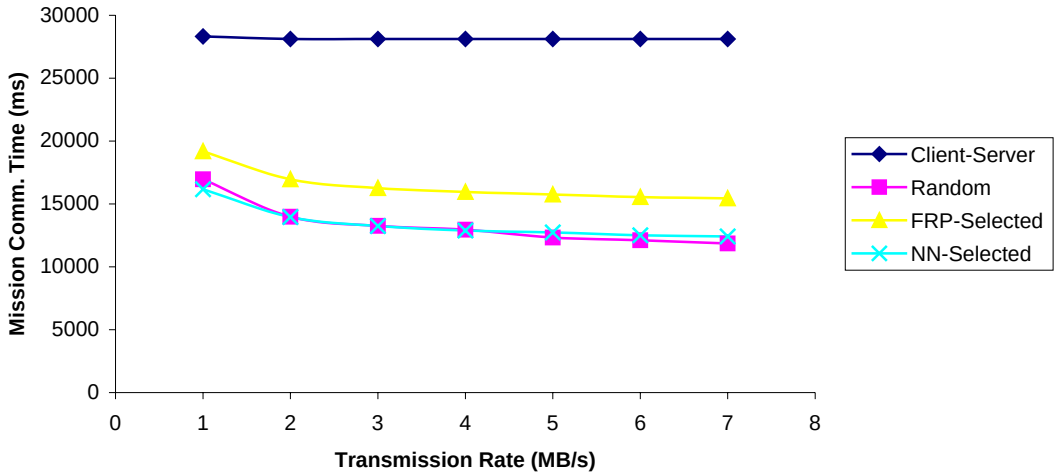


Figure 6-7. Effect of transmission rates on mission communication time

As mentioned before, we create different configurations of hosts for the home, the master itinerary and SA hosts from which the support agent itinerary is created. These hosts are selected in a pseudo-random fashion out of all the hosts in the topology. In this environment we wanted to see the effect of creating more available support agent hosts to choose from. While the default value is 20 as given in Table 6-1, Figure 6-8 tells us that the value we have chosen randomly was actually pretty good and no significant improvements are made with increasing numbers of potential SA hosts. The conclusion is that it is more valuable to have these hosts well distributed over the network rather than having more of them. This result tells us that context-awareness of TPNP is rather powerful characteristic than we have initially anticipated.

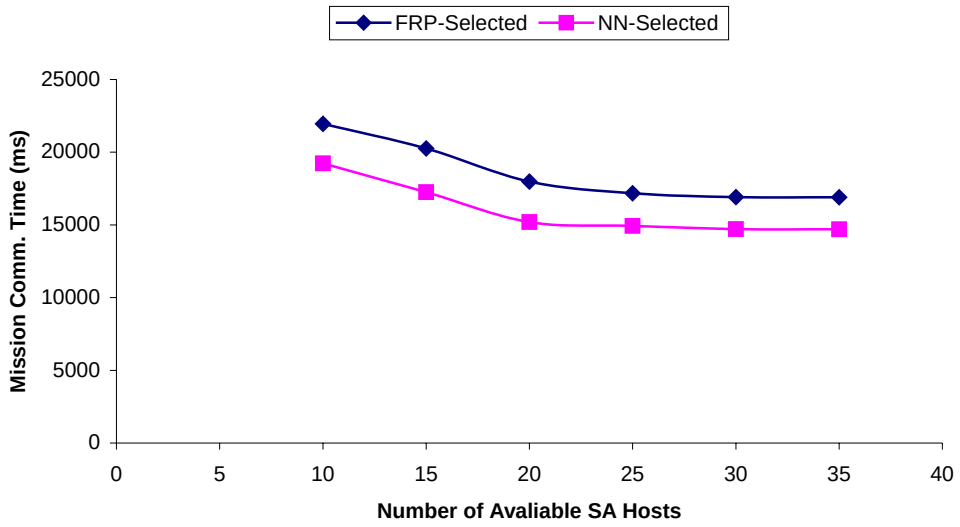


Figure 6-8. Heuristic performance with the available SA hosts in the network

A similar graph is presented in Figure 6-9. This time, we want to see the effect of the number of clusters, which is equal to the number of SA hosts used. The default value is 7 as given in Table 6-1. The increase clearly has a positive performance effect. But this increase diminishes at point 10. Although it is always possible to increase the performance by using more clusters, as discussed before this has a negative effect on the complexity of the system, security, and the difficulty due to availability of the SA hosts in the environment.

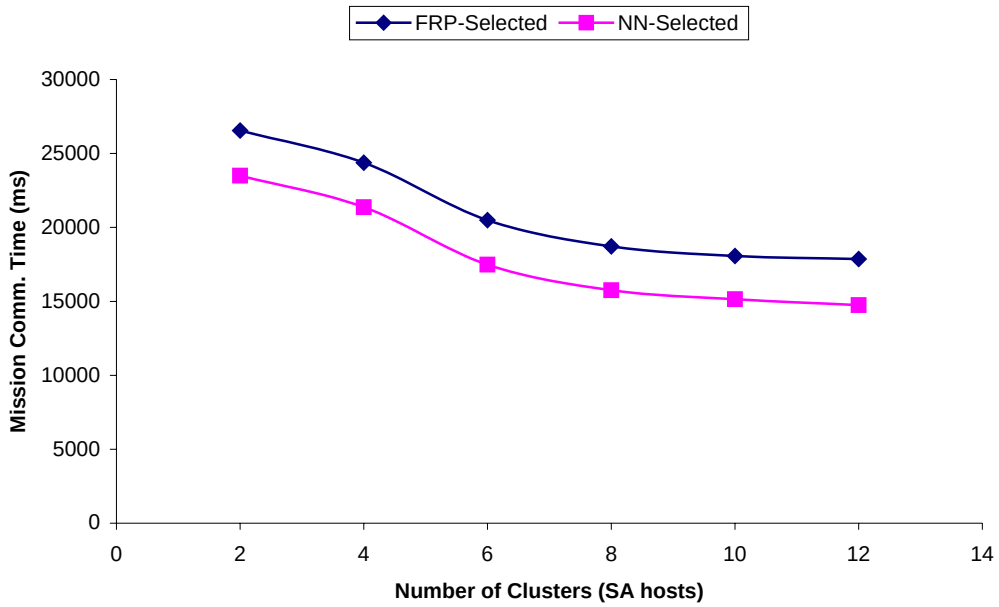


Figure 6-9. Heuristic performance with the number of clusters

6.9 Related Work

Theilmann and Rothermel (2000a) proposed an approach to use Internet distance information for a specific application of mobile agents in information retrieval/filtering. The approach is called the dissemination of mobile agents. Their motivation is the assumption that every host on the Internet is a potential data warehouse but that not all of them can be assumed to employ a mobile agent system. Thus, the necessity of finding nearby hosts to interested warehouses that run a mobile agent system arises. The dissemination approach uses an underlying distance prediction system which is called the Internet Distance Maps (Theilmann & Rothermel, 2000b). The dissemination approach requires mobile agents to be forwarded to hosts participated in the system. Once a host receives a mobile agent, it decides whether forwarding the agent to another host would be beneficial in terms of communication costs. Thus, the agents are forwarded from host to host until they reach a host, which is closest to the intended warehouse. However it is not clear whether the underlying distance maps system is intended for this specific

application only. If this is not the case, mobile agents are to be forwarded to ordinary hosts which do not have the mobile agent system to run them. So, these hosts are intermediaries which used only as routers. This introduces another problem to the already difficult problem of mobile agent protection. If the approach, on the other hand, is intended for mobile agent systems only, then it clearly fails to be a general-purpose, global system. Moreover, the dissemination approach do not address the security problems either the ones introduced by it, or the other general ones. As we already mentioned above, the approach addresses only a specific application area.

To the best of our knowledge, the only study that considered the TSP for mobile agent applications is Brewington et al. (1999). Their target application is again information retrieval. They defined the Traveling Agent Problem, which adds the classical TSP, the probabilities of success for a mobile agent to complete a task, which is to find some specific information on a host. Once the agent finds the information needed it returns home immediately. Since the problem they defined is NP-complete, they simplified it by assuming constant latencies on the network. This makes the problem much simpler so that a polynomial time solution could be given. However, as we have shown in this chapter, there are numerous heuristics for TSP that could be used for mobile agent applications with success in terms of both performance and efficiency.

The literature is rich with studies for quantitative analysis of mobile agent systems and applications. Strasser and Schwehm (1997) proposed a performance model for mobile agent systems for information retrieval applications. Johansen (1998) in addition to performance analysis of mobile agent systems shared their experiences with real mobile agent systems they had been using for years. The work on mobile agent

performance evaluation can be categorized as modeling, quantitative analysis of communication and execution times as well as scalability. A summary survey for both categories is given by Gray et al. (2001). None of these studies however, considered network-awareness.

6.10 Conclusion and Future Work

In this chapter we have shown how the knowledge of inter-host distances could be useful for both classical mobile agent systems and multiple mobile agent systems using the well-known TSP heuristics. We have also shown that the answers to the questions we asked at the beginning of the chapter are both “yes.” In multiple mobile agent systems we have shown that performance overhead of providing protection to mobile agents using the concept of information dispersal for security can be alleviated with negligible off-line computation overhead. For this purpose we have defined categories of security levels and analyzed the communication overhead introduced. We have also clearly presented the network- and context-awareness concepts in mobile agent computing. Overall the goal has been justified.

Both TSP and multiple mobile agents computing are quite broad research areas. We believe that we have introduced an interesting research field with many open questions. Within the scope of this chapter, we could provide only the tip of the iceberg.

As we pointed out at the beginning of the chapter, we did not target any specific application area of mobile agents. Every application has its own requirements and characteristics and these need to be analyzed. For example information retrieval applications should consider the size of the relevant information and therefore the size of the mobile agents. In software distribution, the number of the sites to be visited and the number of agents that need to be employed should be taken into account. This application

may lend itself to a variant of TSP, which is known as vehicle routing problem or multiple salesman problem.

In addition to application execution time, network load of multiple mobile agent systems need to be analyzed.

We have considered only the master/support agent model with two agents in this study. The other models can also be studied with varying number of mobile agents. For example, the master/support agent model admits more than one master agent. How many agents are to be employed for a specific mission of an application and how to route them efficiently by also observing the security requirements are all open questions.

A more comprehensive study should also take into account the time necessary to obtain the network/context information.

It is possible to improve the TSP implementation performance without much efficiency degradation. For example, in addition to the 2-Opt heuristic, 2H-Opt and even 3-Opt heuristics can be implemented.

CHAPTER 7
SUMMARY AND FUTURE DIRECTIONS

This research proposes an architectural model (MMAS) for multiple mobile agent systems as a future computation paradigm with a special focus on the support for security and mobility. Chapter 2 provides a comprehensive background on mobile agent computing and security, and justifies the use of mobile agent technologies. Chapter 3 extends the mobile agent concept to include multiple agent collaboration for enhancing security, which is considered a major hindrance toward wider acceptance of mobile agent computing.

The major technical contributions in the research are categorized in three complementary system supports for MMASs:

- Information dispersal for agent protection – The need for "computing with secrets in public" can be supported by a non-cryptographic scheme using information dispersal, which enhances security through "protection as a whole". Chapter 4 demonstrates this concept with the application of distributed digital signing, which is an essential function for e-commerce systems. The results show collaborating mobile agents can effectively implement distributed digital signing.

- Positioning service for mobile agents – The mobility of mobile agents require an efficient network positioning service. Chapter 5 proposes a peer-to-peer coordinate-based network positioning system for computing the distances between network nodes to facilitate the migration of mobile agents. Simulation results show that the architecture could provide more accurate predictions of the distances than previously proposed approaches, and the system is also more scalable. An additional advantage of the MMAS is that the architecture addresses not only network-awareness but also context-awareness.

- Itinerary management for collaborating agents – The itinerary of a mobile agent(s) is a key component. Traveling with respect to an itinerary list is akin to the traveling salesperson problem, except in the context of multiple collaborating mobile agents, the problem becomes much more complex. Chapter 6 addresses the mission optimization of multiple inter-dependent itineraries, and proposes

heuristics for itinerary traversal based on the positioning service in Chapter 5. Through simulation experiments, it is shown that performance overhead of providing protection to mobile agents in MMAS could be reduced to an acceptable level and is proven feasible.

Likewise, the future directions of the research can be grouped in three major sub-areas:

- Chapter 4 addresses a difficult problem of digital signing with mobile agents for e-commerce applications. However, the responsibility of the security component in the proposed architecture is much boarder. Application of information dispersal for security to general data, code and state information carried by agents remains open. There are numerous approaches to solving the general agent protection problem. Adapting and automating some of the promising solutions in the MMAS is necessary, in particular, how detection mechanisms can be adapted as prevention mechanisms in the architecture.

- The network positioning service discussed in Chapter 5 is a research area of its own. The architecture we proposed provides distance information in terms of delay measurements. It would be useful to include transmission rates and hop counts. Orthogonal to this direction, the other open question is how to achieve better prediction accuracy in these systems. This question is more application-specific in mobile agent computing. We could improve the system with more contextual information in MMASs.

- The mission optimizer component of the MMAS architecture opens up a new and interesting field of research. Chapter 6 integrates network distance estimation and TSP into mobile agent computing, and shows the potential benefits from a broad perspective. Different mobile agent applications will require different types of missions. New mission models necessitate extensions or variants of TSP heuristics. An interesting example is to determine the applicability of multiple salesman or vehicle routing problems to mobile agent computing in general and MMAS in particular.

Furthermore, the Directory system of the proposed architecture was not addressed in the scope of this study and is subject to future research of its own. In traditional mobile agent computing, this component refers usually to a centralized database system, where agents can query and locate the hosts of interest for their missions. In MMASs, the directory component should also be responsible for providing context information, in cooperation with a network positioning system that is capable of providing the

infrastructure for obtaining the information. Therefore, we presume that such a system needs to be a full-fledged, scalable peer-to-peer distributed system in order to collect information from hosts and provide the same information to interested hosts efficiently.

REFERENCES

Baldi, M., Gai, S., & Picco, G. P. (1998). Exploiting code mobility in decentralized and flexible network management, *Mobile Agents, Lecture Notes in Computer Science Vol. 1219*, pp. 13-26, Springer-Verlag.

Baldi, M. & Picco, G. P. (1998). Evaluating the tradeoffs of mobile code design paradigms in network management applications**,** *Proceedings of the 20th International Conference on Software Engineering*, pp. 146 –155, Kyoto, Japan: IEEE Press.

Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., & Yang, K. (2001). On the (im)possibility of obfuscating programs, In J. Kilian (Ed.) *Proceedings of the 21st Annual International Conference, Lecture Notes in Computer Science Vol. 2139*, pp. 1-18, Springer-Verlag.

Bellare, M. & Rogaway, P. (1996) The exact security of digital signatures: how to sign with RSA and Rabin, In U. Maurer (Ed.), *Advances in Cryptology - Eurocrypt 96 Proceedings, Lecture Notes in Computer Science Vol. 1070*, pp. 399-416, Springer-Verlag.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching, *Communications of the ACM, 18(9)*, pp. 509-517.

Bentley, J. L. (1990a). Experiments on traveling salesman heuristics, *Proceedings of First Symposium on Discrete Algorithms*, pp. 91-99, Philedelphia, PA: SIAM Press.

Bentley, J. L. (1990b). K-d trees for semidynamic point sets, *Proceedings of Sixth Annual ACM Symposium on Computational Geometry*, pp. 187-197, New York, NY: ACM Press.

Bentley, J. L. (1992). Fast algorithms for geometric traveling salesman problems, *ORSA Journal on Computing*, 4(4), pp. 387-411.

Berkowitz, S., Guttman, J. D., & Swarup V. (1998). Authentication for mobile agents, *Mobile Agents and Security, Lecture Notes in Computer Science Vol. 1419*, Springer-Verlag, pp.114-136.

Bettini, L., Nicola, R. D., & Loreti, M. (2002). Software update via mobile agent based programming, *Proceedings of the 2002 ACM Symposium on Applied Computing*, pp. 32-36, Madrid, Spain: ACM Press.

Boneh, D., & Franklin, M. (1997). Efficient generation of shared RSA keys, *Lecture Notes in Computer Science Vol. 1233*, Springer-Verlag, pp. 425-439.

Borselius, N., Mitchell, J. C., & Wilson, A. (2001a). On mobile agent based transactions in moderately hostile environments, *Proceedings of IFIP First Annual Working Conference on Network Security*, pp, 173-186, Deventer, The Netherlands: Kluwer, B.V.

Borselius, N., Mitchell, J. C., & Wilson, A. (2001b). Undetachable threshold signatures, *Proceedings of the 8th IMA International Conference, Lecture Notes in Computer Science Vol. 2260*, pp. 239-244, Springer-Verlag.

Boyd, C. (1988). Some applications of multiple key ciphers, *Advances in Cryptology, EUROCRYPT'88 Proceeding, Lecture Notes in Computer Science Vol. 330*, pp. 455-467, Springer-Verlag.

Boyd, C., (1989). Digital multisignatures, *Cryptography and Coding*, pp. 241-246, London, UK: Oxford University Press.

Bradshaw, J. M. (Ed.), (1997). *Software Agents*, Cambridge, MA: AAAI Press/The MIT Press.

Brazier, F. M. T., Overeinder, B. J., Steen, M. V., & Wijngaards, N. J. E. (2002). Agent factory: generative migration of mobile agents in heterogeneous environments, *Proceedings of the 2002 ACM Symposium on Applied Computing*, pp. 101-106, Madrid, Spain: ACM Press.

Breugst, M. & Magedanz, T. (1998). Mobile agents - enabling technology for active intelligent network implementation, *IEEE Network, Special Issue on Active and Controllable Networks*, 12(3), pp. 53 – 60.

Brewington, B., Gray, R., Moizumi, K., Kotz, D., Cybenko G., & Rus, D. (1999). Mobile agents for distributed information retrieval, Chapter 15 in M. Klusch (Ed.) *Intelligent Information Agents – Agent-Based Information Discovery and Management on the Internet*, Berlin: Springer-Verlag.

Busch, C., Roth, V., & Meister R. (1998). Perspectives on electronic commerce with mobile agents, *Proceedings of 11th Amaldi Conference on Problems of Global Security*. pp. 89-101, Moscow, Russia: Russian Academy of Sciences.

Busse, I., Covaci, S., & Leichsenring A. (1999). Autonomy and decentralization in active networks: a case study for mobile agents, in S. Covaci (Ed.) *Proceedings of First International Conference on Active Networks, Lecture Notes in Computer Science Vol. 1653*, pp. 167-179, Springer-Verlag.

Chess, D., Grosof, B., Harrison, C., Levine, D., Parris, C., & Tsudik, G. (1995a). Itinerant agents for mobile computing, *IEEE Personal Communications*, pp. 34-49.

Chess, D., Harrison, C., & Kershenbaum, A. (1995b). Mobile agents: are they a good idea?, *IBM Research Report, RC 19887*, IBM Research Division.

Chess, D. M. (1998). Security issues in mobile code systems, *Mobile Agents and Security, Lecture Notes in Computer Science Vol. 1419*, pp. 1-14.

Chow, R. & Johnson, T, (1997). *Distributed Operating Systems and Algorithms*, Reading, MA: Addison-Wesley.

Claessens, J., Preneel, B., & Vandewalle, J. (2003). (How) Can mobile agents do secure electronic transactions on untrusted hosts? A survey of the security issues and the current solutions, *ACM Transactions on Internet Technology*, 3(1), pp. 28-48.

Corradi, A., Montanari, R., & Stefanelli, C. (1999). Security issues in mobile agent technology, F*uture Trends of Distributed Computing Systems, Proceedings of the 7th IEEE Workshop on FTDCS'99*, pp. 3-8, Cape Town, South Africa: IEEE Press.

Desmedt, Y. (1997). Some recent research aspects of threshold cryptography, *Lecture Notes in Computer Science Vol. 1396*, pp.158-173, Springer-Verlag.

d'Inverno, M. & Luck, M. (2001). *Understanding Agent Systems*, New York: Springer-Verlag.

Dyer, D. (1997). Java decompilers compared, http://www.javaworld.com/javaworld/jw-07-1997/jw-07-decompilers_p.html, July 1997, retrieved September 2001.

ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory, IT-31* (4), pp. 10-18.

Fabre, J.C., Deswarte, Y., & Randell, B. (1994). Designing secure and reliable applications using fragmentation-redundancy-scattering: an object-oriented approach, *Proceedings of Dependable Computing EDCC-1, Lecture Notes in Computer Science Vol. 852*, pp. 21-38, Springer-Verlag.

Faloutsos, M., Faloutsos, P., Faloutsos, C. (1999). On power-law relationships of the Internet topology, *Proceedings of SIGCOMM'99*, pp. 251–262, Cambridge, MA: ACM Press.

Farmer, W., Guttman, J., & Swarup, V. (1996). Security for mobile agents: authentication and state appraisal, *Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS'96)*, pp. 118-130, London, UK: Springer-Verlag.

Feigenbaum, J. & Lee, P. (1997). Trust management and proof carrying code in secure mobile code applications, *Proceedings of DARPA Workshop on Foundations for Secure Mobile Code*, Monterey, CA: Kluver Academic Publishers.

Francis, P., Jamin, S., Jin, C., Jin, Y., Raz, D., Shavitt, Y., & Zhang, L. (2002). IDMaps: a global Internet host distance estimation service, *IEEE/ACM Transactions on Networking*, 9(5), pp. 525-540.

Frankel, Y. & Desmedt, Y.G. (1992). Parallel reliable threshold multisignature, Technical Report: Department of E.E. and C.S., University of Wisconsin-Milwaukee, TR-92-04-02.

Franklin, S. & Gaesser, A. (1997). Is it an agent, or just a program? A taxonomy for autonomous agents, in J.P. Muller, M.J. Wooldridge, N.R. Jennings, (Eds.) *Intelligent Agents III, Proceedings of Third International Workshop on Agent Theories, Architectures and Languages, Lecture Notes in Artificial Intelligence Vol. 1193*, pp. 21-35, Springer-Verlag.

Fray, J.M. & Fabre, J.C. (1991). Fragmented data processing: an approach to secure and reliable processing in distributed computing systems, *Dependable Computing for Critical Applications*, pp. 323-343, Berlin: Springer-Verlag.

Gray, R., Kotz, D., Nog, S., Rus, D., & Cybenko, G. (1996). Mobile agents for mobile computing, *Technical Report, Dept. of Computer Science, Dartmouth College, PCS-TR-96-28.*

Gray ,R. S., Kotz, D., Peterson, R. A., Barton, J., Chacon , D., & Gerken, P. (2001) Mobile agent versus client/server performance: scalability in an information retrieval task, *Proceedings of the 5th International Conference on Mobile Agents, Lecture Notes in Computer Science Vol. 2240*, pp. 229-243, Springer-Verlag.

Gummadi, K. P., Saroui, S., & Gribble, S. D. (2002). King: estimating latency between arbitrary Internet end hosts, *Proceedings of the (SIGCOMM) Internet Measurement Workshop,* Marseille, France: ACM Press.

Hartman, J., Manber, U., Peterson, L., & Proebsting, T. (1996). Liquid software: a new paradigm for networked systems, Technical Report: Department of Computer Science, University of Arizona, TR 96-11.

Hohl, F. (1998a). Time limited blackbox security: protecting mobile agents from malicious hosts, *Mobile Agents and Security, Lecture Notes in Computer Science Vol. 1419*, pp. 92-113, Springer-Verlag.

Hohl, F. (1998b). A model of attacks of malicious hosts against mobile agents, *Proceedings of the 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations*, London, UK: Springer-Verlag.

Hotz, S. M. (1994). Routing information organization to support scalable interdomain routing with heterogeneous path requirements, PhD Thesis, University of Southern California.

Jamwal, V., & Iyer, S. (2003). Mobile agent based realization of a distance evaluation system, in S. Helal, Y. Oie, C. Chang, J. Murai (Eds.) *Proceedings of 2003 Symposium on Applications and the Internet*, pp. 362-369, Los Alamitos, CA: IEEE Press.

Jansen, W. A. (2001). Countermeasures for mobile agent security, *Computer Communications, Special Issue on Advanced Security Techniques for Network Protection*, Elsevier Science BV.

Jansen, W, Mell, P., Karygiannis, T., & Marks, D. (2000). Mobile agents in intrusion detection and response, *Proceedings of the 12th Annual Canadian Information Technology Security Symposium*, Ottawa, Canada.

Johansen, D. (1998). Mobile Agent Applicability, *Proceedings of Mobile Agents: Second International Workshop, MA'98, Lecture Notes in Computer Science Vol. 1477*, pp. 99-111, Springer-Verlag.

Karjoth, G., Asokan, N., & Gulcu, C. (1998). Protecting the computation results of free-roaming agents, *Proceedings of 2nd International Workshop of Mobile Agents, Lecture Notes in Computer Science Vol. 1477*, pp. 195-207, Springer-Verlag.

Karnik, N. (1998). Security in mobile agent systems, Ph.D. Dissertation, Department of Computer Science and Engineering, University of Minnesota.

Katz, J., & Wang, N. (2003). Efficiency improvements for signature schemes with tight security reductions, *Proceedings of the 10th ACM Conference on Computer and Communications Security*. pp. 155-164, New York, NY: ACM Press.

Kaufman, C., Perlman, R., & Speciner, M. (1995). *Network Security*, Englewood Cliffs, NJ: Prentice Hall.

Kiniry J., & Zimmerman D. (1997). A hands-on look at Java mobile agents, *IEEE Internet Computing*, July-August 1997, pp. 21-30.

Klusch M. (Ed.) (1999), Intelligent information agents, *Agent-Based Information Discovery and Management on the Internet*, Berlin: Springer-Verlag.

Kotzanikolaou, P., Burmester, M., & Chrissikopoulos, V. (2000). Secure transactions with agents in hostile environments, *Lecture Notes in Computer Science Vol. 1841*, pp. 289-297, Springer-Verlag.

Lange, D. B., & Oshima, M. (1998). *Programming and Deploying Java Mobile Agents with Aglets*, Reading, MA: Addison-Wesley.

Lawler, E. L., Lenstra, J. K., Kan, A. H. G. R., & Shmoys, D. B. (Eds) (1985). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Chichester: John Wiley & Sons.

Lee, P., & Necula, G. (1997). Research on proof-carrying code for mobile code security, *Proceedings of the DARPA Workshop on Foundations for Secure Mobile Code,* Monterey, CA: Kluver Academic Publishers.

Lim, H., Hou, J. C., & Choi, C-H. (2003). Constructing Internet coordinate system based on delay measurement, *Proceedings of Internet Measurement Conference,* pp. 129-142, New York, NY: ACM Press.

Malkhi, D., & Reiter, M. K. (2000). Secure execution of Java applets using a remote Playground, *IEEE Transactions on Software Engineering*, 26(12), pp. 1197-1209.

Marques, P. J., Silva, L. M., & Silva, J. G. (1999). Security mechanisms for using mobile agents in electronic commerce, *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, pp. 378-383, Washington, DC: IEEE Press.

Meadows, C. (1997). Detecting attacks on mobile agents, Proceedings of *DARPA Workshop on Foundations for Secure Mobile Code*, Monterey, CA: Kluver Academic Publishers.

Meer, H. D., Corte, A. L., Puliafito, A., & Tomarchio, O. (2000). Programmable agents for flexible QoS management in IP networks, *IEEE Journal on Selected Areas in Communication*, 18(2), pp. 145-162.

Minsky, Y., Renesse, R., Schneider, F. B., & Stoller, S. D. (1996). Cryptographic support for fault-tolerant distributed computing, Technical Report, TR96-1600, Dept. of Computer Science, Cornell University.

Murch, R. & Johnson, T. (1998). *Intelligent Software Agents*, Englewood Cliffs,NJ: Prentice Hall.

Ng, S-K. (2000). Protecting mobile agents against malicious hosts, Master's Thesis, Division of Information Engineering, The Chinese University of Hong Kong.

Ng, T. S. E. & Zhang, H. (2002). Predicting Internet network distance with coordinates-based approaches, *Proceedings of IEEE INFOCOM 2002,* Vol. 21, No. 1, pp. 170-179.

Oaks, S. (1999). *Java Security*, Sebastopol: O'Reilly and Associates.

Onbilger, O.K., Newman, R., & Chow, R. (2001). A distributed and compromise-tolerant mobile agent protection scheme, *Proceedings of International Conference on Intelligent Agents, Web Technologies and Internet Commerce,* Las Vegas, Nevada, pp. 394-400.

Oppliger, R. (1999). Security issues related to mobile code and agent-based systems, *Computer Communications,* No. 22, pp. 1165-1170, Elsevier Science BV.

Ordille, J. J. (1996). When agents roam, who can you trust?, *Proceedings of the First Conference on Emerging Technologies and Applications in Communications,* Portland, OR: IEEE Press.

Papavassiliou, S., Puliafito, A., Tomarchio, O., & Ye, J. (2002). Mobile agent-based approach for efficient network management and resource allocation: framework and applications**,** *IEEE Journal on Selected Areas in Communications*, 20(4), pp. 858 –872.

Pias, M., Crowcroft, J., Wilbur, S., Harris, T., & Bhatti, S. (2003). Lighthouses for scalable distributed location, *Proceedings of the 2nd International Workshop on P2P Systems, Lecture Notes in Computer Science Vol. 2429*, Springer-Verlag.

PKCS#1: RSA Cryptography Standard, Version 2.1, RSA Laboratories (2002).

Rabin, M. O. (1989). Efficient dispersal of information for security, load balancing, and fault tolerance, *Journal of the ACM*, 36(2), pp.335-348.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S. (2001). A scalable content-addressable network, *Proceedings of SIGCOMM 2001,* pp. 161-172, San Diego, CA: ACM Press.

Ratnasamy, S., Handley, M., Karp, R., and Shenker, S. (2002). Topologically-aware overlay construction and server selection, *Proceedings of IEEE INFOCOM 2002*, Vol. 21, No. 1, pp. 1190-1199.

Riordan, J., & Schneier, B. (1998). Environmental key generation towards clueless agents, *Mobile Agents and Security, Lecture Notes in Computer Science Vol. 1419,* pp. 15-24, Springer-Verlag.

Rivest, R.L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM*, 21(2), pp. 120-126.

Roth, V. (1998). Secure recording of itineraries through co-operating agents, *Proceedings of 4th ECOOP Workshop on Mobile Object Systems,* London, UK: Springer-Verlag.

Roth, V., Jalali, M., Hartman, R., & Roland, C. (2000). An application of mobile agents as personal assistants in electronic commerce, *Proceedings of 5th Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pp. 121-132, Manchester, UK: Springer-Verlag.

Roth, V. (2001). On the robustness of some cryptographic protocols for mobile agent protection, in G. P. Picco (Ed.) *Proceedings of 5th International Conference on Mobile Agents, Lecture Notes in Computer Science Vol. 2240,* pp. 1-14, Springer-Verlag.

Sander, T., & Tschudin, C.F. (1998). Protecting mobile agents against malicious hosts, *Mobile Agents and Security, Lecture Notes in Computer Science Vol. 1419*, pp. 44-60, Springer-Verlag.

Sandholm, T., & Huai, Q. (2000). Nomad: mobile agent system for an Internet-based auction house, *IEEE Internet Computing*, 4(2), pp. 80-86.

Schill, A., Held, A., Bohmak, W., Springer, T., & Ziegert, T. (1998). An agent based application for personalized vehicular traffic management, In K. Rothermel and F. Hohl (Eds.) *MA'98, Lecture Notes in Computer Science Vol. 1477*, pp. 99-111, Springer-Verlag.

Shamir, A. (1979). How to share a secret, *Communications of the ACM*, 22(11), pp. 612-613.

Stamos, J. W., & Gifford, D. K. (1990). Remote evaluation. *ACM Transaction on Programming Languages and Systems*, 12(4), pp. 537-564.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., & Balakrishnan, H. (2001). Chord: a scalable peer-to-peer lookup protocol for Internet applications, *Proceedings of SIGCOMM 2001*, pp. 17-31, San Diego, CA: ACM Press.

Strasser, M., Schwehm, M. (1997). A performance model for mobile agent systems, H. Arabnia (ed.), *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Volume II*, pp. 1132-1140.

Tanenbaum, A. S., & van Steen, M. (2002). *Distributed Systems, Principles and Paradigms*, Upper Saddle River, NJ: Prentice Hall.

Tang, L. & Crovella, M. (2003). Virtual landmarks for the Internet, *Proceedings of Internet Measurement Conference*, pp. 143-152, New York, NY: ACM Press.

Tangmunarunkit, H., Govindan, R., Jamin, S., Shenker, S., & Willinger, W. (2001). Network topologies, power laws and hierarchy, Technical Report, TR01-746, University of Southern California.

Tennenhouse, D. L., & Wetherall, D. J. (1996). Towards an active network architecture, *Computer Communication Review*, 26(2), pp. 5-17, April 1996.

Tennenhouse, D. L., Smith, J. M., Sincoskie, W. D., Wetherall, D. J., & Minden, G. J. (1997). A survey of active network research, *IEEE Communications*, 35(1), pp. 80-86.

Theilmann, W., & Rothermel K. (2000a). Optimizing the dissemination of mobile agents for distributed information filtering, *IEEE Concurrency*, pp. 53-61.

Theilmann, W., & Rothermel, K. (2000b). Dynamic distance maps of the Internet, *Proceedings of 19th IEEE Infocom 2000*, pp. 275-284, Tel Aviv, Isreal: IEEE Press.

Tschudin, C. F. (1999). Mobile agent security, Chp. 18 of M. Klusch (Ed.) *Intelligent Information Agents – Agent-Based Information Discovery and Management on the Internet,* pp. 431-445, Berlin: Springer-Verlag.

Vigna, G. (1998). Cryptographic traces for mobile agents, *Mobile Agents and Security, Lecture Notes in Computer Science Vol. 1419*, pp.137-153, Springer-Verlag.

Wilhelm, U. G., Staamann, S., & Buttyan, L. (1998). On the problem of trust in mobile agent systems, *Proceedings of NDSS'99,* San Diego, CA: Internet Society Press.

Wilhelm, U. G., Staamann, S., & Buttyan, L. (1999). Introducing trusted third parties to the mobile agent paradigm, *Secure Internet Programming – Security Issues for Mobile and Distributed Objects, Lecture Notes in Computer Science Vol. 1603*, pp. 469-489, Springer-Verlag.

Wohlmacher, P. (1999). Introduction to the taxonomy of multiple cryptography, *Proceedings of the Multimedia and Security Workshop at ACM Multimedia'99,* Orlando, FL: ACM Press.

Wu, T., Malkin, M., & Boneh, D. (1999). Building intrusion tolerant applications, *Proceedings of the 8th USENIX Security Symposium*, pp. 79-91, Washington DC: Usenix.

Yee, B. S. (1999). A sanctuary for mobile agents, *Secure Internet Programming – Security Issues for Mobile and Distributed Objects, Lecture Notes in Computer Science Vol. 1603,* pp. 261-273, Springer-Verlag.

Young A., & Yung, M. (1997). Sliding encryption: A cryptographic tool for mobile agents, in E. Biham (Ed.) *Proceedings of the 4th International Workshop on Fast Software Encryption, Lecture Notes in Computer Science Vol. 1267*, pp. 230-241, Springer-Verlag.

Zander, J., & Forchheimer, R. (1988). The SOFTNET project: a retrospect, *Conference Proceedings on Area Communication, 8th European Conference on Electrotechnics, EUROCON'88,* pp. 343-345, Stockholm, Sweden: IEEE Press.

BIOGRAPHICAL SKETCH

Oguz Kaan Onbilger received his Bachelor of Science degree from the Department of Computer Science and Engineering at Hacettepe University, Turkey, in 1990. He received his Master of Science degree in computer engineering from Middle East Technical University, Turkey, in 1995. He will receive the Doctor of Philosophy degree from the Department of Computer and Information Science and Engineering at the University of Florida in December 2004. Between and during the academic programs he completed, he worked in the industry several years before he joined the doctorate program. His research interests are computer networks and security, mobile code systems, and Internet/distributed computing.