Design and Analysis of Mobile Agent Communication Protocols

In partial fulfillment of the requirements for the degree of

Master of Science

by

Xinyu Feng

Supervised by

Prof. Jian Lu

Institute of Computer Software

Nanjing University, China

May 20, 2002
To My Family
Acknowledgments

First and foremost, I would like to thank my advisor, Prof. Lu Jian, for his guidance and help. It is Prof. Lu who introduces me to the domain of mobile agents and helps me find my way. His moral and financial support, his gentle way of teaching, his generosity and kindness, and his high standards of integrity have been inspirational to me and crucial to the completion of this thesis.

I would also like to thank Prof. Cao Jiannong at the Hong Kong Polytechnic University, who supervised my 6-month research in Hong Kong. Prof. Cao provides remarkable insights into many aspects of my academic and research life. His enthusiasm and wisdom also influence me greatly and teach me a lot. I regard it an honor to have worked with him.

All my colleagues in the Institute of Computer Software also play an important role in the completion of this thesis. I owe much to members of the Mogent group. They are Dr. Tao Xianping, Dong Huan, Li Xin, Zhang Guanqun and Cao Chun. It is always a pleasure to discuss and exchange ideas with them. Their valuable advice helped me achieve much in my study and research. I'm grateful to Ma Xiaoxing, for his useful comments and suggestions. I'm indebted to Cao Chun, Ren Xiaoxin and Lin Xiangyu for their technical support. Sincerely thanks go to all other colleagues at the Institute of Computer Software, for their pleasant companionship in my three-year graduate study and all the happy time we spent together.

Finally, I would like to express my special thanks to my family and my girlfriend, Ms. Zhang Xiaokun. It is their love, understanding, and support that enabled me to accomplish this thesis.
Abstract

Mobile agents are regarded as the future of distributed computing. They are promising to offer a unified and scalable framework for applications in widely distributed heterogeneous open networks, such as Electronic Commerce, Parallel Computing, Information Retrieval, etc. Among essential features of mobile agents, communication is a fundamental ability that enables mobile agents to cooperate with each other by sharing and exchanging information and partial results, and collectively making decisions.

Although process communication has been a cliché in distributed systems research, the presence of mobility raises a number of new challenges in designing message delivery protocols for effective and efficient communications between mobile agents. In designing such a protocol, two fundamental issues must be addressed: (1) tracking the location of the target mobile agent, and (2) delivering the message reliably to the agent.

Many protocols have been proposed in recent years to solve above problems. However, they all have different assumptions and design goals, and their own ways of solving the problems. Due to lack of good understanding of requirements of mobile agent message delivery protocols with respect to different classes of applications, there have been no uniform and structured methods for characterizing protocols and it is difficult to evaluate the effectiveness and performances of particular protocols.

In this thesis we present our in-depth research on the design of mobile agent communication protocols. Contributions of this thesis include:

1. Requirements for the mobile agent message delivery protocols are analyzed, which can be used as excellent guidelines for users to design new protocols or select an appropriate protocol from existing ones for particular applications.

2. A highly flexible and adaptive mailbox-based scheme is proposed, which introduces the
idea of decoupling the agent's mailbox with the agent itself, resulting in a decoupled problem of locating the agent and delivering messages to it. This flexible approach allows users to design a variety of protocols that can be made adaptive to specific application requirements by evaluating the trade-offs between various design considerations.

3. Based on our understanding of requirements for the mobile agent message delivery protocols and the large design space of our mailbox-based scheme, a very generic framework is proposed for designing such protocols. The generic framework can be used to (1) characterize and evaluate various mobile agent communication protocols; (2) help users to clearly specify their requirements; and (3) help users design a flexible, adaptive protocol which can be customized to meet their requirements.

4. Derived from the framework, one protocol named ARP (Adaptive and Reliable Protocol) is described in detail. The protocol satisfies requirements of mobile agents message delivery protocols, i.e. location transparency, reliability, efficiency, asynchrony, and adaptability. ARP is further optimized using a path-compression and garbage collection algorithm, which makes it to work more efficiently. A fault-tolerant version of ARP is also proposed so that the protocol can work in the presence of network and host failures.

5. Push and pull are two canonical techniques for distributed information dissemination. In this thesis pros and cons of these two modes are compared in the context of mobile agent communication. Variations of the basic pull modes and a dynamic combination of push and pull-based algorithms are proposed for high adaptability and efficiency. Analysis of trade-offs between push and pull can be used as guidelines for choosing design parameter of our generic framework.

**Keywords:** Mobile Agents, Communication, Protocol, Framework, Tracking, Message Delivery
Contents

Acknowledgments ........................................................................................................................................1
Abstract ..........................................................................................................................................................ii
Chapter 1. Introduction ...............................................................................................................................1
  1.1 Introduction of Mobile Agents ..................................................................................................1
    1.1.1 Software Agents .................................................................1
    1.1.2 Mobile Agents .................................................................2
  1.2 Remote Communication between Mobile Agents .................................................................4
    1.2.1 Necessity of Remote Communication between Mobile Agents ..........5
    1.2.2 Requirements of Message Delivery Protocols ...............................................6
  1.3 Contributions of the Thesis ........................................................................................................9
  1.4 Organization of the Thesis .......................................................................................................11
Chapter 2. Background and Related Work ..........................................................................................12
  2.1 The Naming Scheme ................................................................................................................12
  2.2 Tracking Mechanisms ..............................................................................................................13
    2.2.1 The Central Server Scheme ..................................................13
    2.2.2 The Forwarding Pointer Scheme ..............................................14
    2.2.3 Broadcast ................................................................................15
    2.2.4 Hierarchical Schemes ............................................................15
  2.3 Efforts on Reliable Message Routing ...................................................................................16
    2.3.1 The Forwarding Pointer Scheme ..............................................16
    2.3.2 Resending-Based Scheme .......................................................17
    2.3.3 Broadcast ................................................................................17
    2.3.4 Synchronization Between Message Passing and Target Migration ..........18
  2.4 Adaptive Protocols ....................................................................................................................19
Chapter 3. The System Model and Basic Assumptions ....................................................................20
  3.1 The System Model ....................................................................................................................20
  3.2 Basic Assumptions ....................................................................................................................22
  3.3 Terminology Definitions ..........................................................................................................23
  3.4 Summary .....................................................................................................................................23
Chapter 4. The Generic Framework .....................................................................................................25
  4.1 The Three-Dimensional Design Model ................................................................................25
    4.1.1 Mailbox-to-Agent Message Delivery ...........................................25
    4.1.2 Frequency of Mailbox Migration ...............................................26
    4.1.3 Synchronization of Message Forwarding and Agent/Mailbox Migration ......27
  4.2 Parameter Combination ...........................................................................................................28
    4.2.1 The Home-Server Based Protocols .............................................29
    4.2.2 Forwarding-Pointer Based Delivery Protocols ..................................30
    4.2.3 Distributed Registration-Based Protocols ......................................31
  4.3 Summary .....................................................................................................................................32
Chapter 5. The Adaptive and Reliable Protocol .................................................................................34
Chapter 1. Introduction

1.1 Introduction of Mobile Agents

1.1.1 Software Agents

Software agents are a new paradigm for developing software applications. More than this, agent-based computing has been hailed as the “next significant breakthrough in software development” [Sar92], and “the new revolution in software” [Gui94]. Currently, agents are the focus of intense interest on the part of many sub-fields of computer science and artificial intelligence. Agents are being used in an increasingly wide variety of applications, ranging from comparatively small systems such as email filters to large, open, complex, mission critical systems such as air traffic control.

Although the term of “software agent” appears frequently today, there has not yet a precise definition of a software agent. One definition of “software agent” that many agent researchers might find acceptable is: a software entity which functions continuously and autonomously in a particular environment, often inhabited by other agents and processes [Sho97]. D.B. Lange [Lan98] gives a detailed definition of a software agent as follows:

**From end-user perspective**, a software agent is a program that assists people and acts on their behalf. Agents function by allowing people to delegate work to them. A property shared by all agents is the fact that they live in some environment. They have the ability to interact with their execution environment, and to act asynchronously and autonomously upon it. No one is required either to deliver information to the agent or to consume any of its output. The agent simply acts continuously in pursuit of its own goals.

In contrast to software objects of object-oriented programming, agents are active entities that work accordingly to the so-called Hollywood Principle:” Don’t call us, we’ll call you!”
From System Perspective, an software agent is a software object that

1) is situated within an execution environment

2) possesses the following mandatory properties:
   a) Reactive: senses changes in the environment and acts accordingly to those changes;
   b) Autonomous: has control over its own actions;
   c) Goal driven: is pro-active;
   d) Temporally continuous: is continuously executing;

3) and may possess any of the following orthogonal properties:
   a) Communicative: be able to communicate with other agents;
   b) Mobile: can travel from one host to another;
   c) Learning: adapts in accordance with previous experience;
   d) Believable: appears believable to the end-user.

1.1.2 Mobile Agents

Emphasizing mobility and interaction, mobile agents are one of the most promising technologies in software agents research. From the perspective of end-users, a mobile agent is a software agent that can roam autonomously during its execution around the heterogeneous network to finish the task assigned by its owner. From system perspective, a mobile agent is an autonomous object or object cluster, which are able to move between locations in a so-called mobile agent platform (MAP). An MAP is a distributed abstraction layer that provides the concepts and mechanisms for mobility and communication on one hand, and security of the underlying system on the other hand [Str96, Pha98].

A mobile agent has the unique ability to transport itself from one system in a network to another in the same network. This ability allows it to move to a system containing an object with which it wants to interact and then to take advantage of being in the same host or network as the object. The mobile agent technology provides following benefits for creating distributed systems [Lan99]:

- 2 -
They reduce the network load and overcome the network latency. Mobile agents allow users to package a conversation and dispatch it to a destination host where interactions take place locally. Thus remote interactions over the network are reduced and the network traffic is decreased. To move the computation to the data rather than data to the computation, mobile agents can also reduce the flow of raw data in the network. The local interactions and data processing also overcome the network latency, which is critical in real-time systems.

They encapsulate protocols. The Internet’s fast growth has increased the number of computers, protocols and data formats for data exchange to a point, where it is cumbersome if not impossible to upgrade protocol code properly. As a result, protocols often become a legacy problem. Mobile agents permit new protocols to be installed automatically, and only as needed for a particular interaction.

They execute asynchronously and autonomously. In traditional client-server systems, the client performing a task must be available to receive incoming messages and react to them; therefore continuous connection between the client and the server is needed. However, in mobile environment where wireless connection is often expensive and fragile, the requirement of continuous connection is probably not economically or technically feasible. To solve this problem, tasks can be embedded into mobile agents, which can then be dispatched into the network. After being dispatched, the agents become independent of the process that created them and can operate asynchronously and autonomously. The mobile device can be disconnected as soon as the agent is transferred and latter reconnect to accept the results of the agent’s task.

They adapt dynamically. By definition, mobile agents are firstly software agents. Therefore they can sense their execution environment (e.g. local computing environment, local information resource, or network bandwidth or connection) and react autonomously to changes. The ability to move also enables multiple mobile agents to distribute themselves dynamically among the hosts in the network to
They are naturally heterogeneous. Network computing is fundamentally heterogeneous, often from both hardware and software perspectives. Because mobile agents are generally computer- and transport-layer-independent (dependent on only their execution environments), they provide optimal conditions for seamless system integration.

They are robust and fault-tolerant. Mobile agent's ability to react dynamically to unfavorable situations and events makes it easier to build robust and fault-tolerant distributed systems. If a host is being shut down, all agents executing on that machine are warned and given time to dispatch and continue their operation on another host in the network.

Although there is no killer application for mobile agents, there are plenty of applications that benefit from using mobile agents, such as E-commerce, personal assistance, secure brokering, distributed information retrieval, telecommunication networks services, parallel processing, etc.

1.2 Remote Communication between Mobile Agents

Communication is an essential ability for mobile agents to collaborate with others by information exchanging and knowledge sharing. Many works on agent communication languages, such as KQML [Fin94, Lab97] or FIPA ACL [FIP97], has been proposed. However, in this thesis, we do not study the common semantic layer for knowledge sharing. Instead we focus our discussion on the underlying transportation layer of inter-agent communication and concern solely with the delivery of opaque application data to a target agent, which is closer to the tradition of research on distributed systems.
1.2.1 Necessity of Remote Communication between Mobile Agents

The typical use of a mobile agent paradigm is for bypassing a communication link and exploiting local access to resource on a remote server. Thus one could argue that, all in all, communication with a remote agent is not important and a mobile agent platform should focus instead on the communication mechanisms that are exploited locally, i.e. to get access to the server or to communicate with the agents that are co-located on the same site. Many mobile agent systems provide mechanisms for local communication, either using some sort of meeting abstraction as initially proposed by Telescript [Whi96], event notification for group communication [Bau97a, Lan98], or, more recently, tuple spaces [Cab98, Pic99].

Nevertheless, as we will argue, remote inter-agent communication is also a fundamental facility in mobile agent platforms. Its necessity can be shown from following aspects:

(1) The mobile computing paradigm makes good compensation to traditional distributed computing which is based on message passing or RPC, but it cannot completely replace the traditional computing mode. Although the mobility of agents has the potential benefits of reducing network traffic and overcoming network latency, these benefits are obtained at the expense of transmitting the state and code of the mobile agent across the network. If the network traffic caused by agent migration is larger than the cost of sending requirements to remote services and receiving the results from the remote server, message passing will be more efficient in terms of network traffic than agent mobility. Experiments [Chi97, Gra01, Str97] have shown that in many scenarios the most efficient way is to combine message passing with agent mobility.

(2) Mobile agent platforms can be used as general-purpose distributed computing middleware, which combine mobility with message passing naturally. Since mobile agents are generally computer- and transport-layer-independent (dependent
on only their execution environments), mobile agent technology can also be used to deploy distributed systems easily over heterogeneous network, in which an agent can be an encapsulated component of the system and not necessarily mobile (called stationary agent). From this point of view, message passing between remote agents, the most popular communication mode in traditional distributed computing, should be an indispensable mechanism in mobile agent platforms.

(3) In many applications cooperating mobile agents need to exchange information and partial results, and collectively making decisions, while migrating around the network. For instance, when using mobile agents for information retrieval over the network, it is efficient for cooperating agents to share the partial results while each of them searching their sub-area, so that the search space can be considerably reduced [Bau97b]. In another example, a mobile agent could visit a site and perform a check on a given condition. If the condition is not satisfied, the agent could register an event listener with the site. This way, while the mobile agent is visiting other sites and before reporting its results, it could receive notifications of state changes in the sites it has already visited and decide whether a second visit is warranted. Other application can be find in [Cao01, Shi99].

1.2.2 Requirements of Message Delivery Protocols

In this thesis, we choose message passing as the communication mechanism we adapt to mobility, because it is a basic and well-understood form of communication in a distributed system. This incurs no loss of generality because more complex mechanisms such as remote procedure call and method invocation are easily built on top of message passing.

Although inter-process message has been a cliché in distributed systems research, agent mobility raises a number of new challenges in designing message delivery mechanisms for effective and efficient communications between mobile agents.

**Location Transparency:** Since a mobile agent has its autonomy to move from host to
host, it is unreasonable, if not impossible, to require that agents have a priori knowledge about their communication peers’ locations before they send messages. Therefore, the first requirement of a practical mobile agent communication protocol is to allow mobile agents to communicate in a location transparent way, i.e., an agent can send messages to other agents without knowing where they reside physically. The message delivery protocol, therefore, is required to keep track of the location of mobile agents.

**Reliability:** A desirable requirement for any communication mechanism is reliability. Programming primitives that guarantee that the data sent effectively reach the communication target, without requiring further actions by the programmer, simplify greatly the development task and lead to applications that are more robust.

In traditional distributed systems, reliability is typically achieved by providing some degree of tolerance to faults in the underlying communication link or in the communicating nodes. However, fault-tolerance techniques are not sufficient to ensure reliability in systems that exhibit mobility. Because mobile agents are typically allowed to move freely from one host to another according to some a priori unknown pattern, it is difficult to ensure that the data effectively reach the mobile agent before it moves again. If this condition is not guaranteed, data loss may occur. Thus, the challenge to reliable communication persists even under the assumption of an ideal transport mechanism, which itself guarantees only the correct delivery of data from host to host despite the presence of faults. It is the sheer presence of mobility, and not possibility of faults, that undermines reliability.

In this thesis, to concentrate our efforts to overcome the message loss caused by the asynchronous nature of message passing and agent migration, we firstly do not deal with fault-tolerant issues and all our discussion is based on the assumption of a fault-free network. In the following part of the thesis, by **reliability** we mean no matter how frequently the target agent migrates, messages can be routed to it in a **bounded** number of hops. In Chapter 6 we will propose fault-tolerance mechanisms to overcome faults that might occur in the system.
Efficiency: The cost of a protocol is characterized by the number of messages sent, size of the messages and the distance traveled by the messages. An efficient protocol should attempt to minimize all these quantities. More specifically, a protocol should efficiently support two operations: “migration” that facilitates the move of an agent to a new site, and “delivery” that locates a specified agent and delivers a message to it. The objective of minimizing the overhead of these two operations results in conflicting requirements [Awe95]. To illustrate this trade-off, consider the two extreme strategies, namely Full-information strategy, in which every host in the network maintains complete up-to-date information about the whereabouts of every agent, and the No-information strategy, which does not require any update of information for mobile agents during migration. Clearly, the former strategy makes the “delivery” operation cheap, but the “migrate” operation becomes very expensive because it is necessary to update the location information at every host. With the latter strategy, the “migration” operation has a zero cost but the “delivery” operation has a very high overhead because it requires searching over the whole network. In general, a protocol should perform well for any or some specific communication and migration pattern, achieving a balance of the trade-off between the costs of “migration” and “delivery”.

Asynchrony: Here, the term asynchrony includes two aspects of meanings, i.e. asynchronous migration and asynchronous execution. The former means that the agent can freely migrate to other hosts whenever necessary. Although coordination of message forwarding and agent migration is necessary to guarantee reliable message delivery, agent mobility should not be over constrained by frequent and tight synchronization. The latter means the agent is independent of the process that created it and the agent home can be disconnected as soon as the agent is transferred. The mobile agent’s asynchronous execution should not be restricted by heavily relying on the agent home for locating the agent and delivery of every message to the agent. In one word, since asynchrony is regarded as an important advantage of the mobile agent paradigm [Che97, Lan99], it is desirable that the protocol can keep the asynchrony of
both migration and execution so that little or no offset of the merits of mobile agent technology will be introduced.

**Adaptability:** Different Applications may have different requirements and thus different emphasis on the above issues. In some applications, asynchrony is favored and thus the agent home should not be relied as the sole location server. In other applications, reliability is more important so synchronization is needed. Different inter-agent communication and agent migration patterns may also have different implications on the update and search cost. Although protocols can be designed for specific applications to achieve optimal performance, it is desirable to have an adaptive protocol in a general-purpose mobile agent system, which can suit as many kinds of applications as possible.

These requirements serve as a guideline for our work on the design of the mobile agent communication protocols, which will be presented in the following chapters.

**1.3 Contributions of the Thesis**

In this thesis, we present a generic framework for designing mobile agents message delivery protocols that satisfy the requirements mentioned above. The framework uses a flexible and adaptive mailbox-based scheme, which associate each mobile agent with a mailbox while allowing the decoupling between them. A mailbox, which is a message buffer used to store incoming messages destined to an agent, can be detached from its owner agent in the sense that the mailbox can reside at a location different from the current location of the owner agent. This flexible approach allows us to design a variety of protocols that can be made adaptive to specific application requirements by evaluating the trade-offs between various design considerations. We describe taxonomy of MA communication protocols using a three-dimensional model based on the proposed scheme. The three dimensions are *mailbox-to-agent message delivery, mailbox migration, and synchronization of message-forwarding with*
Message delivery protocols can be described by combining parameters in these three dimensions.

The generic framework not only covers, as particular cases, several known protocols, but also allows for the design of new ones that are suited for various application requirements. We describe such an efficient and adaptive protocol derived from the model. The protocol, called ARP (Adaptive and Reliable Protocol), guarantees reliable delivery of messages to mobile agents and satisfies other requirements described above. To demonstrate the adaptability of the protocol, using an analytic model as well as extensive simulation experiments, we analyze the impact of mailbox migration frequency on the performance of the protocol, and evaluate the performance under different settings with the comparison of known existing protocols. The results show that the proposed adaptive protocol can be designed to outperform existing protocols in terms of reducing the costs of both the migration and the message delivery operations.

The ARP protocol is further improved in two aspects. Firstly, we propose a path-compression algorithm to remove the redundant host in the migration path of the mailbox. Simulation results show that, by using the path-compression algorithm, the migration cost of the mobile agent can be greatly reduced. This algorithm can also be used for garbage collection, i.e. clearing the useless address of mobile agents cached by hosts in the network. Secondly, fault-tolerance of ARP is improved to handle the crash of hosts on the agent migration path.

Push and pull are alternatives on the dimension of mailbox-to-agent message delivery in our three-dimensional model. Pros and cons of these two modes in the mobile environment are fully discussed and simulation results are presented to compare them in terms of communication overheads and delay of message processing. Two improved version of the basic pull approach, namely greedy pull and distance-based pull, are proposed to reduce delay of message processing and network traffic, respectively. Directions to choose between push and pull modes under different
communication and migration patterns of mobile agents are presented and an algorithm of combining the push and pull modes are proposed to achieve better performance and flexibility.

1.4 Organization of the Thesis

The remaining part of the thesis is organized as follows. Chapter 2 presents a review of related work. Chapter 3 introduces the system model and some basic assumptions of our work. Chapter 4 proposes our generic framework and briefly introduces some of particular protocols derived from the framework. Chapter 5 describes the ARP protocol in detail, which is an adaptive protocol derived from the generic model. We present the design of the protocol, as well as the analysis of its design trade-offs and performance, using both theoretical analysis and simulations. Results and observations are discussed. Impacts of mailbox migration frequency on the performance of ARP are also analyzed in this Chapter. Chapter 6 presents two optimized versions of ARP. In the first version, redundant hosts in the mailbox migration path are removed using the path compression and garbage collection algorithm. In the second version fault-tolerance is improved. Chapter 7 analyzed trade-offs of choosing the push and pull modes for message delivery between the mailbox and the agent. Improved versions of the basic pull mode are proposed. Chapter 8 provides some concluding remarks and directions of our future work.
Chapter 2. Background and Related Work

As analyzed in Chapter 1, the message delivery protocol for mobile agents should satisfy the requirements of location transparency, reliability, efficiency, asynchrony, and adaptability. Location transparency and reliability are two basic requirements of an effective protocol. More specifically, to satisfy these two requirements, the message delivery protocol should be able to:

1. Identify communicating agents in a globally unique fashion. The agent ID should not change whenever the agent migrates to other hosts.

2. Map the ID of the receiver agent to its current address. To deliver messages to an agent, the underlying transport layer must know the current address of the receiver. Since the agent ID do not contain the location information of the agent, the mobile agent platform should support agent-tracking mechanisms which map the agent ID to its current location.

3. Deliver the message reliably to its target agent. This process can be done either in parallel with agents tracking or in a second phase after the address has been got. In both cases, the message delivery scheme should overcome the message loss or chasing problem caused by agent migration.

In the following sections, we provide a review of related work on these three design issues. Work to meet other requirements, such as efficiency and adaptability, is also surveyed.

2.1 The Naming Scheme

There are two basic requirements of the naming scheme of mobile agents. (1) Since a mobile agent can migrate from one host to another, the agent should be identified in a globally unique fashion. (2) To let the mobile agents communicate in a location transparent way, the agent ID should keep unchanged during its lifecycle, even if the
physical address of the agent has changed.

The usual way to identify a mobile agent is to append the name of the agent’s origin host (i.e. agent home) with its title (a free form string used to refer to this agent) [Bel99, Tao00]. The name of the agent home can be either its IP address or its URL. In both cases there should be no two hosts having the same name. Thus it is impossible for agents born at different agent platforms to have the same ID. For agents created at the same host, the host is responsible to manage the name space to ensure that each agent created in it has a unique title. Since the name of the agent home and the title of the agent will not affected by the physical location of the agent, the ID of the agent will keep unchanged during its life cycle (we assume that the name of the agent home will not change during the agent’s life cycle).

We adopt this naming scheme in this thesis.

### 2.2 Tracking Mechanisms

The task of the tracking mechanisms is to obtain the current location of mobile agents. With the presence of mobility in distributed systems, many mobile units tracking schemes have been proposed in the last several years in different contexts, including mobile agents, mobile and wireless communications, and wide-area distributed systems. According to the organization of location servers, the major schemes can be categorized into central server, forwarding pointers, broadcast, and hierarchical location directory.

#### 2.2.1 The Central Server Scheme

The central server scheme makes use of a location server to keep track of the physical location of a mobile object. There are several variations. For example, the Mobile IP protocol [Per96], which is designed for routing IP packets to mobile hosts, uses the home-server scheme, where a mobile host registers its care-of-address with its home agent every time it moves. All the IP packets to the host are sent to the home agent,
which forwards the packets to the host. This scheme is also used in IS-41 [Gal97] for personal communication service (PCS), as well as in mobile agent systems [Mil98, Lan98] and distributed systems.

The central server scheme is simple to implement and has less communication overhead for locating a mobile object. However, it incurs large overhead of updating the locations and delivering messages. The server can be a bottleneck of performance if the number of mobile objects is growing and communication and migration are frequent. It can also be a single-point of failure. The scheme do not support locality of mobile object migration and communication, i.e. migration and communication involve the cost of contacting the server which can be far away. This is the well-known triangle routing problem [Per94]. Cache-based strategies [Lin94, Har94] are proposed to avoid the triangle routing problem. If a cache miss occurs the server is contact for new location. In the Internet Mobile Host Protocol (IMHP) [Per94], packets are forwarded along the forwarding address left by the mobile host if a cache “miss” occurs. All these protocols do not handle message loss caused by mobility.

2.2.2 The Forwarding Pointer Scheme

In the forwarding pointer based schemes for tracking mobile objects [Des98, Mor99, Obj97], each host on the migration path of an object keeps a forwarding pointer pointing to the next host on the path. Each sender knows the home of the target object. Messages are sent to agent home and forwarded to the target object along the forwarding pointers.

The forwarding pointer scheme is also easy to implement and incurs no location registration overhead. However the scheme cannot guarantee message delivery because a message may follow a mobile object which frequently migrates, leading to a race condition. Furthermore, it is not practical for a large number of migrations to distinct hosts (a chain of pointers is growing, increasing the cost of message delivery). Some path compression methods can be used to collapse the chain, e.g.
movement-based and search-based. In the former case the mobile object would send backward a location update after performing a number of migrations; in the latter case, after receiving a number of messages (i.e. after a number of message delivery operations occur). For instance, a search-based path compression technique is proposed in [Des98]. After a message is routed to the target object along the chain, an Update_Entry message is sent back along the chain and forwarding pointers kept in the nodes of the chain are updated. A similar algorithm has been used in Emerald [Jul88], where the new forwarding address is piggybacked onto the reply message in the object invocation.

### 2.2.3 Broadcast

There are three variants of the broadcast scheme, i.e. query broadcast, data broadcast, and notification broadcast. The first two are proposed from the perspective of the message sender. In the query broadcast scheme, the message sender sends query message to all the hosts in the system for the location of the receiver. After receiving the response from the host at which receiver resides, the sender sends the message to the location obtained from the response. In the data broadcast scheme the sender broadcast the message directly to all the hosts in the system. The third is proposed from the perspective of the receiver. After migration the mobile agent broadcast its new location to all the hosts in the system.

Broadcast schemes have less reliance on the agent home for agent tracking or message forwarding, thus they can maintain the disconnected operation ability of mobile agents. They can be implemented in local Internet domain or local Ethernet. Broadcast can be accomplished efficiently in bus-based multiprocessor systems. They are also used in radio networks. However, because of the large communication overhead, it is impractical to broadcast in large-scale networks [Rat01].

### 2.2.4 Hierarchical Schemes

In the hierarchical schemes, a tree-like hierarchy of servers forms a location directory
(similar to DNS). Each region corresponds to a sub-tree in the directory. For each agent there is a unique path of forwarding pointers that starts from the root and ends at the leaf that knows the actual address of the agent. Messages to agents are forwarded along this path. This kind of schemes and their variants have been used to track mobile users [Awe95, Rat01, Kri94], objects [Ste98] and agents [Bel99, Laz98]. In [Kri94] the author explored different update and search strategies that can be used in the hierarchical scheme.

The hierarchical scheme scales better than forwarding pointers and central servers. It supports locality of mobile object migration and communication. However, the hierarchy is not always easy to construct, especially in the Internet environment. The hierarchical scheme itself cannot guarantee message delivery. Messages might also chase their recipients under this scheme.

Readers are referred to [Woj01] and [Pit01] for excellent surveys of above techniques.

2.3 Efforts on Reliable Message Routing

Tracking mechanisms can map the agent’s location-independent ID to its current location. However, they are not sufficient to guarantee message delivery. As discussed in Chapter 1, even though the sender knows the current location of the receiver, the receiver may migrate to other hosts during message transmission. Various approaches have been proposed to overcome message loss caused by migration of recipients.

2.3.1 The Forwarding Pointer Scheme

The idea is the same with the one introduced in Section 2.2.2. Before migration, the mobile object leaves a pointer in its current host pointing to the target host. When a message is sent to an obsolete address of the recipient (this address can be obtained by any of the tracking schemes introduced in Section 2.2), the message is routed along
the forwarding pointer. The forwarding pointer scheme is often used in combination with address caching. A case in point is IMHP [Per94].

Although messages can be routed along the forwarding pointer, there is not an upper bound of the number of hops a message takes before it reaches the recipient. If the recipient migrates frequently, the message may keep chasing the recipient and couldn’t be received until the death of the recipient. Therefore the forwarding pointer can only partially overcome the message loss caused by agent mobility and cannot guarantee reliable message delivery (message routing).

2.3.2 Resending-Based Scheme

To implement reliable message delivery for mobile objects, resending-based TCP-like protocols [Ran00, Oko99, Bak94] are proposed. If a message is missed because of the migration of the recipient, the sender can detect the message loss and resend the message to the new address of the recipient. Using TCP-like slide window mechanism, these protocols can not only overcome message loss caused by both migration of recipients and faults of the network, but also maintain the FIFO order of message delivery. However, as in the forwarding based scheme, when the recipient migrates frequently, there is no upper bound of the number of message resending. Therefore, it cannot satisfy our requirement of reliability mentioned in Chapter 1.

2.3.3 Broadcast

If the sender maintains an obsolete address of the recipient and the message sent to that address couldn’t be delivered to the recipient, the message will be broadcasted to all the hosts in the system. This idea is similar to the data broadcast mentioned in Section 2.2.3, but it is used only when the communication failure occurs because of the recipient’s migration. In Emerald [Jul88] broadcast is used to find an object if a node specified by a forwarding pointer is unreachable or has stale data. According to Murphy [Mur99], however, the simple broadcast cannot avoid message loss caused by object mobility. Murphy proposed in this paper a snapshot based broadcast scheme to
guarantee reliable delivery of messages to highly mobile agents. The protocol can also be extended for group communication for mobile agents.

2.3.4 Synchronization Between Message Passing and Target Migration

From the perspective of concurrency control, the message loss or chasing problem in the message delivery process is caused by concurrent and asynchronous access to the location information of the target agent. The mobile agent migration and the message delivery processes can be regarded as two kinds of database operations. The migration of target agent changes its actual address, which can be regarded as a “write” operation of the location information. The message delivery process needs the target agent’s actual address, which in fact is a “read” operation of the location information. Strategies are proposed to synchronize the message passing and target migration so that messages can reach the target agent within bounded number of hops.

One widely used synchronization strategy is implemented as follows. Before migration the mobile agent informs all the hosts (usually the home of the agent or a central message forwarding server) that might send messages to its current address and waits for ACK from each host (containing the number of messages sent from the host). It then waits for these messages due to arrive. After migration it tells these hosts it has finished moving. During migrations (after sending ACK) the host suspends message forwarding. Variations of the strategy are proposed. For instance, if FIFO message order is maintained in the underlying transport layer, the ACK message does not need to contain the number of messages sent from the host and the agent can leave for the target host as soon as it has collected all the ACK messages. In the Mogent system [Tao00], a synchronous home-server based protocol is proposed to track mobile agents and guarantee message delivery.

The synchronization scheme can guarantee that messages be routed to its target agent with bounded number of hops. However, the agent has to wait for all the ACK messages from message forwarding servers. If there are multiple servers that might
forward messages to the agent, the constraint on the mobile agent migration is prohibitive. In this thesis we also use this kind of synchronization scheme to realize reliable message delivery. However, using a mailbox-based scheme, only the migration of the agent's mailbox is constraint. The agent can move freely about the network.

2.4 Adaptive Protocols

To suit for different mobility patterns, many adaptive algorithms have been proposed in the field of personal communication, including timer-based, movement-based, distance-based and state-based location-updating algorithms [Bar95, Won00]. In these algorithms, mobile users decide whether to update their location information according to different factors concerned. To optimize the location management cost on a per-user basis, a selective location update strategy for PCS users is proposed in [Das97, Sen99]. When a mobile user enters a new Location Area (LA), it can choose whether to update its location information or not. According to its own mobility model and call arrival pattern, each mobile user has its update strategy \( S_u = \{u_i\} \), consisting of a set of binary decision variables \( u_i \) (to update or not to update) for all LAs.

In [Kun97] a tracking agent is used for location tracking and message forwarding for cooperating agents. It is dynamically generated when cooperation starts and is killed after the cooperation is finished. The coordinates of the center of the cooperating agents are set to the average coordinates of each agent. If the distance between the center and the tracking agent is large enough, the tracking agent will migrate to the center so that the communication latency between cooperating agents can be decreased. The authors, however, did not discuss how a new agent could find an existing tracking agent in order to join the cooperation by sharing the tracking agent with others.
Chapter 3. The System Model and Basic Assumptions

3.1 The System Model

In our systems, there are four types of entities, namely *hosts*, *mobile agent platforms*, *mobile agents*, and their *mailboxes*. All the *hosts* are connected by the network and can communicate with one another by sending messages on the network. Each host runs a *Mobile Agent Platform (MAP)* providing communication, mobility and security support for mobile agents. The MAP provides two levels of message passing primitives, i.e. high-level location-independent primitives and low-level location-dependent primitives. Low-level primitives are responsible for point-to-point message delivery between MAPs. Mobile agents call the high-level location-independent primitives to send messages between each other. The high-level primitives implement end-to-end message delivery between mobile agents. They are based on these low-level primitives and provide location transparency and overcome message loss caused by agent mobility. The architecture of the communication support in the MAP is depicted in Figure 3.1. The work proposed in this thesis concentrates on the implementation of the tracking and delivery layer of the MAP. In the following part of the thesis, we use the terms *host* and *Mobile Agent Platform (MAP)* interchangeably, both of which are referring to the tracking and delivery layer of the MAP.
Mobile agents execute on hosts with the support of underlying MAP. Each mobile agent can only communicate to the MAP at the host it currently resides. A mobile agent communicates with other mobile agents through its underlying MAP. To identify the mobile agent in a unique fashion, as mentioned in 2.2, the agent ID has the form of \((hmAddr, title)\), where \(hmAddr\) is the name of the agent’s origin host (i.e. home of the agent) and \(title\) is a free form string assigned by the user. The agent ID remains unchanged during the life cycle of the agent. Before sending messages, the sender agent must obtain the ID of the receiver, which can be either hardwired in the sender’s program or retrieved from directory services.

![Fig. 3.2 The detachment of the agent from its mailbox](image)

A mailbox is a message buffer used to store incoming messages. Every mobile agent in the system is allocated a mailbox. Incoming messages sent to the agent are inserted into the mailbox first. As shown in Figure 3.2, if an agent wants to send a message to another agent, it simply sends the message to the receiver’s mailbox (step 1). Later the receiver receives the message from its mailbox using either pull or push (step 2, 3).

The mailbox is logically one part of the agent, but it can be detached from its owner in the sense that the agent can migrate to a new host while leaving its mailbox at a previous host along its migration path. Thus the communication between agents is divided into two steps: the transmission of a message from the sender to the receiver’s mailbox and the delivery of the message from the mailbox to its owner agent. Since the mailbox is also a mobile object (we do not call it another mobile agent dedicated to message delivery because it has no autonomy to decide its migration), the first step
is identical to the inter-mobile-agent communication, thus it can be realized by any existing message delivery strategies. Notice, however, that for a frequently migrating agent, its mailbox can migrate at a much lower frequency. When to migrate the mailbox is a parameter of the protocol design. The second step, i.e. the delivery of the message from the mailbox to its owner agent, raises new issues that will be discussed later in Chapter 4.

3.2 Basic Assumptions

3.2.1 Asynchronous Communication Mode. We assume that mobile agents communicate in the asynchronous mode. This is reasonable because, with mobile agents roaming the Internet, it is rare that two agents use synchronous communication to talk to each other [Ver99]. The large and unpredicted message delays on the Internet, which can easily become on the order of several seconds, prohibit frequent use of synchronous communication in a mobile agent application. Another reason is that in an open system, communicating agents may have no knowledge about each other before establishing the session. They can only get to know each other by some kinds of brokering and matchmaking mechanisms [Klu01]. In this case it is impossible for agents to communicate synchronously with each other.

3.2.2 Properties of Low-Level Primitives. The low level primitives are implemented using asynchronous and point-to-point message passing. We assume that they have abstracted away the network failure for the higher agent-tracking and message-delivery layer. This can be implemented over existing protocols, such as TCP. For ease of discussion, we also assume that the FIFO property is maintained by the low-level primitives. This can be implemented in a MAP by associating a queue that contains messages to be sent to the same remote MAP (implementation of the low-level primitives is discussed in Chapter 6). However, as mentioned in Section 2.3.4, the FIFO property is not obligatory.

3.2.3 Fault-Free Hosts and MAPs. To concentrate the attention on the problems
caused by agent mobility, we do not consider faults of hosts and MAPs throughout this thesis except in Chapter 6, where fault-tolerance issues are discussed. If not explicitly stated, we assume that the underlying hosts and MAPs are fault-free and there is no message lost due to host crash.

3.3 Terminology Definitions

For ease of exposition, let us first define the terminology used throughout the remaining part of the thesis, which includes the migration path of a mobile agent and its mailbox, and the relationship between their locations.

**Definition 1.** Path\(_a(A)\) is an ordered list of hosts, \((h_{a0}, h_{a1}, \ldots, h_{an})\), that agent A has visited in sequence. The set of hosts on the path is denoted as \(S_a(A) = \{h_{ai} \mid h_{ai} \text{ is on Path}_a(A)\}\). The host \(h_{a0}\) is the home of agent A.

**Definition 2.** Path\(_m(A)\) is an ordered list of hosts \((h_{m0}, h_{m1}, \ldots, h_{mn})\) that the mailbox of agent A has visited in sequence. The set of hosts on the path is denoted as \(S_m(A) = \{h_{mi} \mid h_{mi} \text{ is on Path}_m(A)\}\).

By definition, we have \(S_m(A) \subseteq S_a(A)\) and \(h_{m0} = h_{a0}\). The host \(h_{m0}\) is the home of agent A.

**Definition 3.** \(f_A: S_a(A) \rightarrow S_m(A)\) is a function that maps from the location of agent A to that of its mailbox. For every \(h_{ai} \in S_a(A)\) we have:

\[
f_A(h_{ai}) = \begin{cases} h_{ai} & i = 0, \text{ or } i > 0 \text{ and agent A migrates to } h_{ai} \text{ with its mailbox.} \\
    f_A(h_{a(i-1)}) & i > 0 \text{ and agent A migrates to } h_{ai} \text{ without moving its mailbox.} 
\end{cases}
\]

3.4 Summary

In this chapter we present the system model of our work and the mailbox-based scheme, which is the basis of our work and introduces great flexibility into inter-agent message passing. We also introduce basic assumptions of our work and provide definitions of terminologies used throughout the remaining part of this thesis,
including the migration path of an agent, the migration path of a mailbox, and the relationship between the location of an agent and that of its mailbox.
Chapter 4. The Generic Framework

The generic framework proposed in this thesis is based on the mailbox-based scheme illustrated in Figure 3.2. Its flexibility and adaptability comes from the decoupling between a mobile agent and its mailbox, allowing for the separation of the above two different concerns. We also discuss the design space of the framework, identify the relevant parameters and various protocols that can be derived as special cases.

4.1 The Three-Dimensional Design Model

In our generic framework, choices can be made in three aspects of designing a protocol that best suits the specific requirement of an application. The three aspects are mailbox-to-agent message delivery, mailbox migration, and synchronization of message-forwarding with object-migration.

4.1.1 Mailbox-to-Agent Message Delivery

As mentioned before, messages destined to an agent are all sent to the agent’s mailbox and the agent receives the messages later by either a push or pull operation.

- **Push (PS)**. The mailbox keeps the address of its owner agent and forwards every message to it. In this way the real-time message delivery can be implemented and the message query cost is avoided. But the agent must notify the mailbox its current location after every migration.

- **Pull (PL)**. The agent keeps the address of its mailbox and retrieves messages from the mailbox whenever needed. The mailbox does not need to know the agent’s current location and therefore location registration is avoided. But the query of the remote mailbox increases the message delivery overhead and the message may not be processed in real time.
4.1.2 Frequency of Mailbox Migration

In the mailbox-based scheme, the mailbox is detached from its owner agent. Before migration, the agent can decide whether to take its mailbox with it, i.e. determine the value of $f_A(h_{ai})$ ($h_{ai}$/$S_d(A)$). Given a mobile agent, according to how the value of $f_A(h_{ai})$ is set for its $i^{th}$ migration, the migration pattern of the agent’s mailbox can be categorized as follows:

- **No Migration (NM).** In this case, for every $h_{ai}$$S_d(A)$, we have $f_A(h_{ai}) = h_{ai}$, i.e. $S_m(A) = \{h_{ai}\}$. All the messages are sent to the agent home and the agent receives messages from its home (in either a pull or push mode). The cost for tracking the mailbox is zero, but the message delivery cost is high because all the messages must be forwarded by the agent home. The triangle routing [Per94] increases the communication overhead.

- **Full Migration (FM).** For every $h_{ai}$$S_d(A)$, we have $f_A(h_{ai}) = h_{ai}$, i.e. $S_m(A) = S_d(A)$. The mailbox is part of the data of the mobile agent and migrates with the agent all the time. The cost of message delivery between the mailbox and the agent is zero, but it is difficult to track the mailbox. If the agent (and the mailbox) migrates frequently, there is a trade off between the number of messages that could be lost and how much communication overhead will be introduced to guarantee message delivery.

- **Jump Migration (JM).** Between the above two extreme cases, we have cases where $|S_m(A)| > 1$ and $S_m(A) \subset S_d(A)$. Before its $i^{th}$ migration ($i \geq 1$), the agent determines the value of $f_A(h_{ai})$, which can be either $f_A(h_{ai-1})$ or $h_{ai}$. To determine the value of $f_A(h_{ai})$, an agent can consider factors such as the number of messages it will receive at $h_{ai}$ and the distance between $h_{ai}$ and $f_A(h_{ai-1})$. If an agent seldom receives messages from others at $h_{ai}$, it doesn’t need to take its mailbox to the new host, so $f_A(h_{ai})$ is set to $f_A(h_{ai-1})$. On the other hand, if an agent expects to receive messages frequently from others and $h_{ai}$ is far away
from $f_A(h_{a(i-1)})$, it will be expensive to leave the mailbox unmoved and to fetch messages from $f_A(h_{a(i-1)})$. In this case the agent should migrate to $h_a$ together with its mailbox, i.e. let $f_A(h_a) = h_a$. Under the Jump Migration mode, the protocol can work more flexibly based on a decision best suit particular agent migration and inter-agent communication pattern, reducing the cost of both “tracking” and “delivery” operations.

### 4.1.3 Synchronization of Message Forwarding and Agent/Mailbox Migration

In our framework users can choose whether they need reliable message delivery or not. If a higher reliability is required, we increase the degree of synchronization in order to overcome message loss. The synchronization is performed either for coordinating the message forwarding by the host and the migration of the destination mailbox (SHM), or for coordinating the message forwarding from an agent's mailbox and the migration of the agent (SMA), or both (in this last case we call it a full synchronization, i.e. FS).

The synchronization between the message-forwarding object (mobile agent server or the mailbox) and the moving object (mailbox or mobile agent) can be realized in the following way. Before migration, the moving object sends Deregister messages to all objects that might forward messages to it and waits for ACK messages from them. After all the ACK messages have arrived, it moves to the target host and informs all the message-forwarding objects of its arrival by sending them Register messages. The state change of the moving object is shown in Figure 4.1. Messages can be forwarded to the mobile object when it is in “stationary” and “waiting” states and must be blocked when it is in the “moving” state.

![Fig. 4.1 State switching of a mobile object](image-url)
The above three aspects can be used to develop a three-dimensional model, as shown in Figure 4.2. Each of the three aspects represents one dimension in the model, showing a spectrum of different degrees of constraints for that dimension. The three dimensions are orthogonal. That is, each of these three aspects may be discussed independently of each another, and a property in one dimension can, logically, have various combinations with the properties in the other dimensions. For different applications with different requirements in the three aspects, the required degree of properties can be different. Message delivery protocols can be described by combining the parameters in the orthogonal dimensions.

### 4.2 Parameter Combination

The three-dimensional model introduces taxonomy of mobile agent message delivery protocols. In this section, we describe a classification of message delivery protocols according to different parameter combinations. A string of the format XX-YY-ZZ is used to express a protocol, where XX is for NM, JM or FM, YY is for PL or PS, and ZZ is for NS, SHM, SMA or FS. The overall configuration of a protocol has a special value for each of the three parameters. Most combinations have plausible applications. However, brevity precludes a discussion of the full range of protocols which can be derived, and we study the combinations with the most popular features. Table 4.1 shows the different protocols derived from our framework, with the description of
their location registration modes and whether they can satisfy the required reliability.

An “*” in a string denotes a don’t-care state where multiple values are applicable.

<table>
<thead>
<tr>
<th>Protocols</th>
<th>Location Registration</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>NM-PS-NS</td>
<td>Yes (Agent -&gt;Mailbox)</td>
<td>No</td>
</tr>
<tr>
<td>NM-PS-SMA</td>
<td>Yes (Agent -&gt;Mailbox)</td>
<td>Yes</td>
</tr>
<tr>
<td>NM-PL-NS</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>FM-*-NS</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>JM-PL-NS</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>JM-PS-NS</td>
<td>Yes (Agent -&gt;Mailbox)</td>
<td>No</td>
</tr>
<tr>
<td>FM-*-SHM</td>
<td>Yes (Mailbox-&gt;Host)</td>
<td>Yes</td>
</tr>
<tr>
<td>JM-PL-SHM</td>
<td>Yes (Mailbox-&gt;Host)</td>
<td>Yes</td>
</tr>
<tr>
<td>JM-PS-FS</td>
<td>Yes (Agent -&gt;Mailbox, Mailbox-&gt;Host)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### 4.2.1 The Home-Server Based Protocols

All the protocols under the NM mode adopt a home-server approach. In this case the agent home acts as the message-forwarding server.

The NM-PS-NS protocol is identical with the Mobile IP protocol used in mobile computing [Per96]. The agent registers its current location with its mailbox residing at its home. Messages are sent to the mailbox. The mailbox pushes messages to its owner agent. This protocol does not guarantee message delivery. If the agent migrates during the message forwarding, the message will be lost.

To ensure reliable message delivery, synchronization between agent migration and message forwarding from the mailbox (SMA) is needed. This produces the NM-PS-SMA protocol, a synchronized version of the Mobile IP protocol. In the NM-PL-NS protocol the agent pulls messages from its mailbox, therefore the message delivery can be guaranteed without using synchronization.

The home-server based protocols are simple and work well for small-to-medium systems where the number of agents is small. However, the triangle routing will increase the communication overhead, especially when the range of agent distribution...
is large. In a system with a large number of agents and frequent inter-agent communications, the home may become a performance bottleneck and a single point of failure. Furthermore, ability of mobile agents for disconnected execution is constrained because of the dependence of the agent home as a message-forwarding server.

4.2.2 Forwarding-Pointer Based Delivery Protocols

The FM-*-NS, JM-PL-NS, and JM-PS-NS protocols adopt the forwarding-pointer scheme, in which each host $h_{mi}$ in $S_m(A)$ keeps a forwarding pointer to $h_{m(i+1)}$. Each MAP caches the location, say $h_{mk}$, of the target mailbox if there have been messages sent from this MAP to the target agent before. Consequent messages from the MAP are sent to the cached address $h_{mk}$ directly. If there is no address of the target mailbox in the address cache of the MAP, messages from this MAP will be sent to the home of the target agent, i.e. $h_{m0}$. If the mailbox currently resides at $h_{mk}$, the message is inserted into the mailbox. Otherwise messages are forwarded along the forwarding pointers maintained by hosts in $S_m(A)$. When the mailbox receives the message and finds that the sender has outdated knowledge of its address, it notifies the sending MAP its current location and the mailbox’s address cached by the sending MAP is updated.

In the FM-*-NS protocol the mailbox is bound with its owner agent (the mailbox migrates in the FM mode), therefore there is no remote interaction between the mailbox and the agent; and the push and pull modes make no difference for the agent to obtain messages from its mailbox.

The JM-PL-NS and JM-PS-NS protocols are similar to the FM-*-NS protocol except that the mailbox migrates in the JM mode, which can be regarded as a kind of path compression technique. In the cases where $f_{a}(h_{ai}) \neq h_{ai}$, the pull and push modes are used respectively for getting messages from a mailbox.
In the FM-*-NS protocol, the multi-hops path could degrade the communication performance significantly. In both JM-PL-NS and JM-PS-NS protocols, the mailbox of an agent migrates less frequently than the agent. Thus the message-forwarding path is shorter and the communication overhead is reduced. Besides, the chasing problem may be less probable to occur. Notice, however, neither of the protocols can guarantee message delivery.

There is no location updating cost in the forwarding pointer scheme. The sender sends messages to the cached address of the mailbox of the target agent, therefore the workload of the agent home is decreased. Even if the cache is outdated, messages can still be routed to the target mailbox along its migration path. However, if one host in $S_m(A)$ fails, the target agent can no longer be reached. The most serious problem with this scheme is that it cannot guarantee the reliability of message delivery because a message may keep chasing the target agent if the agent migrates frequently. The forwarding pointer based protocols are applicable to applications where mobile agents migrate extensively but communicate rarely. They are especially preferable when users attach more emphasis on agents’ real-time migration rather than reliable message delivery.

### 4.2.3 Distributed Registration-Based Protocols

The forwarding pointer based delivery protocols, as discussed above, do not guarantee reliable message delivery. In our distributed registration-based protocols, including FM-*-SHM, JM-PL-SHM, and JM-PS-FS protocols, synchronization is used to handle this problem. Before migrating to $h_{mi}$, the mailbox Deregister its current address with all the hosts, $h_{m0}, h_{m1}, ..., h_{mi-1}$, in $S_m(A)$ and waits for ACK messages from them. After arriving at $h_{mi}$, the mailbox registers its new address to all these hosts.

The sender sends messages to the address $h_{mk}$ cached by its underlying MAP. If the mailbox has moved out of $h_{mk}$, host $h_{mk}$ forwards the messages to the current address
of the mailbox and notifies the sender of the new address. Synchronization mechanism between message passing and migration, as discussed in Section 4.1, ensures that the target mailbox will not leave before it receives the message forwarded from $h_{mk}$. The distributed registration-based scheme is similar to the synchronized home-server scheme (NM-PS-SMA) but the role of the agent home is distributed to all the hosts in $S_m(A)$. It can also be regarded as forwarding pointer scheme with migration-based path compression technique, i.e. the agent updates all the pointers on its migration path after one (if FM mode is used) or several (if JM mode is used) migrations.

In the FM-*-SHM protocol we have $S_m(A) = S_d(A)$. If the agent migrates frequently the overhead for synchronization and location registration is unaffordable. But if the mailbox migrates in the JM mode, both the times of the mailbox registration and the number of hosts in the set of $S_m(A)$ are reduced. In the JM-PL-SHM protocol, the pull mode is used by the agent to obtain messages from its mailbox. Synchronization between message sending from hosts and migration of mailbox (SHM) is necessary to guarantee message delivery. In the JM-PS-FS protocol, the mailbox pushes each incoming message to its owner agent, therefore, in addition to SHM, synchronization between message pushing from the mailbox and the migration of the owner agent (SMA) is also needed to achieve reliable message delivery. That’s why FS is used in the protocol.

4.3 Summary

In this Chapter, we present a generic framework for designing mobile agent message delivery protocols. The framework uses a flexible and adaptive mailbox-based scheme, which associate each mobile agent with a mailbox while allowing the decoupling between them. Based on a three-dimensional model taxonomy of mobile agent communication protocols is developed, which captures the main features of any message delivery protocol. Message delivery protocols can be described by
parameters in these three dimensions. The generic framework not only covers, as particular cases, several known protocols, but also allows for the design of new ones that are suited for various application requirements. It has the following advantages: (1) it can be used to characterize and evaluate various mobile agent communication protocols; (2) it can help users to clearly specify their requirements; (3) it can help users design a flexible, adaptive protocol which can be customized to meet their requirements.

In the following chapters of this thesis, we analyze the trade-offs of choosing design parameters in our three-dimensional model. In Chapter 5, while evaluating the performance of one specific protocol derived from the generic framework, we illustrate the impact of mailbox migration frequency on the performance of the protocol, using an analytic model as well as extensive simulation experiments. The discussion is also applicable to other protocols derived from the framework. In Chapter 7 we compare the Push and Pull modes used for interaction between mobile agents and their mailboxes. Pros and cons of these two modes are fully explored and their performance are compared in terms of network traffic and message processing delay. Improved versions of the pull operation are proposed and the idea of dynamically combining these two modes for better performance is introduced.
Chapter 5. The Adaptive and Reliable Protocol

In this chapter, we propose an adaptive and reliable protocol derived from the framework. The protocol, called ARP (Adaptive and Reliable Protocol), represents the JM-PL-SHM combination of the parameters. It guarantees reliable delivery of messages to mobile agents and satisfies other requirements described in Chapter 1.

5.1 The Protocol

In the ARP protocol, the mailbox of agent A, namely $M_A$, migrates in the JM mode. Migration of $M_A$ and message forwarding from all the hosts on $Path_m(A)$ are synchronized. The agent A periodically pulls messages from $M_A$. Upon pulling, messages in $M_A$, if any, will be delivered to the agent. In cases there is no message in $M_A$, either a blocking or an un-blocking receive operation can be implemented - the mobile agent can continue its execution or is suspended until a new message arrives. We assume that the “send” operation is always asynchronous (a synchronous “send” can always be simulated by letting the sending agent, after it has put the message in the message system, change to a receiver and wait for an acknowledgement).

The physical address of $M_A$ is maintained in an entry of the address table in each host on $Path_m(A)$. In addition to the address, there are another two fields in the entry. One is a “valid” tag which is used to indicate whether the address of $M_A$ is outdated. Another field is a pointer to a blocked message queue, which is used to temporarily keep the messages sent to $M_A$ if the “valid” tag is false, i.e. $M_A$ is moving on its way to a new host.

The protocol defines operations for two processes: migrating and message-forwarding. In the migrating process, if the agent A decides to migrate with its mailbox $M_A$, it will first de-register $M_A$’s current address $h_{m_i}$ and, upon reaching the new destination host, register the new address with all the hosts $h_{m_j}$ where $0 \leq j \leq i$ and $h_{m_j} \in S_m(A)$. In the
message-forwarding process, messages are sent to \( M_A \) at the address cached by the sender. If \( M_A \) has been moved to another host, the messages will be forwarded to its current address. The protocol is illustrated by Figures 5.1 and 5.2, and presented more formally in pseudo Java code as shown in Sections 5.1.1 and 5.1.2.

5.1.1 Migrating

Before moving to a new host \( h_{ai} \), the agent determines whether to bring its mailbox. It sends a “MVMB” message to its mailbox at \( f_{A}(h_{ai+1}) \) if it decides to do so. The “MVMB” message contains the address of \( h_{ai} \). The pseudo code of this operation is shown in the function `OnMigration_Agent()`.

```java
OnMigration_Agent() { //executed by agents before moving
    if (\( f_{A}(h_{ai}) = h_{ai} \)) {
        // the agent decides to go with its mailbox
        sendMsgToMailBox("MVMB", \( f_{A}(h_{ai-1}) \), \( h_{ai} \));
        //step (1) in Figure 5.1
    }
    migrateTo(\( h_{ai} \));
    //migrate to the target host-Step (2)’ in Figure 5.1
}
```

On receiving the “MVMB” message, the mailbox executes the function `ProcessMVMBMsg_Mailbox()`. It sends “DEREGISTER” messages to all the hosts on \( Path_{m}(A) \), including the local host (Step (2) in Figure 5.1). After it has collected the “REPLY” messages from all the hosts, it migrates to the destination host (Step (4) in Figure 5.1).
Chapter 5. The Adaptive and Reliable Protocol

Figure 5.1).

\[\text{ProcessMVMBmsg\_MB}\text{(msg)}\{  \text{//executed by mailbox}  \\
\quad \text{for}\text{(every host } h_{m_j}\text{ on } path_{m}(A)) \text{//including the local host}  \\
\quad \text{SendMsgToMAP}\text{("DEREGISTER", } h_{m_j}, f_{h}(h_{a(i-1)})\};  \\
\quad \text{wait until all REPLY msgs from these hosts arrive;}  \\
\quad \text{targetHost} = \text{msg.getConten()};  \\
\quad \text{//get the address of target host, i.e. } h_{a_l}  \\
\quad \text{migrateTo}\text{(targetHost);}  \\
\}\]

On arriving at \( h_{a_l} \), the agent executes the function \( \text{OnArrival\_Agent()} \). It first checks whether \( f_{A}(h_{a}) \) is \( h_{a} \) or not. If not, it does not need to register its new address. Otherwise it sends a “REGISTER” message to every host on \( Path_{a}(A) \) (Step (5) in Figure 5.1).

\[\text{OnArrival\_Agent()}\{  \text{//executed by the agent}  \\
\quad \text{if} \ (f_{A}(h_{a_l})\neq h_{a_l})  \\
\quad \quad \text{return}; \quad \text{//do nothing}  \\
\quad \text{setMBAddress} \ (h_{a_l}); \quad \text{//update the address of its mailbox}  \\
\quad \text{append } h_{a_l} \text{ to } path_{a}(A);  \\
\quad \text{for}\text{(every host } h_{m_j}\text{ on } path_{a}(A))  \\
\quad \quad \text{SendMsgToMAP}\text{("REGISTER", } h_{m_j}, h_{a_l})\;  \\
\}\]

The mobile agent platform (MAP) in a host is responsible for processing all the control messages. Its operation is illustrated in \( \text{MessageProcessing\_MAP()} \) as shown below.

\[\text{MessageProcessing\_MAP}\text{(msg)}\{  \text{//executed by MAP}  \\
\quad \text{switch}(\text{msg.getKind()}\{}  \\
\quad \quad \text{case DEREISTER:}  \\
\quad \quad \quad \text{AddressEntry entry} =  \\
\quad \quad \quad \quad \text{addressTable.getAddr}(\text{msg.getSender()});  \\
\quad \quad \quad \text{entry.VALID} = \text{false};  \\
\quad \quad \quad \text{sendMsgToMailBox}\text{("REPLY", msg.getConten(), null);}  \\
\quad \quad \quad \quad \text{//step (3) in Figure 5.1.}  \\
\quad \quad \text{case REGISTER:}  \\
\quad \quad \quad \text{AgentID sender} = \text{msg.getSender()};  \\
\quad \quad \quad \text{AddressEntry entry} = \text{addressTable.getAddr(sender);}  \\
\quad \quad \quad \text{if} \ (\text{entry} == \text{null})\{}  \\
\quad \quad \quad \quad \text{// REGISTER msg is from the local host, create a}  \\
\quad \quad \quad \quad \text{//new entry in address table for sender’s address.}  \\
\\}
\]
Chapter 5. The Adaptive and Reliable Protocol

entry = new AddressEntry(sender);
insert entry into the local address table;
}
entry.VALID = true;
entry.address = msg.getContent();
while (there are messages in the block queue for sender){
    Message blockedMsg = entry.blockQueue.getNextMsg();
    sendMsgToMAP("AGENTMSG", entry.address, blockedMsg);
    sendMsgToMAP("UPDATE", sender_of_blockedMsg, entry.address);
    // update the address cached by the sender
} // end of while
entry.blockQueue.clear();
} // end of switch

5.1.2 Message Routing

Figure 5.2 illustrates the message forwarding process. Suppose an agent wants to send a message to agent A. Referring to the function SendMessage_Agents(), the sender first checks whether the address of MA has been cached locally. If so, it sends the message to the cached address. Otherwise it sends the message to A’s home host (Step (1) in Figure 5.2).

SendMessage_Agents(Message msg){
    // executed by mobile agent
    if (the receiver’s address is in local cache){
        sendMsgToMAP("AGENTMSG", address in cache, msg);
    } else{
        String homeAddress =receiverID.getHome();
        SendMsgToMAP("AGENTMSG", homeAddress, msg);
    }
}
When a host receives a message destined to agent A, it checks whether $M_A$ is currently resides locally. If so, it inserts the message to $M_A$ directly. Otherwise the message is forwarded to $M_A$’s current address recorded in the local address table. See the function $\text{MessageRouting\_MAP}(\cdot)$ for details.

$$\text{MessageRouting\_MAP}(\text{agentMsg})$$

\[
\begin{align*}
\text{AgentID } & \text{ receiver } = \text{ the target agent of agentMsg;} \\
\text{if } & \text{ (the receiver’s mailbox is local)} \\
& \text{ insert agentMsg to the mailbox;} \\
\text{else} & \\
\text{AddressEntry } & \text{ entry } = \text{ addressTable.getAddress(receiver);} \\
\text{if } & \text{ entry.VALID)} \\
& \text{ sendMsgToMAP(“AGENTMSG”, entry.address, agentMsg);} \\
& \text{ //Step (2) in Figure 5.2 } \\
& \text{ sendMsgToMAP(“UPDATE”, agentMsg.getSender(), entry.address);} \text{ //Step (2)’ in figure 5.2} \\
\text{else} & \text{ //valid tag is false: receiver is migrating} \\
& \text{ entry.blockQueue.insert(agentMsg);} \\
& \text{ //insert the message to the block queue;}
\end{align*}
\]

The $\text{MAP}$ hosting the sender agent caches $M_A$’s new address, which is contained in the incoming “UPDATE” message. Next time when other agents send messages from the same $\text{MAP}$ to A, they will send the message to the new address of $M_A$.

The ARP protocol satisfies the requirements described in Chapter 1. The sender does not need to know the address of the receiver agent. Messages are sent to agent home or to the cached address directly and forwarded to receiver by the agent server. Synchronization of message delivery and migration is used to avoid message loss. The protocol can guarantee that messages are forwarded at most once before they reach the mailbox of the target agent. The location transparency and reliability properties are proved in the next section. Asynchrony is improved because constraint on agent mobility is released as synchronization only occurs between mobile agent systems and mailboxes and the mobile agent can migrate to a new host whenever it
wants without waiting for the messages in transit. Efficiency and adaptability of the protocol are fully discussed in the Section 5.3 and Section 5.4.

5.2 Properties of ARP

Theorem 5.1 shows that ARP can provide location transparency from the point of view of a sender agent.

**Theorem 5.1.** With ARP, a sender agent can send its messages without knowing where the target agent is located.

**Proof.** According to the function “SendMessage-Agent()”, when the sender agent wants to send a message to another, it will check if there is the receiver’s address cached in the underlying MAP. If there is the receiver’s address in the cache, it will send the message to this address without caring about whether it is outdated. Otherwise it will resolve the receiver’s home address from the receiver’s ID and send the message to its home. In both cases, the sender need not specify the current location of the receiver when it wants to send a message. □

The following lemmas and Theorem 5.2 show the effectiveness of ARP, i.e., it can guarantee the delivery of messages. Furthermore, the message will be forwarded at most once to reach the mailbox of the recipient so that it will not chase its recipient.

**Lemma 5.1.** Suppose $M_A$, the mailbox of agent A, is currently located at host $h_{mj}$ and $Path_m(A)$ is $(h_{m0}, h_{m1}, \ldots, h_{mi}, \ldots, h_{mj})$. For all $h_{mi}$ in $Path_m(A)$, if $h_{mi}$ was not down, $M_A$ must have received the REPLY message from $h_{mi}$ before it leaves $h_{mj}$.

**Proof.** If $M_A$ wants to leave $h_{mj}$, it must send DEREGISTER messages to all the hosts in $Path_m(A)$. Since $h_{mi}$ is not down and it is assumed that the underlying location dependent communication mechanisms can shield the network failure, $h_{mi}$ will at last receive the DEREGISTER message and send a REPLY message to $M_A$. According to the function $ProcessMVMBMsg_{MB}()$, $M_A$ cannot leave $h_{mj}$ until it collects all the
REPLY messages from all the hosts in $Path_m(A)$ or when time out. Since we need not worry about network failure, we can conclude that the REPLY message from $h_{mi}$ will reach $M_A$ before $M_A$ leaves. □

Lemma 5.2. For all $h_{mi}$ in $Path_m(A)$, the valid tag of $M_A$’s address in the address table is true only if the address reflects exactly the current location of $M_A$, i.e., the address kept in the address table is not outdated.

Proof. Suppose the address of $M_A$ kept in $h_{mi}$’s address table is $h_{mj}$. If the valid tag is true, from the function MessageProcessing_MAP() we can conclude that $h_{mi}$ must have received $M_A$’s REGISTER message from $h_{mj}$, and the DEREGISTER message has not arrived yet. So the REPLY message has not been sent out from $h_i$. From Lemma 5.1 we know that $M_A$ is still at $h_{mj}$ and cannot leave until it has collected all the REPLY messages from hosts in $Path_m(A)$, including $h_{mi}$. So the address $h_{mj}$ kept in the address table reflects the current location of $M_A$. □

Theorem 5.2. All the messages can be delivered to their recipients’ mailboxes by being forwarded at most once.

Proof. Suppose sender agent $S$ is sending a message $m$ to receiver agent $A$, and $A$’s mailbox $M_A$ is located at host $h_{mj}$. Let $Path_m(A)$ be $(h_{m0}, h_{m1}, ..., h_{mi}, ..., h_{mj})$. Without loss of generality, we assume that $A$’s address kept in the cache of the underlying MAP containing $S$ is $h_{mi}$ and $0 \leq i \leq j$ (if there is no record of $A$’s address in the cache of $S$, the message will be sent to $A$’s home, which is $h_{m0}$ in $Path_m(A)$).

$S$ will obtain $A$’s address, say $h_{mi}$, from the address cache and send the message $m$ directly to $h_{mi}$. When $m$ arrives at $h_{mi}$, 3 cases could happen:

Case 1: $i = j$. The message $m$ will be directly inserted into $M_A$ without being forwarded. No matter where $A$ resides, it can get $m$ from its mailbox later.

Case 2: $i < j$ and $h_{mi}$ has not received $M_A$’s DEREGISTER message from $h_{mj}$. In this case, $m$ will be processed before $M_A$’s DEREGISTER message. To deliver $m$ to $M_A$, $h_{mi}$ will check its address table and find $M_A$’s address is $h_{mj}$ and the valid tag is “true”. 

- 40 -
So $m$ is forwarded to $h_{mj}$. Since $m$ is processed earlier than $M_A$’s DEREGERISTER message, $m$ is forwarded to $h_{mj}$ earlier than the REPLY to the DEREGERISTER message. The FIFO property can guarantee that $m$ arrives at $h_{mj}$ earlier than the REPLY message. From Lemma 5.1, we can conclude that $M_A$ cannot migrate to other hosts during the transmission of $m$. After $m$ arrives at $h_{mj}$, it will be inserted into $M_A$. So $A$ can receive $m$ later from $M_A$ and $m$ is forwarded only once before reaching $M_A$.

**Case 3:** $i < j$ and $h_{mi}$ has received $M_A$’s DEREGERISTER message from $h_{mj}$, i.e., $M_A$ has left for $h_{m(j+1)}$. The host $h_{mi}$ checks the valid tag of $M_A$’s address. If it is “true”, the address of $M_A$ kept in the address table must be $h_{m(j+1)}$ (warranted by Lemma 5.2) and $m$ is forwarded to $M_A$ in the same way discussed in the second case. If the valid tag is “false”, we can conclude that $M_A$ is on its way to $h_{m(j+1)}$. The message $m$ will be put into the blocked message queue. It won’t be forwarded until $M_A$ reaches $h_{m(j+1)}$ and its REGISTER message arrives at $h_{mi}$. After the REGISTER message arrives, $m$ will be forwarded to $h_{m(j+1)}$. As discussed in the second case, $M_A$ will not leave during the transmission of $m$, since $m$ will arrive at $h_{m(j+1)}$ earlier than the REPLY message. Therefore, in this case, $m$ is also forwarded only once and $A$ can get $m$ later from $M_A$.

From the above discussion of all the three cases, we can conclude that all the messages can be delivered to their recipients’ mailboxes by being forwarded at most once. □

### 5.3 Performance Modeling and Evaluation

In this section, we demonstrate the efficiency and adaptability of the ARP protocol, using an analytic model as well as extensive simulation experiments. We first formulate the performance model and then analyze trade-offs of choosing design parameters, such as the time and frequency of mailbox migration, and evaluate the performance of the protocol under different settings with the comparison of variations of existing protocols.
5.3.1 The Performance Model

The performance metrics used for analysis is the total network traffic, which are characterized by the number of messages sent and size of the messages. The following parameters are defined:

- \( t_m \): the message inter-arrival time.
- \( f_m(t) \): the (exponential) probability distribution function of message inter-arrival time.
- \( p(t, n) \): the probability distribution function of the number of messages during time \( t \).
- \( \lambda \): the mean message arrival rate, i.e. \( f_m(t) = \lambda e^{-\lambda t} \), \( p(t, n) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \)
- \( t_r \): the residence time the mobile agent spends in a host.
- \( f_r(t) \): the (exponential) probability distribution function of mobile agent residence time.
- \( 1/\mu \): the mean residence time, i.e. \( f_r(t) = \mu e^{-\mu t} \).
- \( \eta \): the message-to-mobility ratio. It is the expected number of messages per move, i.e. \( \eta = \lambda / \mu \).
- \( p_k \): the probability that the mobile agent will not take its mailbox with it during the \( k^{th} \) migration, i.e. \( p_k = Prob(f_{th}(h_{ak}) = f_{th}(h_{ak+1})) \).
- \( p_{hit} \): the probability that the receiver’s location information cached by the sender is correct. It denotes the probability that the mailbox has not moved since the last message.

We make a conservative assumption that if the agent leaves \( h_{ai} \), it would not re-visit \( h_{ai} \). The impact of this assumption is that the derived \( p_k \) and \( p_{hit} \) will be slightly lower than the true ones, and the total communication cost is a little bit higher.

Suppose that before the \( i^{th} \) migration of the agent, its mailbox resides at host \( h_{mj} \) i.e. \( f_{th}(h_{ai(i-1)}) = h_{mj} \), \( j < i \). The cost of the agent’s \( i^{th} \) migration (denoted by \( C_{mig}(i) \)) and the cost for delivering messages to the agent during its residence at \( h_{ai} \) (denoted by \( C_{delivery}(i) \)) can be formulated as follows:
\[ C_{\text{msg}}(i) = (1 - p_i)(p_{\text{i-1}}C_{\text{mvmb}} + jC_{\text{deregister}} + jC_{\text{reply}} + (j + 1)C_{\text{register}}) + p_i \times 0 \]  

(5.1)

\[ C_{\text{delivery}}(i) = \frac{\lambda}{\mu}(C_{\text{sender}} \rightarrow h_{mk} + (1 - p_{\text{hit}})(C_{h_{mk}} \rightarrow \text{mailbox} + C_{\text{notify}}) + p_{\text{hit}} \times 0 \]  

(5.2)

\[
+ p_i \left( C_{\text{mailbox}} \rightarrow \text{agent} + \alpha C_{\text{request}} \right) + (1 - p_i) \times (0)
\]

where \( h_{mk} \) is the address of the mailbox cached by the sender, and \( \alpha \) is the ratio of the number of query messages to the number of messages obtained from the mailbox. Since there might be no messages or more than one messages when the mailbox receives a query message, \( \alpha \) is not necessarily greater than 1 and its value is dependent on the query frequency of the agent. Because \( f_j(h_{a(i-1)}) = h_{mk} \), we have \( j \) as the number of migrations the mailbox has made before the agent’s \( i^{th} \) migration. The expected number of messages during the agent’s residence time at host \( h_{a(i)} \) is \( \lambda / \mu = \eta \).

We use \( C_{\text{ctrl}} \) and \( C_{\text{msg}} \) to denote communication traffic taken by a control message (e.g. \( C_{\text{mvmb}} \)) and an agent message (e.g. \( C_{\text{mailbox}} \rightarrow \text{agent} \)), respectively. Since control messages, such as “MVMB”, “REGISTER”, and “UPDATE” may be much smaller in size than agent messages, they should not be counted in the same way. Now we have:

\[ C_{\text{msg}}(i) = (1 - p_i)(p_{\text{i-1}}C_{\text{ctrl}} + (3j + 1)C_{\text{ctrl}}) \]  

(5.3)

\[ C_{\text{delivery}}(i) = \eta(C_{\text{msg}} + (1 - p_{\text{hit}})(C_{\text{msg}} + C_{\text{ctrl}}) + p_i(C_{\text{msg}} + \alpha C_{\text{ctrl}})) \]  

(5.4)

From Equations 5.3 and 5.4, we can see that if an agent migrates without taking its mailbox \( \rho_i = 1 \), the migration cost is 0. By deciding the value of \( p_i \) to adjust the migration cost, the ARP protocol works in an adaptive way, where the NM-PL-NS protocol and the FM-\*-SHM protocol become two special examples representing two extreme cases of the ARP protocol. In the NM-PL-NS case, we have \( p_i = 1 \) for all \( i \), and \( p_{\text{hit}} = 1 \). Thus the migration cost is 0, but the message delivery cost is expensive, which is \( \eta(2C_{\text{msg}} + \alpha C_{\text{ctrl}}) \). In the FM-\*-SHM protocols, \( p_i \) is always 0 for all \( i \) and the message delivery is less expensive, but the cost for the \( i^{th} \) migration is given by \( C_{\text{msg}} \) = \( (3j + 1)C_{\text{ctrl}} \).
5.3.2 Impact of Mailbox Migration Frequency on Performance

To reduce the total communication cost, which includes the migration cost and the message delivery cost, compromise must be made between the two extremes according to specific application requirements. In this subsection, we analyze the impact of mailbox migration frequency on the performance of the ARP protocol.

Before the \(i^{th}\) migration, there may be many ways for a mobile agent to decide whether the mailbox should move with its owner agent, i.e. to determine whether \(f_A(h_{ai})\) should be \(f_A(h_{a(i-1)})\) or \(h_{ai}\). The agent could consider factors such as the number of messages it will receive at \(h_{ai}\) and the distance between \(h_{ai}\) and \(f_A(h_{a(i-1)})\). If an agent seldom receives messages from others at \(h_{ai}\), it doesn’t need to take its mailbox to the new host, so \(f_A(h_{ai})\) is set to \(f_A(h_{a(i-1)})\). On the other hand, if an agent expects to receive messages frequently from others and \(h_{ai}\) is far away from \(f_A(h_{a(i-1)})\), it will be expensive to leave the mailbox unmoved and to fetch messages from \(f_A(h_{a(i-1)})\). In this case the agent should migrate to \(h_{ai}\) together with its mailbox, i.e. let \(f_A(h_{ai}) = h_{ai}\). In our analysis, we use a fixed threshold number of messages, \(T\), to decide the value of \(f_A(h_{ai})\). Before the \(i^{th}\) migration, the agent estimates the number of messages it will receive at host \(h_{ai}\) (to simplify the problem, we ignore the differences in the distances between hosts. In the next section, we will show the distance’s effect on the communication overhead). If the number exceeds the threshold \(T\), the agent migrates with its mailbox. Otherwise it moves alone. So we have:

\[
p_k = \text{Prob}(f_A(h_{ak}) = f_A(h_{a(k-1)}))
\]

\[
= \text{Prob(\text{agent A will receive no more than } T \text{ messages at host } h_{ak})}
\]

\[
= \int_{t=0}^{\infty} \left( \sum_{n=0}^{T} p(t,n) f_r(t) dt \right) e^{-\lambda t} dt,
\]

\[
= \left[ \sum_{n=0}^{T} \frac{(\lambda t)^n}{n!} e^{-\lambda t} \right] e^{-\mu t} dt,
\]

\[
= 1 - \left( \frac{\lambda}{\lambda + \mu} \right)^{T+1} = 1 - \left( \frac{\eta}{1 + \eta} \right)^{T+1}.
\]

Since \(p_k\) has the same value for a fixed threshold \(T\) and for every \(k\), we use \(p\) to denote
the probability, i.e. for every $k$ we have $p_k = p = 1 - \left(\frac{\eta}{1 + \eta}\right)^{T+1}$.

We use $t_h$ to denote the residual residence time of the mailbox seen by an arriving message. Because agent migration is assumed to be a Poisson process (the assumption) and the probability that the agent will move its mailbox is $1 - p$, the migration of the mailbox is a stream split from the agent migration with probability $1 - p$. So the migration of the mailbox is also a Poisson process and the residence time is exponentially distributed with the mean value $1/(1 - p)\mu$. Then, from the residual lifetime property [Kle76], the probability distribution function of the residual residence time $t_h$ is exponential, and $f_{t_h} = (1 - p)\mu e^{-(1 - p)\mu t_h}$. So we can represent $p_{hit}$ as:

$$p_{hit} = \text{Prob}(t_c < t_h) = \int_{t_c=0}^{\infty} f_m(t_m) \int_{t_h=t_m}^{\infty} f_h(t_h) dt_h dt_m$$

$$= \int_{t_m=0}^{\infty} \lambda e^{-\lambda t_m} \int_{t_h=t_m}^{\infty} (1 - p)\mu e^{-(1 - p)\mu t_h} dt_h dt_m$$

$$= \frac{\lambda}{\lambda + (1 - p)\mu} = \frac{\eta}{1 - p + \eta}.$$  

Finally, because $f(h_{a(i-1)}) = h_{mj}$, we have $j$ as the number of migrations the mailbox has made before the agent’s $i^{th}$ migration. Since the probability that the mailbox migrates with its owner agent is $1 - p$, we can know that $j$ is binomially distributed and $p(i - 1, j) = C_{i-1}^j (1 - p)^i p^{i-1-j}$. The expectation of $j$ is $(1 - p)(i - 1)$. Now we can rewrite Equations (5.3) and (5.4) as:

$$C_{msg}(i) = (1 - p)(p C_{ctrl} + (3(1 - p)(i - 1) + 1)C_{ctrl})$$

$$C_{delivery} = \eta(C_{msg} + (1 - p_{hit})C_{msg} + C_{ctrl}) + p(C_{msg} + \alpha C_{ctrl})$$

Equations 5.7 and 5.8 are also applicable to the NM-PL-NS and FM-*-SHM protocols, in which $T$ is $+\infty$ and $-1$ respectively. The NM-PL-NS and FM-*-SHM protocols, therefore, are special cases of the ARP protocol. They can also be thought of as
variations of the home server scheme and the forwarding pointer scheme with path compression, respectively.

5.4 Numerical and Simulation Results

To evaluate the performance of the ARP protocol as formulated in Section 5.3 under various conditions, we have conducted numerical analysis and simulation experiments. In the numerical analysis, we use a fixed message number as the threshold \( T \) and show the effect of \( T \) on the migration cost, the message delivery cost and the total cost. In the simulation model, the distance between hosts is taken into account. Performance evaluation results are obtained with the comparisons of the ARP protocol and the NM-PL-NS and FM-\( ^\ast \)-SHM protocols (which are special, extreme cases of the adaptive ARP).

5.4.1 Numerical Results

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>the number of migrations the agent makes during its life cycle</td>
<td>100</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>the ratio between the number of query messages and the number of agent messages got from the mailbox</td>
<td>1.5</td>
</tr>
<tr>
<td>( C_{\text{msg}} )</td>
<td>the communication cost of an inter-agent message</td>
<td>1</td>
</tr>
<tr>
<td>( C_{\text{ctrl}} )</td>
<td>the communication cost of a protocol message</td>
<td>1/4</td>
</tr>
</tbody>
</table>

The setting of parameter values for our numerical analysis is shown in Table 5.1. We differentiate the communication cost of inter-agent messages and protocol messages by their sizes, and ignore the difference in the distances between hosts. We use a fixed threshold value \( T \) through the agent’s life cycle. As discussed in Section 5.3, we can obtain the cost of the NM-PL-NS and FM-\( ^\ast \)-SHM protocols by setting \( T \) to \(+\infty\) and \(-1\) respectively. In fact, when \( T \) is large enough, \( p \) is very close to 1 (see Equation 5.5) and the communication cost is close to the cost under a threshold of \(+\infty\). Therefore, to show the cost of the NM-PL-NS protocol, we only need to show the cost of ARP with a very large value for \( T \). In all the figures shown in the rest of this section, the performance of the two special, extreme-case protocols, namely the NM-PL-NS and
FM*-SHM protocols, are represented by the two end points of the curves, respectively.

Figures 5.3(a) and 5.3(b) show the migration cost and the message delivery cost, respectively, under different threshold \( T \). Curves with different message-to-mobility ratios are shown. We can observe that the migration cost and the message delivery cost vary in opposite directions as the threshold rises (the number of mailbox migration decreases as a result). It is expected that there is an optimal point at which the lowest total cost can be achieved. Figure 5.4(a) shows the total cost, i.e. the sum of the migration cost and the message delivery cost, changing as a function of \( T \). To see the performance more clearly, the X-axis is expressed in terms of \( \lg(T+2) \). We use \( \lg(T+2) \) instead of \( \lg T \) because \( T \), as in Figures 5.3(a) and 5.3(b), starts from \(-1\). The same trick is used in Figure 5.4(b) and Figure 5.6. As expected, there do exist an
optimal threshold at which the total cost is lower than the two extreme cases, although
the advantages over the NM-PL-NS protocol may not be very obvious. We can also
observe that the advantage of the ARP becomes more obvious as the
message-to-mobility ratio, i.e. $\eta$, increases. This is because as $\eta$ increases, the
message delivery cost of the NM-PL-NS protocol increases more quickly than that of
ARP with a proper threshold value.

### 5.4.2 Simulation Results

The numerical results do not show apparently the advantage of the ARP protocol in
efficiency over the centralized NM-PL-NS protocol. However, ARP decentralizes the
role of the agent home as a location-lookup server to avoid the shortcomings of the
NM-PL-NS protocol that the agent home may become the single point of failure and
the performance bottleneck. With this achievement, the efficiency of this distributed
mode is still no worse than the NM-PL-NS protocol. Besides, in the numerical model,
the difference in distances between hosts was ignored, thus the effect of the triangle
routing problem inherent in the NM-PL-NS protocol on the performance was
overlooked. To show the communication overhead in a more realistic situation, we
implemented the ARP protocol in a simulated mobile agent environment. The
distance model is constructed in the simulations. As we can see from the following
simulation results, the ARP protocol works more efficiently than the NM-PL-NS
protocol if we take into account the affect of distance on the communication overhead.
Table 5.2. The assumptions made in our simulation

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>the number of migrations the agent makes during its life cycle</td>
<td>100</td>
</tr>
<tr>
<td>( 1/\mu )</td>
<td>the expectation of the agent’s residence time, which is exponentially distributed</td>
<td>10</td>
</tr>
<tr>
<td>( 1/\lambda )</td>
<td>the expectation of the interval between message arrivals, which is exponentially distributed</td>
<td>( \lambda = \eta \cdot \mu )</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>the ratio the number of query messages to that of agent messages got from the mailbox</td>
<td>1.5</td>
</tr>
<tr>
<td>( \text{Loc}(\text{sender}) )</td>
<td>the coordinates of the sender agent in our distance model</td>
<td>(0, 50)</td>
</tr>
<tr>
<td>( \text{Loc}(h_a) )</td>
<td>the coordinates of the host on ( Path_{\text{receiver}} ) in our distance model</td>
<td>( \text{Loc}(h_a) = (\text{Loc}(h_{a(i-1)}).x + \delta, 0) ) (( i &gt; 0 )) ( \text{Loc}(h_a) = (0, 0) )</td>
</tr>
<tr>
<td>( \delta )</td>
<td>the distance between two consecutive hosts on ( Path_{\text{receiver}} ). ( \delta ) is uniformly distributed from –20 to 20. The sign means the direction.</td>
<td></td>
</tr>
<tr>
<td>( \text{Dist}(h_i, h_j) )</td>
<td>the distance between two arbitrary hosts ( h_i ) and ( h_j )</td>
<td>( \text{Dist}(h_i, h_j) = \sqrt{(h_i.x - h_j.x)^2 + (h_i.y - h_j.y)^2} )</td>
</tr>
<tr>
<td>( d )</td>
<td>the average distance between two arbitrary hosts</td>
<td>40</td>
</tr>
<tr>
<td>( C_{\text{msg}}(h_i, h_j) )</td>
<td>the communication cost of an inter-agent message between hosts ( h_i ) and ( h_j )</td>
<td>( \text{Dist}(h_i, h_j) )</td>
</tr>
<tr>
<td>( C_{\text{ctrl}}(h_i, h_j) )</td>
<td>the communication cost of a protocol message between hosts ( h_i ) and ( h_j )</td>
<td>( \text{Dist}(h_i, h_j)/4 )</td>
</tr>
</tbody>
</table>

Table 5.2 defines the values of the parameters used in the simulation. Figure 5.5(a) illustrates our first distance model, where the sender agent is fixed at the point (0, 50), and the receiver agent is born at point (0, 0) and moves along the x-axis. The distance for the agent’s each migration is uniformly distributed from –20 to 20 (inclusive). The sign represents the direction along the x-axis.

A mobile agent determines whether to migrate its mailbox with it by using both the threshold number \( T \) of messages to be received at the new site and the estimated distance between the current location of the mailbox and the new site. In our
simulation, $T$ is fixed during the agent’s life cycle. Before the agent’s $i^{th}$ migration, it generates a random number representing $\delta$ and counts the distance between $f(A(h_{ai(i-1)})$ and $h_{ai}$. It also predicts the number of messages, denoted as $n_i$, that it will receive at host $h_{ai}$ by generating a random residence time at $h_{ai}$ and a series of messages that will arrive during the residence time. Then it compares $\text{Dist}(f(A(h_{ai(i-1)}), h_{ai}) \times n_i$ and $T \times d$ where $d$ is the average distance between two arbitrary hosts in our distance model. If the former is greater, the agent sets $f(A(h_{ai}))$ to $h_{ai}$, i.e. it takes the mailbox to its target host. Otherwise $f(A(h_{ai}))$ should be $f(A(h_{ai(i-1)})$.

Figure 5.4(b) shows the total communication cost with different values of the threshold $T$ and the message-to-mobility ratio $\eta$. Comparing with Figure 5.4(a), we can see that, with the distance model added, the ARP protocol gains more in performance over the NM-PL-NS protocol.

To show how the triangle-routing problem in the NM-PL-NS protocol affect the performance, we construct another distance model (as shown in Figure 5.5(b)), in which the effect of triangle routing is dramatized. The agent migrates in one direction and $\delta$ is uniformly distributed from 0 to 20. The location of the sender agent is much farther away from the receiver’s home. Figure 5.6 shows the communication costs when the sender is at locations (0, 500) and (1000, 0) respectively. We observe that the more serious the triangle-routing problem is, the more advantages we can gain from the ARP protocol.

Fig. 5.6 the total cost of the second distance model, where the triangle routing problem is dramatized
5.5 Summary

In this chapter a particular protocol derived from the generic framework is described in detail. The protocol, called ARP (Adaptive and Reliable Protocol), represents the JM-PL-SHM combination of the parameters. It satisfies the following requirements presented in Chapter 1.

**Location Transparency and Reliability.** Using ARP, the sender does not need to know the address of the receiver agent. Messages are sent to agent home or to the cached address directly and forwarded to receiver by the agent server. Synchronization of message delivery and migration is used to avoid message loss. The protocol can guarantee that messages are forwarded *at most once* before they reach the mailbox of the target agent. The location transparency and reliability properties are proved in Section 5.2 as Theorem 5.1 and Theorem 5.2 respectively.

**Asynchrony.** Asynchrony is improved in ARP because 1) with the address caching mechanism, the reliance on the agent home as the location server and message forwarding server is reduced; 2) the constraint on agent mobility is released as synchronization only occurs between mobile agent systems and mailboxes and the mobile agent can migrate to a new host whenever it wants without waiting for the messages in transit.

**Efficiency and Adaptability.** The efficiency and adaptability of ARP have been illustrated in Section 5.2 and Section 5.4. By separating the mailbox from its owner agent, the mailbox-based scheme also introduces great flexibility and adaptability. To see this, design trade-offs and performance of the ARP protocol are analyzed in these two sections using both an analytical model and simulations. By properly deciding the migration frequency of the mailbox, the protocol can reduce the costs of both “migration” and “message delivery”.

The impact of mailbox migration frequency on the communication cost is also modeled and analyzed in this chapter. Although in the context of ARP, the conclusion can be applied to other protocols derived from our generic framework using JM mode.
The model and analysis can also be used as a guide for users to properly decide the mailbox migration frequency according to the migration and communication pattern of mobile agents in particular applications, so that a better balance between the “migration” cost and the “message delivery” cost can be achieved and the total communication overhead can be minimized.
Chapter 6. Optimization and Fault-Tolerance Issues

6.1 Path Compression and Garbage Collection

In the ARP protocol, the mailbox of the agent $A$, denoted as $M_A$, has to deregister and register with all the hosts in $S_m(A)$ (see Definition 2) for each migration. As $S_m(A)$ becomes larger, the migration overhead will increase. However, since the address cache maintained by the host is updated when the host sends a message to an outdated address of $M_A$, it is likely that each host tends to refer to the latest host in $Path_m(A)$ as the current address of $M_A$ unless it has not sent messages to $M_A$ for a long time. Hosts in the front of $Path_m(A)$ may no longer be referred to by other hosts. These hosts can be removed from $Path_m(A)$ to reduce the migration overhead of $M_A$.

Another problem is the management of the address cache maintained in each host. If a local agent $S$ at the host sends messages to agent $A$, the host will cache the physical address of $M_A$. However, when $S$ leaves the host, the address of $M_A$ in the cache may no longer be used by other agents. Some garbage collection mechanism should be used to remove useless addresses maintained in a host’s cache.

In this section we propose an algorithm to shorten the migration path of the mailbox. The algorithm can also be used to remove the useless address maintained in a host’s cache.

6.1.1 The Path Compression Algorithm

We first define the terminology, which will be used in the following discussions.

**Definition 4.** Let $H$ denote the set of all the hosts in the network and let $A$ denote a mobile agent. The function $R_A: H \rightarrow S_m(A)$ is defined as follows:
\[
R_A(h_s) = \begin{cases} 
    h_{mi} & h_{mi} \in S_m(A) \text{ and } h_{mi} \text{ is cached by host } h_s \text{ as the current address of } M_A \\
    h_{mi0} & \text{otherwise}
\end{cases}
\]

**Definition 5.** The set \( S_R(A, h_{mi}) = \{ h_s \mid R_A(h_s) = h_{mi} \} \) denotes the set of all the hosts referring to \( h_{mi} \) as the current address of \( M_A \).

**Definition 6.** For each host \( h_{mi} \in S_m(A) \) and \( i > 0 \), \( h_{mi} \) is called a **redundant host in** \( S_m(A) \) if it will no longer receive any messages destined to agent \( A \) unless \( M_A \) revisits it.

By definition, we know the redundant host can be safely removed from \( S_m(A) \) without affecting the reliability of message delivery. The objective of our path compression algorithm is to identify redundant hosts and to remove them from \( S_m(A) \). We extend our protocol in the following way:

1) Each agent address in the cache of host \( h_s \) is associated with a timer which is initially set to 0 and starts as soon as the address is added to cache. Let the timeout value for the timer be \( \text{TTL} \). When the timer reaches \( \text{TTL} \), the address is removed from the cache. By function \( \text{SendMessage\_Agents()} \), when an agent on this host wants to send a new message to \( A \) and finds that \( A \)'s address is not in the cache, it will send the message to \( A \)'s home. Each time the address is accessed (updated or used by a sender agent as the current address of the receiver agent), the timer is reset to 0.

2) Each host \( h_{mi} \in S_m(A) \) maintains a reference table \( T(A) \) for agent \( A \). The table \( h_{mi}.T(A) \) contains a set of addresses and a “\text{closed}” tag. Later we will see by **Lemma 6.2** that \( h_{mi}.T(A) \) maintains addresses of hosts in the set \( S_R(A, h_{mi}) \). The table is created when \( M_A \) visits \( h_{mi} \) if there no \( T(A) \) exists. On creation, the address set is empty and the \text{closed} tag is set to “false”. 
3) On receiving $M_A$’s “REGISTER” message from $h_{mj}$ ($i\neq j$), $h_{mi}$ checks the reference table $T(A)$. If $T(A).\text{closed}$ is “true”, nothing is done. Otherwise, $h_{mi}$ sets $T(A).\text{closed}$ to “true” and starts the timer associated with $T(A)$. Both $T(A)$ and the record of agent A in the address table are removed from $h_{mi}$ as soon as $T(A)$ becomes empty or the timer reaches $\text{TTL} + \text{MTL}$, where $\text{MTL}$ is the Maximum message Transmission Latency of the network.

4) When $h_{mi}$ receives a message destined to A from sender agent S which is residing at host $h_s$, it checks $T(A).\text{closed}$. If $T(A).\text{closed}$ is “false” and $h_s$ has not been in $T(A)$, $h_s$ is added to $T(A)$. If $T(A).\text{closed}$ is “true”, $h_s$ is removed from $T(A)$.

5) When $h_{mi}$ receives $M_A$’s “DEREGISTER” message from $h_{mj}$ ($i\neq j$) and finds that the record of agent A in the address table has been removed, it sends an “NAK” message, instead of the “REPLY” message, to $h_{mj}$. Upon receiving the “NAK” message, $M_A$ removes $h_{mi}$ from its migration path $Path_m(A)$.

### 6.1.2 Correctness of the Algorithm

Here we present an informal proof of the effectiveness of the algorithm. Firstly we present two basic assumptions. Later we will show that with a little revision, the algorithm can also work even without these assumptions, although the performance may be degraded.

**Assumption1.** The message transmission latency of the network is no larger than $\text{MTL}$. Each host can effectively obtain the value of $\text{MTL}$.

**Assumption2.** The interval that a host sends two consecutive messages destined to the same receiver agent is no less than $2\text{MTL}$.

**Lemma 6.1.** For any host $h_{mi} \in S_m(A)$ and $i > 0$, $h_{mi}$ will not receive any messages to agent A from $h_s$ after $h_s$ is removed from $h_{mi}, T(A)$. 

- 55 -
Proof. According to the extended protocol, \( h_s \) is removed from \( h_{mi}.T(A) \) only if \( h_{mi} \) receives a message sent to agent \( A \) from \( h_s \) and the closed tag of \( h_{mi}.T(A) \) is “true”. By the function \texttt{MessageRouting\_MAP()}\( h_{mi} \) will send an “UPDATE” message to \( h_s \) informing it of the new address of \( M_A \). Since we assume the interval that \( h_s \) sends two consecutive messages to \( A \) is no less than 2MTL, \( h_s \) must have received the “UPDATE” message from \( h_{mi} \) and updated its cache before sending the next message. In other words, \( R_A(h_s) \) must have changed to the new address of \( M_A \) when \( h_s \) sends the next message to agent \( A \). \( \Box \)

**Lemma 6.2.** For all hosts \( h_s \in S_R(A, h_{mi}) \), either \( h_s \) has been in the table \( h_{mi}.T(A) \), or it will be added to \( h_{mi}.T(A) \) within MTL.

**Proof.** Suppose \( M_A \) is residing at \( h_{mi} \) and \( R_A(h_s) = h_{mj} \) \((i \neq j)\). The message destined to agent \( A \) will be sent to \( h_{mj} \) by \( h_s \). According to our protocol, \( h_{mj} \) will forward the message to \( h_{mi} \) and send an “UPDATE” message to \( h_s \). Upon receiving the “UPDATE” message, \( h_s \) will update its address cache and let \( R_A(h_s) = h_{mi} \) (therefore we have \( h_s \in S_R(A, h_{mi}) \)). After \( h_{mi} \) receives the forwarded message from \( h_{mj} \), it will add \( h_s \) to table \( h_{mi}.T(A) \). If the arrival of the agent message at \( h_{mi} \) is earlier than the arrival of the “UPDATE” message at \( h_s \), \( h_s \) would have been added to \( h_{mi}.T(A) \) before it updates its cache. Otherwise, since the “UPDATE” message and the agent message are sent out by \( h_{mj} \) almost at the same time and the transmission time of each message is no larger than MTL, we can easily reach the conclusion that \( h_s \) will be added to \( h_{mi}.T(A) \) within MTL after \( R_A(h_s) \) turns to \( h_{mi} \), i.e. \( h_s \in S_R(A, h_{mi}) \). \( \Box \)

**Lemma 6.3.** There won’t be any new hosts join the set \( S_R(A, h_{mi}) \) after the closed tag of \( h_{mi}.T(A) \) turns to true.

**Proof.** The closed tag of \( h_{mi}.T(A) \) turning to true implies that \( M_A \) has left \( h_{mi} \) for \( h_{m(i+1)} \) and \( h_{mi} \) has received the “REGISTER” message of \( M_A \) from \( h_{m(i+1)} \). Then we know that all the hosts \( h_{mj} \) in \( S_m(A) \) must have received the “DEREGISTER” message of \( M_A \).
sent from $h_{mi}$ (warranted by Lemma 5.1). By the function MessageProcessing_MAP(), we know either $M_A$’s address maintained in the address table of $h_{mj}$ has been updated ($h_{mj}$ has also received the “REGISTER” message of $M_A$ from $h_{mi(i+1)}$), or the valid tag of $M_A$’s address in the address table of $h_{mj}$ is “false”. In neither case will $h_{mj}$ return to any message sender an “UPDATE” message containing $h_{mi}$ as the current address of $M_A$ (see the function MessageRouting_MAP()). Therefore there will not be new hosts join $S_{g(A, h_{mi})}$.

**Theorem 6.1.** $h_{mi}$ is a redundant host in $S_m(A)$ if the closed tag of $h_{mi}$.T(A) is true and $h_{mi}$.T(A) is empty.

**Proof.** According to Lemma 6.2 and Lemma 6.3, if the closed tag of $h_{mi}$.T(A) is “true” and $h_{mi}$.T(A) is empty, we have $S_{g(A, h_{mi})} = \emptyset$. From Lemma 6.1 and Definition 5, we know $h_{mi}$ will no longer receive any messages destined to agent $A$ unless $M_A$ revisits it. Therefore $h_{mi}$ is a redundant host in $S_m(A)$. $\square$

**Theorem 6.2.** $h_{mi}$ is a redundant host in $S_m(A)$ if the timer of $h_{mi}$.T(A) reaches TTL + MTL.

**Proof.** If the timer of $h_{mi}$.T(A) has reached TTL + MTL, we know the closed tag of $h_{mi}$.T(A) is “true”. That’s because the timer is started only after the closed tag of $h_{mi}$.T(A) is set to “true”. If $h_{mi}$.T(A) is empty, we have proved that $h_{mi}$ is a redundant host (Theorem 6.1). Otherwise, let’s suppose $h_s$ is in $h_{mi}$.T(A). According to our protocol, we know that $h_{mi}$ has not receives any messages to agent $A$ from $h_s$ since the closed tag of $h_{mi}$.T(A) has turned to “true”. This implies that:

1) The address of $M_A$ cached by $h_s$, if has not been cleared, is $h_{mi}$ (i.e. $R_h(h_s) = h_{mi}$) and it has not been updated since the closed tag of $h_{mi}$.T(A) turns to “true”.

2) The address of $M_A$ cached by $h_s$ has not been read for a period of TTL since the closed tag of $h_{mi}$.T(A) turns to “true”. That’s because the address of $M_A$ cached by
$h_s$ is read only if there are messages sent from $h_s$ to agent A. Since the period of TTL + MTL has passed and the transmission time of a message is less than MTL, we know there have been no messages sent from $h_s$ to agent A for, at least, a period of TTL.

From 1) and 2) we know the physical address of $M_A$ in the cache of $h_s$ cannot has been accessed (updated or read) for at least TTL seconds. Therefore $h_s$ must have removed the address of $M_A$ from its cache. According to the definition we have $h_s \notin S_{M}(A, h_m)$ and $h_s$ can be safely removed from $h_{m_i}T(A)$. In this way we can safely empty $h_{m_i}T(A)$. By Theorem 6.1 we know that $h_{m_i}$ is a redundant host.

From above proof, we can see that Theorem 6.1 and Theorem 6.2 depend on Assumption 1 and Assumption 2. Without these assumptions, messages to agent A may arrive at $h_{m_i}$ even after $h_{m_i}$ has been removed from $S_{m}(A)$. In this case we let $h_{m_i}$ forward the message to $h_{m_0}$, i.e. the home of agent A. Since $h_{m_0}$ always holds the physical address of $M_A$, it can forward the message to $M_A$. Although messages may be forwarded once more to reach the target agent and the workload of A’s home is increased, the reliability of message delivery is maintained in the extended protocol.

Since an address in the address cache is removed if it is not accessed within TTL, the protocol also provides a way to clear the useless address maintained in the address cache of each host. If an agent wants to communicate with another agent whose address has been cleared from the cache prematurely, the message is sent to the agent home. The expense of the garbage collection is that the workload of the agent home may be increased.

The value of TTL affects the performance of the extended protocol greatly. If TTL is very large, the probability that a sender cannot find a receiver’s address in the cache is little and there is little increment of the workload of the receiver’s home. However, the redundant host in the migration path of the receiver’s mailbox may not be removed in time and we cannot get much optimization of the migration cost. On the
other hand, with a small TTL, the migration cost can be greatly reduced by path compression, but more messages must be forwarded by the receiver’s home. There are two extreme cases about the value of TTL. One extreme case is that TTL is greater than the life cycle of the receiver agent. Then there would be no path compression happening during the life cycle of the receiver agent and the optimized ARP works the same way with ARP. Another extreme is that TTL is set to 0. In this case all messages are forwarded by the receiver’s home and the mailbox of the receiver only needs to keep its home in its migration path. In this case the process of message delivery between the message sender and the mailbox of the receiver is identical with the NM-PS-SHM protocol mentioned in Chapter 4.

6.1.3 Performance Evaluation

6.1.3.1 The Simulation Model

In this Section, we compare the performance of the optimized protocol with that of ARP. Firstly we present definitions of parameters used in our simulation model.

- The receiver is denoted as the agent A with the mailbox $M_A$.
- $S$: the set of senders in the network that might send messages to agent A.
- $\forall i, 1 \leq i \leq 99, x_i = \begin{cases} 1 & f_d(h_{ai}) = h_{ai} \text{(see Definition 3)} \\ 0 & \text{Otherwise} \end{cases}$
- $t_s$: the intervals that the sender $s_i$ sends two consecutive messages to agent A. In our simulation $t_s$ is exponentially distributed with the expectation of $1/\lambda$.
- $t_r$: the residence time mobile agents (both senders and the receiver) spend in a host. $t_r$ is exponentially distributed with the expectation of $1/\mu$.

By removing redundant hosts the optimized protocol can shorten the migration path maintained by the mailbox. To differentiate the actual migration path of the mailbox that is denoted as $S_{m}(A)$ and the path maintained by the mailbox, in which the redundant hosts have been removed, we use $S_{p}(i)$ to denote the set of hosts maintained in the migration path of $M_A$ after the $i^{th}$ migration of agent A. Thus we have $S_{p}(0)=h_{a0}$. 

- 59 -
In ARP we have either \( S_p(i) = S_p(i-1) \) \( (f_A(h_{ai}) = f(h_{ai-1})) \) or \( S_p(i) = S_p(i-1) \cup \{h_a\} \) \( (f_A(h_a) = h_a) \), where \( i \geq 0 \). In the optimized protocol, we have either \( S_p(i) = S_p(i-1) \) \( (f_A(h_{ai}) = f(h_{ai-1})) \) or \( S_p(i) = S_p(i-1) \cup \{h_{ai}\} - R(i) \) \( (f_A(h_a) = h_{ai}) \), where \( i \geq 0 \) and \( R(i) \) denotes the set of redundant hosts identified before the \( i^{th} \) migration of agent A.

In our simulation model, we let \( H = \{h_0, h_1, \ldots, h_{99}\} \) (see Definition 4) and \( S = \{s_0, s_1, \ldots, s_9\} \), where \( s_0, s_1, \ldots, s_9 \) are all mobile agents in the system. The receiver agent A migrates sequentially from host \( h_0 \) to host \( h_{99} \). The agent A uses a threshold value of \( T (0 \leq T \leq 1) \) to determine whether to take \( M_A \) to the target host. Before each migration, agent A will compare \( T \) with a randomly generated number \( p \), which uniformly distributed over \([0, 1]\). If \( T \) is greater than \( p \), agent A will migrate with \( M_A \); otherwise it will move alone. Therefore \( T \) is the probability that agent A will move with its mailbox. \( T \) remains unchanged during the life cycle of agent A. Senders in \( S \) are randomly distributed on hosts in \( H \). For each migration they choose their target host randomly from these 100 hosts.

As we can see from Section 6.1.1, the main differences between the optimized protocol and ARP concentrate on the process of migration of \( M_A \) and message delivery between senders and \( M_A \). Interactions between the agent A and its mailbox \( M_A \) are the same in these two versions of ARP. Hence to compare these two protocols in terms of communication overhead, we only consider in our simulation the migration of \( M_A \) and the message delivery process between senders and \( M_A \).

If the agent A takes \( M_A \) to its target host in the \( i^{th} \) migration, the migration overhead involves the transmission cost of Deregister messages, ACK messages, Register messages; otherwise the migration overhead is zero. The overhead of \( i^{th} \) migration of the agent A can be denoted as:

\[
C_{mig}(i) = x_i((1-x_{i-1})C_{ctrl}+2(|S_p(i-1)|-1)C_{ctrl}+(|S_p(i)|-1)C_{ctrl})
\]  

(6.1)

Equation 6.1 is applicable to both ARP and the optimized version. The first term in the outer parenthesis denotes the cost of the \( MVMB \) message. The second term denotes
the cost of Deregister and Reply messages sent before the $i^{th}$ migration of the agent A. The last term is the cost of register messages after the $i^{th}$ migration. The total migration overhead during the life cycle of the agent A is given by:

$$C_{mig} = \sum_{i=1}^{99} C_{mig}(i)$$

(6.2)

While the agent A is residing at $h_i$, the cost of message delivery between the sender $s_j$ and $M_A$ involves the cost of message passing from $s_j$ to the cached address (or the home of agent A), and the cost of message forwarding and the UPDATE message if cache miss occurs; if the forwarding host has been removed prematurely as a redundant host, the cost of once more message forwarding must be added. Thus the delivery cost is given by:

$$C_{del}(i, j) = n_{i, j} \left( C_{msg} + (1 - p_{hit})((1 - p_a) C_{msg} + C_{msg} + C_{ctrl}) \right)$$

(6.3)

where $n_{i,j}$ denotes the number of messages would send to the agent A while the agent A is residing at $h_i$; $p_{hit}$ denotes the probability that the $M_A$’s location information cached by the underlying host of the sender is correct; $p_a$ denotes the probability that the forwarding host has not been prematurely removed from the migration path of $M_A$. In ARP $p_a$ is always 1. In the optimized ARP, we have proved that $p_a$ is always 1 if Assumption 1 and 2 are satisfied. Otherwise $p_a$ may be less than 1. The total message delivery cost is given by:

$$C_{del} = \sum_{i=1}^{99} \sum_{j=0}^{9} C_{del}(i, j)$$

(6.4)

The total communication cost is given by:

$$C_{total} = C_{mig} + C_{del}$$

(6.5)

where $C_{mig}$ and $C_{del}$ are given by Equation 6.2 and 6.4.
6.1.3.2 Simulation Results and Performance Analysis

In our simulation, we ignore the transmission delay of messages and agents and assume that MTL is zero; therefore Assumption 1 and 2 are satisfied in our simulation environment. As in Chapter 5, we differentiate the communication cost of inter-agent messages and protocol messages by their sizes, and ignore the difference in the distances between hosts. Costs of an agent message and a control message are set to 4 and 1, respectively.

![Fig. 6.1(a)](image1.png) The total migration cost of optimized ARP under different TTL and mailbox migration frequency.

![Fig. 6.1(b)](image2.png) The total message delivery cost of optimized ARP under different TTL and mailbox migration frequency.

Figure 6.1(a) and 6.1(b) show the migration overhead of agent A and the message delivery cost between senders and \(M_A\), respectively, in the optimized ARP. Curves with different mailbox migration frequency are illustrated. The value of TTL varies from 0 to 1000. To see the performance more clearly, the X-axis is expressed in terms of \(\lg(TTL+1)\). We use \(\lg(TTL+1)\) instead of \(\lg T\) because \(T\) starts from \(-1\). The same trick is used in Figure 6.2.

The migration and message delivery cost of ARP are not shown separately in above figures. However, as discussed in Section 6.1.2, the optimized ARP works in the same way with ARP when TTL is larger than the life cycle of the receiver agent. This property is verified by our simulation experiments because both the migration overhead and the message delivery cost keep unchanged when TTL is large enough.
The right end of each curve represents the performance of ARP. As shown in Figure 6.1(a), the agent migration cost can be reduced using the path compression algorithm. The migration cost is minimized when TTL is 0 because in this case all messages are forwarded by the receiver’s home and the mailbox of the receiver only needs to keep its home in its migration path, i.e. \(|S_p(i)|\) is always 1. The migration cost increases with the increase of TTL. The upper bound is the migration cost of ARP.

However, the reduction of migration cost is at the expense of the message delivery cost. From Figure 6.1(b) we can see that the message delivery is large when TTL is small because more messages have to be forwarded by the agent home. We can also draw this conclusion from Equation 6.3 because the more quickly the cache is cleared, the smaller \(p_{hit}\) will be. When TTL gets larger, the sender agents can take full advantage of the cached address of \(M_A\) and \(p_{hit}\) will get larger.

Figure 6.2 illustrates the total communication cost, i.e. the sum of the total migration cost and the total message delivery cost. As we can see, in most cases the communication cost of ARP, represented by the right end of curves, can be reduced by properly setting the value of TTL.

### 6.2 Fault Tolerance Issues

Throughout the discussion of our generic framework and the ARP protocol, the failures of hosts and channels are ignored to concentrate on problems caused by agent mobility. However, these failures do exist in the real world and the separation of the mailbox from the agent increases ARP’s vulnerability to these failures. In this section
a fault-tolerant version of ARP is proposed to make the ARP keep working in presence of these failures.

### 6.2.1 Failure Model

In our model all hosts are connected by the network and can communicate with one another by sending messages on the network.

- **Hosts**: Each host is subject to fail-stop failure [Sch83, Sch84] and no failure is permanent, i.e. all encountered failures are transient. The failure causes the host to halt and to lose its internal volatile states. However, the stable storage survives failures and data can be recovered and accessed upon recovery of the host.

- **Network**: The network may lose or duplicate messages, or delivery them out of order. We assume there is no network partition and the network delivers only uncorrupted messages.

- **Other assumptions**: We assume hosts holding message senders and the receiver will not fail during message transmission. We also assume the agent and the mailbox will not get lost or corrupted during migration, i.e. the underlying mobile agent platform can provide exact-once agent and mailbox migration.

As mentioned in Chapter 3, there is an MAP running on each host. Each MAP provides two levels of message passing primitives, i.e. high-level location-independent primitives and low-level location-dependent primitives. In previous discussion we have assumed that the low-level primitives have abstract away the network failure and provided FIFO order of point-to-point message passing between MAPs. The implementation of the low-level primitives is presented in the following Section 6.2.2. Based on these low-level primitives, the ARP protocol is revised in Section 6.2.3 to tolerant failures of hosts.
6.2.2 Implementation of the Low Level Primitives

Low-level primitives provide reliable in-order point-to-point message passing between MAPs. These primitives are implemented in a TCP-like way but work in the application level. The reason why we do not use existing TCP directly is that only reliable message delivery between hosts is not sufficient. Taking hosts failures into consideration, although TCP messages may arrive at target host reliably, the target host may crash before the message is delivered to the MAP. In this case the sender would have no idea of what packets have been delivered to the target MAP. In this section the terms of “sender” and “receiver” denote communicating MAPs instead of mobile agents.

Like TCP, the implementation of low-level primitives is based on the sliding window mechanism. Each message is labeled with a sequence number. The sender transmits the message and buffers it in the outgoing window until the arrival of an acknowledgement. If the acknowledgement does not arrive within the sender’s expected round trip time, the message is retransmitted. Upon reception, the message is inserted into the appropriate of the receiver's incoming window, or discarded as a duplicate if it repeats some messages already in the window or has a sequence number too small for the window.

The low-level primitives can be regarded as an implementation of the TCP protocol at the application level. However, the outgoing window is stored in stable storage. The receiver does not send an acknowledgement until the message in the incoming window has been stored in the stable storage.

The low-level primitives can also be used for detecting failure of hosts. If the sender has not received acknowledgement from the receiver after several retransmission, the sender can regard the receiver as unreachable and throw exceptions to be handled by upper-level primitives.
6.2.3 The Fault-Tolerant ARP

In this section the ARP protocol is improved to tolerate host failures. High-level primitives implement the fault-tolerant ARP on top of these low-level primitives, which screens faults of point-to-point message passing between MAPs. Therefore the high-level primitives can overcome message loss due to agent mobility, host failures and network faults, i.e. it can accomplish reliable end-to-end message delivery for mobile agents in the real sense.

To overcome host failures, ARP should be extended to accomplish following tasks:

(1) **Failure Detection.** As mentioned above, the low-level message passing primitives can be used for detecting failure of hosts. If the sender has not received acknowledgement from the receiver after several retransmission, it will know the receiver host has crashed. Moreover, Deregister messages and query messages in ARP can be used to detect the failure of the receiver host. If the sender cannot get the response messages (Reply messages and agent messages, respectively) after a long enough period of time, it’ll know the receiver host has been crashed.

(2) **Handling the failure of the host holding the mailbox.** If the host holding the mailbox fails, the ARP can not keep working because a) other hosts on the migration path of the mailbox can no longer forward messages to the mailbox; b) The owner agent of the mailbox can no longer pull messages from the mailbox; d) Messages buffered in the mailbox are lost or temporarily unavailable. Techniques to handle the failure of the mailbox will be presented in Section 6.2.3.1.

(3) **Handling the failure of other hosts on the migration path of the mailbox.** The failure of a host on the migration path of the mailbox will affect the ARP in the following way: a) if the address of the failed host is in the cache of a sender
host as the current address of the receiver, the sender cannot send messages to the receiver agent; b) messages blocked at this host may be lost; c) the property that the message will be forwarded at most once before it arrives at the target mailbox can no longer be satisfied. The host may receive agent messages after recovery. However, the current address of the receiver’s mailbox maintained by the host may be outdated because it may have missed some deregister and register messages before the recovery. As a result it may forward the message to the outdated address and cause one more message forwarding. Techniques to handle the failure of the mailbox will be presented in Section 6.2.3.2.

6.2.3.1 Handling the failure of the host holding the mailbox

Following techniques can be used to overcome the failure of the mailbox.

1) **Creation of a new mailbox.** In the fault-tolerant version of ARP, the mobile agent also keeps the migration path of its mailbox. The agent queries its mailbox for messages periodically and the query message is used to detect the failure of the mailbox. If the agent detects that the mailbox is unreachable, it will create a new mailbox locally and send **Register** messages to all the hosts on the migration path of the old mailbox. The failed host containing the old mailbox is also in the path. Hosts’ processing of the **Register** message is the same with ARP. In the fault-tolerant version of ARP, agents’ mailboxes are stored in stable storage of each host.

2) **Block of message forwarding.** If a host on the migration path of the mailbox wants to forward messages to the mailbox and finds that the mailbox is unreachable, it blocks the message forwarding. The message is inserted into the block queue of the corresponding receiver and the valid tag of the receiver’s address is set to false. The address table on each host is also stored in stable storage, which includes addresses of mailboxes that have ever passed it, corresponding valid tags and blocked message queues.
3) **Recovery of the mailbox.** Since no failure is permanent in our failure model, the failed host holding the mailbox will recover later. The mailbox on it can also be recovered because it is stored in the stable storage. On recovery, the mailbox needs to do nothing. If it receives the query message from its owner agent, it knows there is no new mailbox generated during its failure. It will keep working as if it has never failed before. If the host receives the **Register** or **Deregister** message from a newly generated mailbox, it forwards the old mailbox to the new one before processing the message. At the side of the new mailbox, these two mailboxes will be merged into one so that messages buffered in the old mailbox will not be lost.

### 6.2.3.2 Handling the failure of hosts on the migration path of the mailbox.

The ARP protocol is revised in the following way to overcome the failure of hosts on the migration path of the mailbox.

1) **Revision of the sender host.** In ARP each host caches the address of receiver agent's mailbox if there have been agents on the host sending messages to the receiver agent. Messages to the receiver agent are sent directly to the cached address. However, if a sender host, say $h_s$, caches the address of a failed host and sends a message to it, it will find that the host is unreachable and there is nowhere else to send the message. To handle this problem, we require that the sender host maintain out-dated addresses in its cache in addition to the latest address. For instance, suppose the address of the mailbox in $h_s$’s cache is $h_{m1}$. Later $h_s$ receives an **Update** message of the mailbox from $h_{m2}$. The address of $h_{m2}$ is added to cache as the current address of the mailbox. However, $h_{m1}$ is not removed from the cache. As a result $h_s$ may maintain a list of addresses in its cache as the current address of the receiver. The first address should be $h_{m0}$, i.e. the home of the receiver agent. When $h_s$ finds that $h_{m2}$ becomes unreachable, it will remove $h_{m2}$ from its cache and send messages to $h_{m1}$. If all the hosts in the cache of $h_s$ are unreachable, $h_s$ has to multicast the message to find the mailbox of the receiver.
2) Revision of the mailbox. Since hosts on the migration path of the mailbox may crash, the mailbox does not need to collect all the Reply messages from these hosts before migration. After waiting a long enough period of time the mailbox will move to the target host even if it has not collect all the Reply messages. The hosts whose Reply messages are not received will be regarded as failed hosts. On arrival at the target host, the mailbox will send Register messages to all the hosts on its migration path, including these “failed hosts” that are identified before migration.

3) Recovery of hosts. On recovery, the host recovers the address table stored in its stable storage. It sets all “valid” tags in the address table to false until it receives Register messages from corresponding mailboxes.

The fault-tolerant ARP can keep working in presence of failure of hosts. Messages will not be lost due to failure of hosts or migration of target hosts.

6.3 Summary

In this Chapter the ARP protocol is further improved to work more efficiently and robustly. Firstly, a path-compression algorithm is proposed to remove the redundant host in the migration path of the mailbox. Properties of the algorithm are proved. Simulation experiments are conducted to compare the performance of ARP and its optimized version using the path-compression algorithm. Results show that, by properly setting the value of TTL, the migration cost of the mobile agent can be greatly reduced using the path-compression algorithm. This algorithm can also be used for garbage collection, i.e. clearing the useless address of mobile agents cached by hosts in the network.

Secondly, a fault-tolerant version of ARP is presented to handle the crash of hosts on the agent migration path. In preceding chapters failures of network and hosts are ignored. In this Chapter the fault-tolerance issues are discussed so that the ARP
protocol can work more robustly. Similar to TCP, implementation of low-level message passing primitives of MAP is proposed, which not only provides reliable in-order point-to-point message passing between MAPs, but also can be used for detection of the failure of hosts. ARP is also revised to keep working in presence of failure of hosts on the mailbox migration path. Implemented on the basis of low-level primitives, the fault-tolerant ARP can overcome message loss due to agent mobility, host failures and network faults, i.e. it can accomplish reliable end-to-end message delivery for mobile agents in the real sense.
Chapter 7 Interaction between Mobile Agent and Its Mailbox: Push or Pull

7.1 The Push and Pull Approaches

As discussed in Chapter 4, there are two approaches for the mobile agent to get messages from its mailbox, namely push and pull. Since the push and pull modes are different in terms of network traffic only when the mobile agent and its mailbox reside at different hosts (i.e. \( f_A(h_a) \neq h_a \)), in the following discussion we assume the mailbox migrates under the NM mode. This assumption is convenient for us to focus on the comparison of the push and pull modes. For the same reason, we only investigate the interaction between the receiver agent and its mailbox and do not consider the message passing between the sender and the receiver’s mailbox because the latter is the same in these two modes.

7.1.1 The Push Approach

In the push mode, the mailbox maintains the location information of its owner agent. Incoming messages are pushed to the current address of the agent. After each migration, the mobile agent registers its new location with its mailbox. Here the
registration message plays the role of a subscribe message. Subsequent incoming messages are pushed to the agent’s new address.

The simple push mode mentioned above cannot guarantee reliable message delivery. When a message is pushed to the address kept in the mailbox, the owner agent may have left for another host. Although it can be further forwarded, the message may keep chasing the target agent. To avoid message loss or chasing problems caused by agent mobility, synchronization between the message pushing from the mailbox and the migration of its owner agent should be used (SMA). As described in Chapter 4, SMA is implemented in the following way. Before migration, the agent deregisters its current location with its mailbox and waits for the ACK message. After it receives the ACK message the agent migrates to the new location and registers its new location with the mailbox after arrival. As shown in Figure 4.1, messages can be pushed to the mobile agent when it is in “stationary” and “waiting” states and must be blocked when it is in the “moving” state. The effectiveness of the synchronization has been proved in Section 5.

7.1.2 The Pull Approach

In the pull mode, the mailbox buffers the messages to its owner agent and does not need to keep its location information. The mobile agent queries its mailbox periodically for messages. Receiving the query message, the mailbox forwards one buffered message to the agent. If there is no message in the mailbox, a “null” message is sent to the agent as a reply.

Either a synchronous or an asynchronous query operation can be implemented by the agent. Synchronous query means the agent suspends its execution after issuing a query until it receives the reply from its mailbox. In this way the agent can ensure that there won’t be any messages forwarded to it during its migration. If asynchronous query is used, the agent can continue its execution after a query. To avoid message loss, the agent cannot migrate to other hosts until all the reply arrives.
The agent always knows the location of its mailbox and initiates the request for messages, so location registration is unnecessary in the pull mode. Since the agent won’t leave for next host without receiving the response to its query, there is no message loss or chasing problems in the pull mode.

7.1.3 Properties of Push and Pull

In what follows, the properties of push and pull and their pros and cons are analyzed in term of the reliability, resiliency to failures, constraint on the agent mobility, support of real-time processing, communication overhead and flexibility.

**Reliability.** By reliability we mean the messages can be routed to its target agent within a bounded number of hops. As discussed above, either message loss or the chasing problem may happen under the simple push mode without synchronization. The synchronized push mode can avoid these problems and thus guarantee reliable message delivery. In the rest of this paper we only concern about the synchronized push mode. The term push and synchronized push are used interchangeably and both of them denote the synchronized push mode. In the pull mode, since the receiver agent takes the initiative to request messages from its mailbox, the agent can ensure that there won’t be any messages forwarded to it during its migration. Therefore the requirement of reliable message delivery can easily be satisfied.

**Resiliency to failures.** In the push-based approach the location and status (e.g. stationary, moving, and waiting shown in Figure 4.1) of the agent must be maintained at its mailbox during the agent’s life cycle. The state of the agent is lost if the host containing the mailbox fails. After recovery, the mailbox may have lost the trace of the agent. Moreover, the agent cannot detect the failure of its mailbox and re-register with it until its next migration. In contrast, the pull-based mailbox is resilient to failures due to its stateless nature. Besides, querying periodically, the agent can easily detect the failure of its mailbox (see Chapter 6).
**Constraint on the agent mobility.** In the synchronized push mode, the agent has to deregister with its mailbox and wait for the ACK message before its migration, therefore the agent mobility is constrained and the migration time is increased. In the pull mode, if synchronized query operation is used, the agent can leave for next host as soon as it finishes its execution at this host, but the execution time is increased. For asynchronous query, the agent also has to wait for the arrival of all the response of its query before migration. However, by deciding the time and number of queries, the agent can flexibly reduce the constraint on its migration.

**Support of real time message processing.** In the push mode, unless the agent is in “moving” status, messages are forwarded to their target agents immediately after they reach the mailbox. The sender has greater certainty that the message will reach its target within an appropriate timeframe. However, in the pull mode, the transmission time of a message depends not only on the network delay, but also on the frequency at which the receiver queries its mailbox. Therefore the delayed time for the message getting processed by the receiver is longer in the pull mode.

**Communication overheads.** In the push mode, three extra messages, namely Deregister, ACK and Register, are needed for each agent migration in the push mode. In the cases the agent migrates frequently but seldom communicates, the communication overhead of the push mode is significant. On the other hand, two messages are needed in the pull mode for each query, i.e. the query message and the response from the mailbox. Moreover, to decrease the delay of message processing, the receiver may query at a higher frequency than the frequency of the message arrival at the mailbox. Therefore the pull-based approach is liable to impose a larger load on the network. Communication overheads of these two modes depend on the migration and communication pattern of the mobile agent.

**Flexibility.** Since the agent has the autonomy to decide the time and frequency of the queries for messages, more flexibility is introduced in the pull mode. For example, the agent can adjust its query frequency dynamically. If it is in urgent need of information
from its coordinator, it may query at a higher frequency. Otherwise a lower frequency is adopted. Distance can be another factor of concern. If the current location of the agent is very far from its mailbox, it can query the mailbox at a much lower frequency or do not query at all. When it migrates to a nearer host to its mailbox, it can query more frequently and process more messages buffered in the mailbox.

### 7.2 Performance Analysis of Push and Pull

In this section we evaluate the performance of the push and pull mode in terms of network traffic and delay of message processing. Simulations are performed for the evaluation study. Results of the simulation are presented and discussed.

Firstly we define the following parameters (Definitions of some parameters, such as $n$, $t_m$, $\lambda$, $t_r$, $1/\mu$, and $\eta$, are the same as those in Chapter 5. They are repeated here just for clarity):

- $n$: the number of agent migrations
- $t_m$: the message inter-arrival time
- $\lambda$: the mean message arrival rate, i.e. the expected the number of messages that arrive within one unit of time.
- $t_r$: the residence time the mobile agent spends in a host
- $1/\mu$: the expectation of the agent’s residence time at a host
- $\eta$: the message to migration ratio, i.e. $\eta = \lambda \mu$
- $\tau$: the ratio of the query frequency of the agent to the arrival rate of the messages ($\tau > 1$)
- $t_d(i)$: the time when message $m_i$ arrives at the mailbox
- $t_s(i)$: the time that message $m_i$ is sent from the mailbox to the target agent

As mentioned above, to compare the network traffic of the push and pull modes in the mailbox-based model, we only need to concern the communication cost between the agent and its mailbox, since the costs of the message transmission between senders and the mailbox are the same for these two modes. In the push mode, the
communication cost involves the transmission cost of Deregister messages, ACK messages, Register messages and all the agent messages that the mailbox forwards to its owner. Hence the total communication cost of the push mode during the agent’s life cycle is given by:

\[ C_{\text{push}} = n(C_{\text{deregister}} + C_{\text{ack}} + C_{\text{register}}) + n\eta*C_{\text{msg}} \]  

(7.1)

where \( C_{\text{deregister}} \), \( C_{\text{ack}} \), and \( C_{\text{register}} \) denote the transmission cost of one piece of Deregister message, ACK message, and Register message respectively; \( C_{\text{msg}} \) denotes the transmission cost of one piece of agent messages.

In the pull mode, the query frequency is proportional to the message arrival rate and is given by \( \tau \lambda \). The communication cost involves the cost of query messages, agent messages sent by the mailbox as responses, and “null” responses. It is given by:

\[ C_{\text{pull}} = n(\tau \lambda *C_{\text{query}} + (\tau - 1) \eta*C_{\text{null-reply}} + \eta*C_{\text{msg}}) \]  

(7.2)

where \( C_{\text{query}} \), \( C_{\text{null-reply}} \), and \( C_{\text{msg}} \) denote the transmission cost of one piece of query message, agent messages sent by the mailbox as response, and “null” response, respectively.

In our simulation, the communication cost is characterized by the number of messages sent, size of the messages and the distance traveled by the messages. The interval

Fig. 7.1 the mobility and distance model used in our simulation
between message arrival at the mailbox, i.e. $t_m$, and the residence time the agent spends at a host, i.e. $t_r$, are exponentially distributed with the expectation of $\frac{1}{\lambda}$ and $\frac{1}{\mu}$, respectively. The transmission delay of messages and agents are proportional to their size and the distance traveled by them. The mobility model of the agent is shown in Figure 7.1. The distance between the mailbox and the agent is uniformly distributed over $[0, 100]$ and the angle $\theta$ is uniformly distributed over $[0, 2\pi]$. Table 7.1 shows the assumptions and parameters used in our simulation.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{\text{msg}}(h_i \leftrightarrow h_j)$</td>
<td>$\text{Distance}(h_i, h_j)$</td>
<td>Communication cost of an agent message from host $h_i$ to host $h_j$.</td>
</tr>
<tr>
<td>$C_{\text{deregister}}(h_i \leftrightarrow h_j)$, $C_{\text{register}}$, $C_{\text{query}}$, $C_{\text{null-reply}}$</td>
<td>$\text{Distance}(h_i, h_j)/4$</td>
<td>Communication cost of a control message, which are smaller in size ($\frac{1}{4}$ of that of the agent message).</td>
</tr>
<tr>
<td>$T_{\text{msg}}(h_i \leftrightarrow h_j)$ (in milliseconds)</td>
<td>$\text{Distance}(h_i, h_j)$</td>
<td>Transmission time of a single agent message from host $h_i$ to host $h_j$, which is proportional to the distance and message size.</td>
</tr>
<tr>
<td>$T_{\text{ctrl}}(h_i \leftrightarrow h_j)$ (in milliseconds)</td>
<td>$\text{Distance}(h_i, h_j)/4$</td>
<td>Transmission time of a single control message, e.g. register message, query message, etc.</td>
</tr>
<tr>
<td>$T_{\text{agent}}(h_i \leftrightarrow h_j)$ (in milliseconds)</td>
<td>$2\text{Distance}(h_i, h_j)$</td>
<td>Transmission time of the agent, which is larger in size.</td>
</tr>
<tr>
<td>$\text{Distance}(h_i, h_j)$</td>
<td>$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$</td>
<td>The distance between host $h_i$ and host $h_j$, which is set to the geometric distance in the x-y plane.</td>
</tr>
<tr>
<td>$\lambda\mu$</td>
<td>5 seconds</td>
<td>Mean residence time</td>
</tr>
<tr>
<td>$n$</td>
<td>100</td>
<td>The number of agent migrations</td>
</tr>
</tbody>
</table>

The total communication cost is shown in Figure 7.2(a). We can see that when the agent migrates frequently but receives few messages, i.e. when the message-to-mobility ratio is low, the performance of the pull mode get ahead of the push mode. That’s easy to understand because in the push mode the agent has to register and deregister its location for every migration, no matter whether it will receive messages at the target host. However, when the number of messages received at each host increases, the overhead of query messages in the pull mode outweighs that of the register and deregister messages in the push mode.
We use the average delay of message processing to evaluate the support of real time message processing in the push and pull mode. Suppose there are totally $m$ messages forwarded from the mailbox, the average delay of message processing is given by:

$$D = \frac{\sum_{i=1}^{m} (t_{i}(i) - t_{a}(i))}{m}$$

(7.3)

In our simulation we assume that the workload of the mailbox is very light and the query requirement of the agent is responded immediately after its arrival at the mailbox. In the push mode, incoming agent messages are also processed by the mailbox as soon as their arrival. Unless the target agent is in “moving” status, messages are forwarded to it without delay. Thus our simulation result of the delay of message processing is the lower bound of the real value.

Figure 7.2(b) illustrates the delay of message processing in the push and pull modes, with different message to mobility ratio and query frequency. We can observe that the delay of message processing of the pull mode is much larger than that of the push mode.
7.3 Improvements of the Pull Based Approach

Simulation results presented in Section 7.2 show that the pull mode outperforms the push mode only when the message to mobility ratio is very low, i.e. the agent migrates frequently but seldom exchange information with others. Moreover, the pull mode will introduce much more delay on processing of messages by the receiver than the push mode. However, as we discussed in Section 7.1.3, the pull mode can introduce more flexibility and the message processing delay and communication overhead can be further reduced. In this section, we propose two improvements to the pull-based approach, namely greedy pull and distance based pull. To differentiate it from the proposed versions, the pull mode discussed in Section 7.1 is referred to as simple pull in the rest of this paper.

7.3.1 Greedy Pull

In the simple pull mode discussed in Section 7.1, the agent queries the mailbox once for one message only. Even though there may be many messages in the buffer, the mailbox forwards only one message as the response of one query from the agent. In this approach the agent has more autonomy and flexibility to process the messages, since it can decide the exact number of messages forwarded from the mailbox. This advantage will be shown in Section 7.3.2. The constraint on the agent mobility is also lower in this method because for each response at most one message is forwarded from the mailbox. The agent does not need to wait too long for the arrival of the response before its migration. However, if the agent needs to process all the messages buffered by the mailbox, it has to query the mailbox more frequently than necessary. To get all the incoming messages, the ratio of the query frequency to the message arrival rate, namely $\tau$, must be greater than 1. It’s a waste of bandwidth and the delay of message processing is large.

There is another way for the mailbox to process the query messages from the agent, i.e. it forwards all the buffered messages, if any, in a batch to the agent as the
response. We call this approach *greedy pull* because the agent requires all the messages in one query. The greedy pull is a hybrid method of the push and pull mode because messages may be forwarded to the agent without its explicit solicitation and beyond its expectation. In this mode the agent can query the mailbox at a much lower frequency and the ratio $\tau$ can be much lower than 1 as long as the agent is not in urgent requirement of the messages.

![Figure 7.3(a)](image1)
![Figure 7.3(b)](image2)

**Fig. 7.3(a)** Comparison of the communication cost under the push, simple pull and greedy pull modes

**Fig. 7.3(b)** Comparison of delay of message processing under simple pull and greedy pull modes

Figure 7.3(a) and 7.3(b) shows the performance comparison of the simple pull and the greedy pull modes. The assumption and parameter setting of our simulation are almost the same with that in Section 3, except that if $m$ ($m \geq 1$) messages are forwarded to the agent in a batch, the communication cost is $m^{*}\text{dist}$ and the transmission delay is $m^{*}\text{dist}$ milliseconds. We can see that with the same query frequency, the communication cost of the greedy pull is very close to that of the simple pull mode. The former is a little larger, that’s because more “null” responses are sent in the greedy pull mode. However, since the ratio $\tau$ can be lower than 1, the communication cost of the greedy pull can be greatly reduced by using a small query frequency. Moreover, with the same query frequency, the delay of message processing in the greedy pull mode is much lower. We can observe that, in the greedy pull mode, the average delay with the ratio of 0.5 and 1 are approximate to those of the simple pull mode with the ratio 1.5 and 2.0, respectively. Therefore, with the same tolerance of
delay of message processing, the communication cost of the greedy pull mode is much lower than that of the simple pull mode.

### 7.3.2 Distance-based Pull

To suit for different mobility patterns better, many adaptive algorithms are proposed in the field of personal communication, including timer-based, movement-based, distance-based and state-based location-updating algorithms [Bar95, Won00]. In these algorithms, mobile users decide whether to update their location information according to different merits. Like mobile users, the mobile agent is also an autonomous object and can estimate the number of messages it will receive and decide whether and when to process them. The pull mode provides flexibility for the agent to make these decisions. The agent can query the mailbox at various frequencies which can be adjusted dynamically. In this section we propose an adaptive distance based pull algorithm, in which the agent adjusts its query frequency according to the distance between it and the mailbox.

In the distance based pull algorithm, the query frequency is decided by the distance between the agent and the mailbox and the message-to-mobility ratio. If the agent resides at a host far away from the mailbox, it will reduce the query frequency; hence fewer messages will be forwarded from the mailbox. After it moves nearer to the mailbox, the agent queries more frequently and processes more messages buffered at the mailbox. To control the number of messages forwarded from the mailbox, the agent requires one message in one query, as in the simple pull mode.

The agent mobility model and the distance model in our simulation are the same with those defined in Section 7.2. An empirical formula is used in our simulation to decide the query frequency, which is given by:

\[
    f = \tau \lambda = 2(1 - \text{dist}(a, r)/\text{max\_distance})\lambda \tag{7.4}
\]
where the \( \text{dist}(a, r) \) is the estimated distance between the agent and the mailbox; \( \text{max\_distance} \) is the maximal possible value of the distance, which is set to 100 in our distance model. Thus the ratio of the query frequency to the message arrival rate, denoted by \( \tau \), is totally decided by the distance and varies over \([0, 2]\).

The communication cost of the distance based pull algorithm is illustrated in Figure 7.4(a). As we can see, the communication cost is greatly reduced in the adaptive algorithm, which is even lower than the push-based algorithm. However, the expense of the performance improvement is the sacrifice of the real-time support of message processing. Since most messages are buffered at the mailbox and processed only when the agent moved to a host near the mailbox, the delay of message processing is increased, as shown in Figure 7.4(b).

Besides the distance, urgency of the receiver’s requirement of messages can be another factor to be concerned in designing adaptive pull algorithm. The receiver agent can adjust the query frequency according to its requirement of real-time message processing in addition to distance. If it is in urgent use of some information from its communication partner, the agent can increase the query frequency. Otherwise a lower frequency can be used. Since the urgency of the requirement is too
application specific, it is not modeled in our simulation and users can take it into account in particular applications.

### 7.4 Trade-offs between Push and Pull

Push and pull are two canonical techniques for web data dissemination. In [Deo01] the author showed that the push-based approach performs better than the pull mode in terms of number of messages. However, it is not necessarily true in the mobile environment, where the receiver keeps moving and changing its address. In the push mode, since the mobile agent has to register its address on arrival at a new host and synchronize the migration and message forwarding, the communication overhead is substantial if the message-to-mobility ratio is low. On the other hand, since the receiver has more autonomy in the pull mode to decide the number of messages to be received from the mailbox, more flexibility is introduced and adaptive algorithm can be designed to reduce the communication overhead.

From the results of our simulation experiments on the push and pull based algorithms and the two variant of the pull mode, we observe that the push mode is suitable to communication intensive applications, where the message-to-mobility ratio is high and the agent needs real time processing of messages. However, if the agent migrates frequently and low constraint on the agent mobility is preferred, user can choose the pull mode. According to the particular requirement of applications, greedy pull or distance based pull can be chosen for lower delay of message processing or communication cost.

Table 7.2 summarizes our discussion and gives a comparison of the synchronized push, simple pull, greedy pull and distance based pull approaches. Since the push and pull modes have complementary properties, applications with different requirements, such as fault resiliency, constraint on agent mobility, support of real time processing, communication overhead and flexibility, can choose different communication modes.
Table 7.2. Properties of the message delivery modes

<table>
<thead>
<tr>
<th></th>
<th>Reliability</th>
<th>Resiliency</th>
<th>Mobility</th>
<th>Support of real time processing</th>
<th>Communication overhead</th>
<th>Flexibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync Push</td>
<td>Yes</td>
<td>Low</td>
<td>High</td>
<td>Strong</td>
<td>Depends on the Message-to-Mobility Ratio</td>
<td>Low</td>
</tr>
<tr>
<td>Pull</td>
<td>Yes</td>
<td>High</td>
<td>Low</td>
<td>Weak</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Greedy Pull</td>
<td>Yes</td>
<td>High</td>
<td>Low</td>
<td>Weak</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>Distance Based Pull</td>
<td>Yes</td>
<td>High</td>
<td>Low</td>
<td>Weak</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>

For a better balance of communication cost and delay of message processing, a combination of the push and pull algorithm can be used. That is, the agent can switch between the pull and push modes. If currently a push mode is used and the agent wants to switch to the pull mode at the next host, it does not register its new address with the mailbox after its arrival at the host. If messages are needed, the agent queries the mailbox as in the pull mode. Without the register message, the status of the agent kept at the mailbox is always “moving” and incoming messages are blocked at the mailbox. Messages will not be forwarded to the agent unless the mailbox receives query messages or register message. If the agent wants to switch to push mode, it just sends a register message to the mailbox, which acts as a subscribe message. The mailbox changes the status of the agent to “stationary” and resumes push of messages to the agent. The switch between push and pull can be decided by the number of messages the agent needs at next host, the distance between the agent and the mailbox, and the urgency of the requirement.

7.5 Summary

In this chapter we explore the two possible approaches, namely synchronized push and pull, for the mobile agent to reliably obtain messages from its mailbox. We show that the push and pull modes have complementary properties in terms of agent mobility constraint, communication overhead, support of real time message processing, mailbox’s resiliency to failures, and flexibility.
From the results of our simulation experiments on the push and pull based algorithms and the two variant of the pull mode, we conclude that the push mode is suitable to communication intensive applications, where the message-to-mobility ratio is high and the agent needs real time processing of messages. However, if the agent migrates frequently and low constraint on the agent mobility is preferred, user can choose the pull mode. According to the particular requirement of applications, greedy pull or distance based pull can be chosen for lower delay of message processing or communication cost. For a better balance of communication cost and delay of message processing, urgency of the requirement of messages can be concerned and a combination of the push and pull algorithm can be used in particular applications.
Chapter 8. Conclusion and Future Work

8.1 Conclusion

Communication is an essential ability for mobile agents to collaborate with others by information exchanging and knowledge sharing, however, mobility brings new challenges to inter-agent message passing. In this thesis, we first analyze the necessity of remote message passing in mobile agent systems and present requirements of message delivery protocols for mobile agents. We believe mobile agent message delivery protocols should satisfy the requirements of location transparency, reliability, asynchrony, efficiency and adaptability.

After a comprehensive review of related work, a very general framework has been proposed for designing message delivery protocols in mobile agent systems. The framework uses a flexible and adaptive mailbox-based scheme, which associate each mobile agent with a mailbox while allowing the decoupling between them. With different combination of the mailbox migration frequency, the message delivery mode between the mailbox and the agent, and the synchronization mode between message forwarding and migration, the framework not only covers several previously known location management strategies, such as home-server scheme and forwarding-pointer scheme, but also provides the designer with the possibility to define protocols better suited for specific applications. It has the following advantages: (1) it can be used to describe and evaluate various mobile agent communication protocols; (2) it can help users to clearly specify their requirements; (3) it can help users design a flexible, adaptive protocol which can be customized to meet their requirements.

One such protocol derived from this framework, namely ARP, is presented in detail. Using ARP, the sender does not need to know the address of the receiver agent. Messages are sent to the agent home or to the cached address directly and forwarded
to receiver by the agent server. Synchronization of message delivery and migration is used to avoid message loss. The protocol can guarantee that messages are forwarded \textit{at most once} before they reach the mailbox of the target agent. Asynchrony is improved in two aspects. Firstly, with the address caching mechanism, the reliance on the agent home for agent tracking and message forwarding is reduced. Secondly, the constraint on agent mobility is released because synchronization only occurs between mobile agent systems and mailboxes and the mobile agent can migrate to a new host whenever it wants without waiting for the messages in transit. By separating the mailbox from its owner agent, the mailbox-based scheme also introduces great flexibility and adaptability. To see this, design trade-offs and performance of the ARP protocol are analyzed using both an analytical model and simulations. We conclude that by properly deciding the migration frequency of the mailbox, the protocol can be designed to reduce the costs of both “migration” and “message delivery”. The model and analysis can be used as a guide for users to properly decide the mailbox migration frequency according to the migration and communication pattern of mobile agents in particular applications, so that a better balance between the “migration” cost and the “message delivery” cost can be achieved and the total communication overhead can be minimized. The analysis and conclusion of the impact of mailbox migration frequency on the communication cost can also be applied to other protocols derived from our generic framework using JM mode.

The ARP protocol is further improved to work more efficiently and robustly. Firstly, a path-compression algorithm is proposed to remove the redundant host in the migration path of the mailbox. Simulation results show that, by using the path-compression algorithm, the migration cost of the mobile agent can be greatly reduced. This algorithm can also be used for garbage collection, i.e. clearing the useless address of mobile agents cached by hosts in the network. Secondly, a fault-tolerant version of ARP is presented to handle the crash of hosts on the agent migration path. Implemented on the basis of low-level message passing primitives, which provide reliable in-order point-to-point message passing between MAPs, the
fault-tolerant ARP can overcome message loss due to agent mobility, host failures and network faults, i.e. it can accomplish reliable end-to-end message delivery for mobile agents in the real sense.

*Push* and *pull* are alternatives on the dimension of *mailbox-to-agent message delivery* in our three-dimensional model. Pros and cons of these two modes in the mobile environment are fully discussed and simulation results are presented to compare them in terms of communication overheads and delay of message processing. Two improved version of the basic pull approach, namely greedy pull and distance-based pull, are proposed to reduce delay of message processing and network traffic, respectively. From the results of our simulation experiments on the push and pull based algorithms and the two variant of the pull mode, we conclude that the push mode is suitable to communication intensive applications, where the message-to-mobility ratio is high and the agent needs real time processing of messages. However, if the agent migrates frequently and low constraint on the agent mobility is preferred, users can choose the pull mode. According to the particular requirement of applications, greedy pull or distance based pull can be chosen for lower delay of message processing or communication cost. For a better balance of communication cost and delay of message processing, urgency of the requirement of messages can be concerned and a combination of push and pull algorithms can be used in particular applications.

**8.2 Directions of Future Work**

**8.2.1 Modeling Mobility of Mobile Agents**

To evaluate the performance of the mobile agent message delivery protocol either by numerical models or by simulation experiments, it is essential to model the migration pattern of the mobile agents. The model should illustrate the network topology, the range of agents’ migration, and distance (which might be expressed in number of hops, round trip time, available bandwidth or anything else) between the source and target
hosts of mobile agents’ migration. However, it is a challenging task to design a general mobility model for mobile agents. Although there have been many mobility models proposed in the research field of personal communication service [Won00], the mobility model of the mobile user is quite different from the mobile agent. Since the speed of people’s mobility is constrained by the speed of the transportation vehicles, the mobile user has to pass through adjacent cells during his migration and the mobility is restricted in a limited area within a fixed timeframe. Things are different for the mobile agent, which is a software object transmitted through computer network. The migration pattern can be very different in different applications. The scope of migration also varies greatly from the local network to the Internet scope. In this paper we attempt to use a uniformly distributed distance model to characterize the agent mobility (see Chapter 5 and 7). With the wide application of mobile agent technology, the mobile agent migration pattern can be generalized and more realistic model can be designed. On the other hand, for specific applications, more refined migration model can be defined for performance modeling and analysis.

8.2.2 Research on Mobile Agent Multicast

Multicast is a group communication mechanism which is to send a message to a group of processes. In mobile agent systems, mobile agents are often grouped to perform some operations cooperatively; therefore mobile agent multicast is another essential communication mechanism in mobile agent systems.

In [Jia99], the author classified multicast requirements of different applications along four dimensions. Each describes a distinct feature of multicast protocols.

1) Delivery atomicity: refers to the feature that either the specified number of group members receive the message safely or none at all.

2) Delivery ordering: specifies the desired order in which multicast messages should be delivered to group members. There are several degrees of ordering of messages in group communication. A weaker degree of ordering is a subset of a stronger degree. The degrees of orderings are arbitrary ordering, FIFO ordering, causal
ordering and total ordering.

3) Real-time delivery: specifies the time constraints within which a multicast session should complete.

4) Fault-tolerance: specifies the capability that a multicast service can continue correctly in the presence of failures.

There has been a wide range of multicast mechanisms developed to meet the need of various distributed applications in last decade. However, the presence of mobility makes it more difficult to satisfy above requirements in mobile agent systems. For instance, mobility makes it more difficult to maintain message delivery ordering and real-time delivery. One direction of our future research is to expand our mailbox-based scheme for mobile agent multicast. The basic idea can be that a group of mobile agents share a common mailbox dedicated for multicast. However, special algorithms should be designed to overcome difficulties caused by agent mobility and to satisfy requirements mentioned above.
Appendix. Publications and Projects

Bibliography


- 91 -


Appendix. Publications and Projects


Delivery Scheme for Mobile Agents”, Proc. of 7\textsuperscript{th} IEEE Intl. Workshops on
Enabling Technologies: Infrastructure for Collaborative Enterprises 1998 (WET
ICE’98).

[Lin94] Y. Lin, “Determining the User Locations for Personal Communications Services
Networks”, IEEE Transactions on Vehicular Technology, Vol. 43, No. 3, August
1994, 466-473.

L. Rohermel and F. Hohl, (eds), Mobile Agents: 2\textsuperscript{nd} Int. Workshop MA’98, LNCS

[Mor99] L. Moreau, “Distributed directory service and message router for mobile agents”,

141-150.


for Java Applications”, Transactions of Information Processing Society of Japan,
1(1), 1999

Internet”, Computer Networks and ISDN Systems, 27(3), Elsevier Science,


Appendix. Publications and Projects


Appendix. Publications and Projects


