

## Path Pruning in Mailbox-based Mobile Agent Communications

JIANNONG CAO, LIANG ZHANG, XINYU FENG<sup>+</sup> AND SAJAL K. DAS<sup>++</sup>

*Department of Computing  
The Hong Kong Polytechnic University  
Hung Hom, Kowloon, Hong Kong  
E-mail: {csjcao, cszhangl}@comp.polyu.edu.hk*

*<sup>+</sup>Department of Computer Science  
Yale University  
New Haven, Connecticut, U.S.A.  
E-mail: xinyu.feng@yale.edu*

*<sup>++</sup>Department of Computer Science and Engineering  
University of Texas at Arlington  
Arlington, Texas, U.S.A.  
E-mail: das@cse.uta.edu*

This paper is concerned with the design of efficient algorithms for mobile agent communications. We first describe a novel mailbox-based framework for flexible and adaptive message delivery in mobile agent systems and a specific adaptive protocol derived from the framework. Then we present the design and verification of a path pruning and garbage collection algorithm for improving the performance of the proposed protocol. Simulation results show that by properly setting some parameters, the algorithm can effectively reduce both the number of location registrations and the communication overhead for each registration. Consequently, the total location registration overhead during the life cycle of a mobile agent is greatly reduced. The algorithm can also be used for clearing useless addresses cached by hosts in the network.

**Keywords:** mobile agent, communication protocol, mailbox, path pruning, garbage collection

### 1. INTRODUCTION

Mobile agent technology is often described as the future of distributed computing. It promises to offer a unified and scalable framework for such applications in widely distributed heterogeneous open networks as e-commerce, information retrieval, process coordination, mobile computing and network management [1]. A mobile agent is in essence a program that is able to move autonomously around the network during its execution to finish the tasks assigned by its owner.

In various situations mobile agents need to communicate with each other. Remote inter-agent communication is thus a fundamental facility in mobile agent systems. Al-

---

Received October 15, 2003; accepted November 15, 2003.

Communicated by Ten-Hwang Lai, P. Sadayappan, Yu-Chee Tseng and Yi-Bing Lin.

though process communication has been a cliché in research on distributed systems, agent mobility raises a number of new challenges in the design of effective and efficient message delivery mechanisms for mobile agent systems. These are described as follows.

**Location Transparency:** Since a mobile agent has autonomy to move from host to host, it is unreasonable, if not impossible, to require a mobile agent to have a priori knowledge about its communication peers' locations before sending 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 is, therefore, required to keep track of the locations of all the mobile agents within the system.

**Reliability:** 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 this paper we do not deal with the fault-tolerant issues and all our discussions are based on the assumption of a fault-free network. (Actually, we have in [2] built up a fault-tolerant architecture to deal with both point-to-point and end-to-end reliability.) However, even an ideal fault-free transport mechanism is not sufficient to ensure successful message delivery [3]. The asynchronous nature of message passing and agent migration may cause the loss of messages during the agent's migration.

**Asynchrony:** Here, asynchrony includes two aspects – asynchronous migration and asynchronous execution of mobile agents. First, although coordination of message forwarding and agent migration are necessary to guarantee reliable message delivery, agent mobility should not be over-constrained by frequent and tight synchronization. Second, since supporting disconnected operation is regarded as an important advantage of the mobile agent paradigm [1], the agent's ability of disconnected execution should not be restricted by relying heavily on the agent home for delivering every message to the agent. It is desirable that the protocol can possess both asynchronous migration and asynchronous execution so that no reduction in the merits of mobile agent technology will be introduced.

**Efficiency:** The cost of a protocol is characterized by the number of messages sent, the 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: (i) *migration* that facilitates the movement of a mobile agent to a new site, and (ii) *delivery* that locates a specific agent and delivers a message to it. The objective of minimizing the overhead of these two operations results in conflicting requirements [4]. In general, a protocol should perform well for some specific communication and migration patterns, achieving a balance of the tradeoff between the costs of migration and delivery.

**Adaptability:** Different applications may have different requirements and thus a different emphasis on the above issues. In some applications, asynchrony is favored and,

therefore, the agent home should not be relied on as the sole location server. In other applications, reliability is more important and so synchronization is needed. Different inter-agent communication and agent migration patterns may also have different effects on the migration and delivery costs. 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 is suitable for as many applications as possible.

In this paper we first describe a generic framework for the design of mobile agent communication protocols. The framework uses a flexible and adaptive mailbox-based scheme which associates each mobile agent with a mailbox while allowing decoupling between them. Based on this framework, we derive a protocol that satisfies all of the above requirements. Then we focus on the design of a path pruning and garbage collection algorithm for further improving the performance of the proposed protocol. Simulation results show that by properly setting some parameters, the algorithm can effectively reduce the total location registration overhead during the life cycle of a mobile agent.

The remaining part of this paper is organized as follows. Section 2 presents the mailbox-based framework. Section 3 proposes the adaptive protocol derived from the framework. Section 4 introduces the path pruning algorithm. Section 5 gives the performance analysis. Section 6 describes some related works. The final section provides the concluding remarks.

## 2. MAILBOX-BASED FRAMEWORK

In this section we describe a generic mailbox-based framework for the design of flexible and adaptive mobile agent communication protocols. The framework uses a flexible and adaptive mailbox-based scheme which associates each mobile agent with a mailbox. As shown in Fig. 1, incoming messages to the receiver are inserted into the mailbox first (step 1) and later received from the mailbox with either a pull or push operation (steps 2 and 3). The mailbox 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

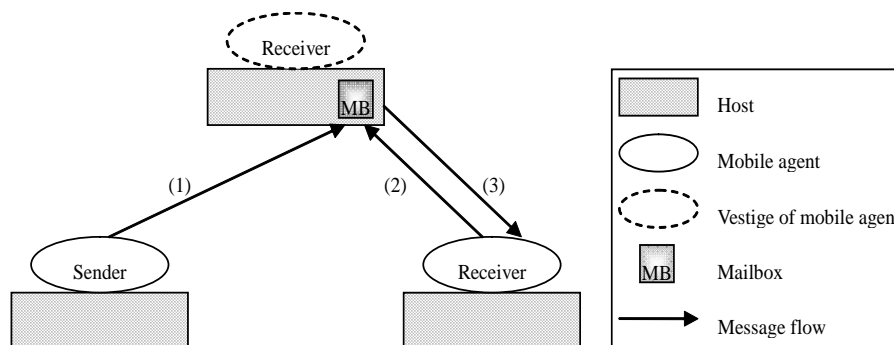


Fig. 1. The mailbox-based scheme.

migration path. Before each migration, the mobile agent determines whether or not to take its mailbox to the new site according to some criteria. The migration paths of the mobile agent and the mailbox, as well as their relationship are defined as follows.

**Definition 1** The migration path of a mobile agent  $A$ , denoted  $Path_a(A)$ , is an ordered list of hosts  $(h_{a0}, h_{a1}, \dots, h_{an})$  that  $A$  has visited in sequence, where  $h_{a0}$  is  $A$ 's home. The set of hosts on the path is denoted  $S_a(A) = \{h_{ak} \mid h_{ak} \text{ is on } Path_a(A)\}$ .

**Definition 2** The migration path of the mailbox of a mobile agent  $A$ , denoted  $Path_m(A)$ , is an ordered list of hosts  $(h_{m0}, h_{m1}, \dots, h_{mn})$  that the mailbox has visited in sequence. The set of hosts on the path is denoted  $S_m(A) = \{h_{mk} \mid h_{mk} \text{ is on } Path_m(A)\}$ . By definition, we have  $S_m(A) \subseteq S_a(A)$  and  $h_{m0} = h_{a0}$ .

**Definition 3** The function  $f_A : S_a(A) \rightarrow S_m(A)$  maps the location of a mobile agent  $A$  to that of its mailbox such that for every  $h_{ak} \in S_a(A)$ , we have:

$$f_A(h_{ak}) = \begin{cases} h_{ak} & k = 0, \text{ or } k > 0 \text{ and } A \text{ migrates with its mailbox} \\ f_A(h_{a(k-1)}) & k > 0 \text{ and } A \text{ migrates without its mailbox} \end{cases}$$

In the mailbox-based framework, choices can be made in three aspects of designing a protocol that can best suit the requirements of an application. The three aspects are:

- **Mailbox Migration Frequency.** A mobile agent might always be on the move, leaving its mailbox at home with *No Migration* (NM); or it might always migrate with its mailbox, resulting in a *Full Migration* (FM) pattern; or it might determine dynamically upon each migration, which we name *Jump Migration* (JM).
- **Mailbox-to-Agent Message Delivery.** As mentioned before, messages destined to a mobile agent are sent to its mailbox first and later received with either a push or pull operation. In the *Push* (PS) mode, the mailbox keeps the address of its owner and forwards every incoming message to it. In the *Pull* (PL) mode, the mobile agent keeps the address of its mailbox and retrieves messages from the mailbox whenever needed.
- **Migration-Delivery Synchronization.** Users can determine whether they need reliable message delivery or not. If users require high reliability, they can overcome message loss by (1) *Synchronizing the Host's message forwarding and the Mailbox's migration* (SHM), or by (2) *Synchronizing the Mailbox's message forwarding and the Agent's migration* (SMA), or (3) both, known as *Full Synchronization* (FS). NS denotes the extreme case of *No Synchronization* being performed.

With these the three aspects, a three-dimensional design space can be generated. As shown in Fig. 2, each aspect represents one orthogonal dimension. Since the three dimensions are independent of each other, designers can combine properties from different dimensions in various ways. The full range of properties can thus vary greatly.

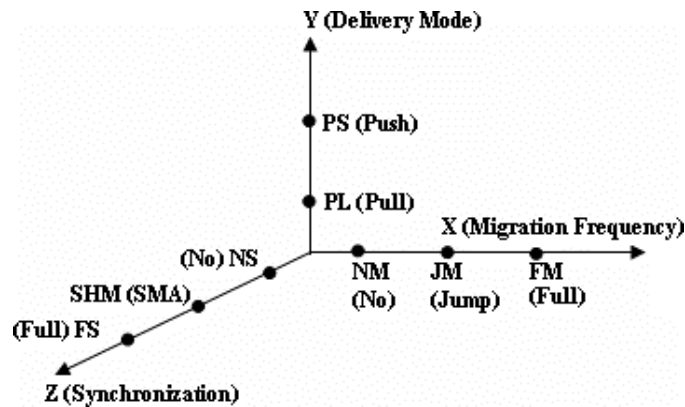


Fig. 2. 3-D design space.

Combining parameters from all three dimensions yields a taxonomy of mobile agent communication protocols. A string of the format  $XX-YY-ZZ$  expresses a protocol in which  $XX$  represents mailbox migration frequency (NM, JM, or FM),  $YY$  stands for mailbox-to-agent message delivery (PL or PS), and  $ZZ$  symbolizes migration-delivery synchronization (NS, SHM, SMA, or FS). A protocol's overall configuration has a special value for each of the three parameters. The framework not only covers several well known protocols but also allows for the design of new ones that can fulfill various application requirements. Interested readers may refer to [5] for detailed discussions of parameter combinations and corresponding protocols.

### 3. AN ADAPTIVE AND RELIABLE PROTOCOL

In this section we propose an *adaptive and reliable protocol* (ARP) derived from the three-dimensional framework. The ARP represents the JM-PL-SHM combination of the parameters. It guarantees reliable message delivery and can be designed to adapt to a changing environment.

In the ARP each host on  $Path_m(A)$  maintains the current address of the mailbox  $M_A$  in an *address table* which consists of five attributes: (i) the ID of  $M_A$ , (ii) the current address of  $M_A$ , (iii) a valid tag, (iv) the number of messages  $mNum$  that have been forwarded to  $M_A$ , and (v) a message block queue for  $M_A$ . The valid tag is used to indicate whether the address of  $M_A$  is outdated or not. Later we will see that the valid tag switches to *false* only when  $M_A$  is under migration. The message block queue is used to temporarily buffer messages to  $M_A$  when it is moving. The protocol also defines the operations for two processes, *Migrating* and *Message-forwarding*, which are presented in the next two subsections.

#### 3.1 Migrating

Fig. 3 illustrates the mobile agent migrating process. Before moving to a new host

$h_{ak}$ , the agent  $A$  determines whether to take its mailbox  $M_A$  to the new location. If it decides to do so, it will send an “MVMB” message to  $M_A$  (step 1) informing it to migrate to  $h_{ak}$ . The “MVMB” message contains the address of  $h_{ak}$ .

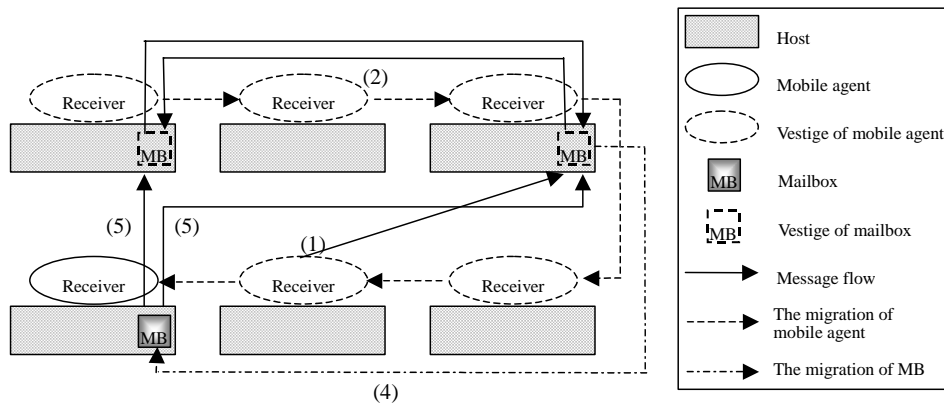


Fig. 3. The migrating process.

On receiving the “MVMB” message,  $M_A$  will send “DEREGISTER” messages to all the hosts on  $Path_m(A)$  (step 2) telling them to suspend the message forwarding and start buffering incoming messages for it. It is not until  $M_A$  has collected all the “REPLY” messages (step 3) that it can prepare to move to the new location. Besides informing  $M_A$  about the reception of the “DEREGISTER” message, the “REPLY” message also carries information about the number of messages  $mNum$  that have been forwarded to  $M_A$ . In this way,  $M_A$  can tell whether or not there are still any data messages in transmission at the time it receives the “REPLY” message. If there do exist some,  $M_A$  has to also wait for these messages before it can finally perform the migration (step 4).

When  $M_A$  arrives at  $h_{ak}$ , it registers its new address by sending each host on  $Path_m(A)$  a “REGISTER” message (step 5). This “REGISTER” message will restart the message forwarding process.

### 3.2 Message-forwarding

Fig. 4 illustrates the message-forwarding process. When a mobile agent wants to send a message to the agent  $A$ , it will first check  $A$ 's address from its local cache. If it exists, the message will be sent to the cached address. Otherwise, it will be sent to  $A$ 's home.

When a host receives a message destined to  $A$ , it will check from the address table whether  $M_A$  is residing locally or not. If it is yes, it will directly insert the message into  $M_A$ . Otherwise, it will forward the message to  $M_A$ 's current address (step 2), while at the same time, it will send back a “UPDATE” message to the sender (step 2') to refresh its cache about  $M_A$ 's current location.

In this protocol senders do not need to know the receiver's current location.

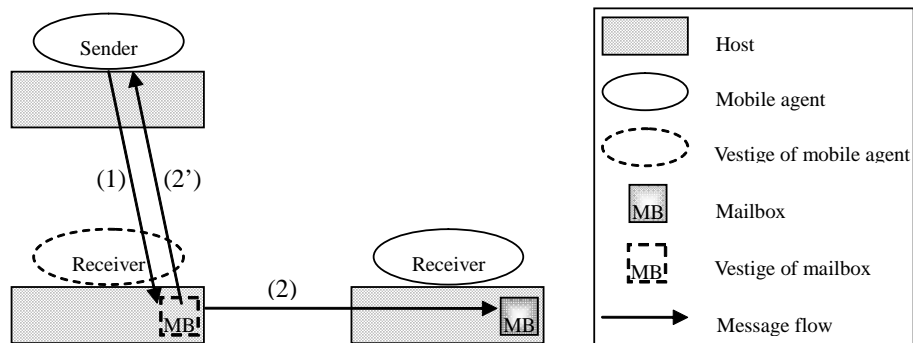


Fig. 4. The message-forwarding process.

Messages are first sent to the cached address or the receiver's home address and later forwarded to the receiver's mailbox. Synchronization between message delivery and mailbox migration is used to avoid message loss. The protocol guarantees that messages are forwarded at most once before they reach the mailbox of the receiver. Asynchrony is improved because constraints on agent mobility are released as synchronization involves only the mailbox, and the mobile agent can migrate to new locations whenever they want without waiting for the messages in transit. The protocol can also work in an adaptive way since it performs in a JM mode and the criterion that determines whether or not to move the mailbox remains unspecified. Base on the particular requirements of an application the designer can design a set of adaptive policies that can govern the migration pattern of the mailbox to maximize the fulfillment of the requirements according to the dynamically changing environment. For example, in [6] we have given an example that uses a single threshold  $T$  to determine the migration of the mailbox. Here  $T$  means the number of messages that the mobile agent expects to receive during the residence time at the new host. Simulations show that the protocol justifies the value of  $T$  so that the total communication cost always remains minimized. In the next section, we will present a path pruning and garbage collection algorithm to enhance the efficiency of the APR.

#### 4. A PATH PRUNING ALGORITHM

In the ARP the mailbox  $M_A$  of an agent  $A$  has to first deregister and then register with all the hosts on  $Path_m(A)$  for each migration. The migration overhead will increase continuously without an upper bound as  $Path_m(A)$  becomes longer and longer. Therefore, the protocol is only suitable for applications with a short agent life cycle and limited number of migrations. For applications with frequent and long term agent migration, some selective deregister/register algorithms must be figured out so as to maintain the migration overhead within an affordable scale.

Fortunately, we find that since the cache maintained by the sender is updated whenever it sends a message to an outdated address of  $M_A$ , it is very likely that each sender will refer to the latest host in  $Path_m(A)$  as the current address of  $M_A$ . The hosts in the

the front of  $Path_m(A)$  may no longer be referred to by any senders. These hosts can be safely removed from  $Path_m(A)$  to reduce migration overhead.

Another problem is the management of the cache. If an agent  $S$  sends messages to an agent  $A$ , the host in which  $S$  resides will cache the address of  $A$ 's mailbox  $M_A$ . However, when  $S$  leaves the host, the address of  $M_A$  in the cache might not any longer be used by any other agent. On the other hand, after the death of agent  $A$ , its address cached by other hosts will also become useless. Some garbage collection mechanism is needed to remove useless addresses in a cache so that the storage space of storage can be reused.

In this section we propose an algorithm to shrink the migration path of a mailbox and also to remove useless addresses maintained in a cache.

#### 4.1 The 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  $A$  denote a mobile agent. The function  $R_A : H \rightarrow S_m(A)$ , which defines the place where a host will forward a message to  $A$ , can be expressed as follows:

$$R_A(h_s) = \begin{cases} h_{mk} & h_{mk} \in S_m(A) \text{ and } h_{mk} \text{ is cached by } h_s \text{ as the current address of } M_A \\ h_{m0} & \text{otherwise} \end{cases}$$

**Definition 5** The set  $S_R(A, h_{mk}) = \{h_s \mid R_A(h_s) = h_{mk}\}$  denotes the set of all hosts that refer to  $h_{mk}$  as the current address of  $M_A$ . In other words, if a host  $h_s$  forwards a message  $A$  through  $h_{mk}$ , it must belong to  $S_R(A, h_{mk})$ .

**Definition 6** For each host  $h_{mk} \in S_m(A)$  and  $k > 0$ ,  $h_{mk}$  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 that redundant hosts can be safely removed from  $S_m(A)$  without affecting the reliability of message delivery. Therefore, the objective of our path pruning algorithm is to identify and remove those redundant hosts from  $S_m(A)$ . We extend the ARP as follows.

#### Data Structure Extension

- (1) Each cached address is associated with a timer which is initially set to 0 and starts as soon as the address is added to the cache. Each time the address is accessed (either updated or requested), the timer resets to 0. When the timer reaches TTL, the address is removed.
- (2) Each host  $h_{mk} \in S_m(A)$  maintains a reference table  $T(A)$  for the agent  $A$ , which contains a set of addresses and a *closed* tag. Later we will show by *Lemma 2* that  $T(A)$  maintains the addresses of those hosts in the set  $S_R(A, h_{mk})$ . The table is created if  $T(A)$

does not exist when  $M_A$  visits  $h_{mk}$ . On creation, the address set is empty and the *closed* tag is set *false*.

### Algorithm Extension

- (1) When  $h_{mi} \in S_m(A)$  receives  $M_A$ 's "REGISTER" message from  $h_{mj}$  ( $i \neq j$ ), it checks the reference table  $T(A)$ . If  $T(A).closed$  is *true*, nothing will be done. Otherwise,  $h_{mi}$  sets  $T(A).closed$  to *true* and starts the timer associated with  $T(A)$ . Both  $T(A)$  and the record of  $M_A$  in the address table are removed from  $h_{mi}$  as soon as  $T(A)$  becomes empty or the timer reaches  $TTL + MTL$ , where MTL means the maximum message transmission latency of the network.
- (2) When a host receives a message destined to  $A$  from  $S$  residing at host  $h_s$ , it first checks the reference table  $T(A)$ . If  $T(A)$  has already been removed from  $h_s$ , due to either of the reasons described in the previous paragraph, it will forward the message to  $A$ 's home, which always maintains the current address of  $M_A$ . (The agent home should not perform the path pruning algorithm. Otherwise, it would be possible for the agent home to not know the location of the mailbox.) Otherwise, it will check  $T(A).closed$ . If  $T(A).closed$  is *false* and  $h_s$  is not in  $T(A)$ ,  $h_s$  is added to  $T(A)$ . If  $T(A).closed$  is *true* and  $h_s$  is in  $T(A)$ ,  $h_s$  is removed from  $T(A)$ .
- (3) When  $h_{mi} \in S_m(A)$  receives  $M_A$ 's "DEREGISTER" message from  $h_{mj}$  ( $i \neq j$ ) and finds that the record of  $M_A$  has been removed from its address table, it will send back 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  $Path_m(A)$ .

The path pruning algorithm can be integrated into the original ARP as shown in the following pseudo-code where the instructions in boldface represent the modifications.

### 4.2 Correctness

Here we present an informal proof of the effectiveness of the algorithm. Firstly, we present two basic assumptions. Later we will show in the simulation that with a minor revision, the algorithm can work even without these assumptions, although the performance is slightly degraded.

**Assumption 1** The message transmission latency of the network is no larger than MTL.

**Assumption 2** The interval in which a sender sends two consecutive messages destined to the same receiver agent is no less than  $2MTL$ .

**Lemma 1** For any host  $h_{mk} \in S_m(A)$  and  $k > 0$ ,  $h_{mk}$  will not receive any messages to agent  $A$  from host  $h_s$  after  $h_s$  is removed from table  $T(A)$ .

**Proof:** According to the path pruning algorithm,  $h_s$  is removed from  $T(A)$  only if  $h_{mk}$  receives a message destined for agent  $A$  from  $h_s$  and the *closed* tag of  $T(A)$  is *true*.  $h_{mi}$  will send an “UPDATE” message to  $h_s$  informing it of the new address of  $M_A$ . By *Assumption 2*,  $h_s$  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  $A$ .

**Lemma 2** For all hosts  $h_s \in S_R(A, h_{mk})$ , either  $h_s$  already exists in  $T(A)$ , or it will be added to  $T(A)$  within MTL.

**Proof:** Suppose  $M_A$  is residing at  $h_{mk}$  and  $R_A(h_s) = h_{mj}$  ( $k \neq j$ ). The message destined for agent  $A$  will be sent to  $h_{mj}$  by  $h_s$ . According to our algorithm,  $h_{mj}$  will forward the message to  $h_{mk}$  and send an “UPDATE” message to  $h_s$ . Upon receiving the “UPDATE” message,  $h_s$  will update its cache and let  $R_A(h_s) = h_{mk}$  (therefore, we have  $h_s \in S_R(A, h_{mk})$ ). After  $h_{mk}$  receives the data message from  $h_{mj}$ , it will add  $h_s$  to the reference table  $T(A)$ . If the arrival of the data message at  $h_{mk}$  is earlier than the arrival of the “UPDATE” message at  $h_s$ ,  $h_s$  would have been added to  $T(A)$  before it updates its cache. Otherwise, since the “UPDATE” message and the data message are sent out by  $h_{mj}$  at almost 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  $T(A)$  within MTL after  $R_A(h_s)$  turns to  $h_{mk}$ , i.e.  $h_s \in S_R(A, h_{mk})$ .

**Lemma 3** No new hosts will join set  $S_R(A, h_{mk})$  after the *closed* tag of  $T(A)$  turns *true*.

**Proof:** Turning the *closed* tag of  $T(A)$  *true* implies that  $M_A$  has left  $h_{mk}$  for  $h_{m(k+1)}$  and  $h_{mk}$  has received the “REGISTER” message from  $M_A$  at  $h_{m(k+1)}$ . All the hosts  $h_{mj} \in S_m(A)$  must have received the “DEREGISTER” message from  $M_A$  at  $h_{mk}$ . Either  $M_A$ 's address maintained in the address table of  $h_{mj}$  has been updated ( $h_{mj}$  has also received the “REGISTER” message from  $M_A$  at  $h_{m(k+1)}$ ), 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  $h_{mk}$  as the current address of  $M_A$ . Therefore, no new hosts will join  $S_R(A, h_{mk})$ .

**Theorem 1**  $h_{mk}$  is a redundant host in  $S_m(A)$  if the *closed* tag of  $T(A)$  is *true* and  $T(A)$  is empty.

**Proof:** According to *Lemma 2* and *Lemma 3*, if the *closed* tag of  $T(A)$  is *true* and  $T(A)$  is empty, we have  $S_R(A, h_{mk}) = \emptyset$ . From *Lemma 1* and *Definition 5*, we know that  $h_{mk}$  will no longer receive any messages destined for agent  $A$  unless  $M_A$  revisits it. Therefore,  $h_{mk}$  is a redundant host in  $S_m(A)$ .

**Theorem 2**  $h_{mk}$  is a redundant host in  $S_m(A)$  if the timer of  $T(A)$  reaches  $\text{TTL} + \text{MTL}$ .

**Proof:** If the timer of  $T(A)$  has reached  $\text{TTL} + \text{MTL}$ , we know the *closed* tag of  $T(A)$  is

*true*. This is because the timer is started only after the *closed* tag of  $T(A)$  is set *true*. If  $T(A)$  is empty, we have proved that  $h_{mk}$  is a redundant host (*Theorem 1*). Otherwise, suppose  $h_s$  is in  $T(A)$ . According to our algorithm, we know that  $h_{mk}$  has not received any messages destined for agent  $A$  from  $h_s$  since the *closed* tag of  $T(A)$  has turned *true*. This implies that:

- The address of  $M_A$  cached by  $h_s$ , if not being cleared, is  $h_{mk}$  (i.e.  $R_A(h_s) = h_{mk}$ ) and it has not been updated since the *closed* tag of  $T(A)$  turned *true*.
- The address of  $M_A$  cached by  $h_s$  has not been read for a period of TTL since the *closed* tag of  $T(A)$  turns *true*. That is 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 that there have been no messages sent from  $h_s$  to agent  $A$  for at least a period of TTL.

From the above two points, we know the physical address of  $M_A$  in the cache of  $h_s$  has been neither updated nor read for at least TTL time units. Therefore,  $h_s$  must have removed the address of  $M_A$  from its cache. By definition, we have  $h_s \notin S_R(A, h_{mk})$ , and  $h_s$  can be safely removed from  $T(A)$ . In this way, we can safely empty  $T(A)$ . By *Theorem 1*, we know that  $h_{mk}$  is a redundant host.

Proofs of *Theorem 1* and *Theorem 2* depend on both *Assumption 1* and *Assumption 2*. Without these assumptions, messages to the agent  $A$  may arrive at  $h_{mk}$  even after  $h_{mk}$  has been removed from  $S_m(A)$ . In this case, we let  $h_{mk}$  forward the message to  $h_{m0}$ , i.e., the home of agent  $A$ . Since  $h_{m0}$  should always know the physical address of  $M_A$ , it can finally 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 can be maintained.

Since an address in the cache is removed if it is not accessed within TTL, the algorithm also provides a way to clear useless addresses maintained in the 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 for delivery.

The value of TTL greatly affects the performance of the path pruning algorithm. If TTL is very large, the probability that a sender cannot find a receiver's address in the cache is small and there is a small increment in the workload of the receiver's home. However, the redundant hosts in the migration path of the receiver's mailbox may not be removed in time, and we cannot achieve much reduction of the migration cost. On the other hand, with a small TTL, the migration cost can be greatly reduced by path pruning, but more messages must be forwarded by the receiver's home. There are two extreme cases of the value of TTL. One is that TTL is greater than the life cycle of the receiver. In this case, there would be no path pruning performed and the protocol works just the same way as the original ARP. The other extreme case 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, which can be viewed as a variation of a home server-based message delivery scheme [7, 8].

## 5. PERFORMANCE ANALYSIS

In this section we evaluate the performance of the path pruning algorithm under different circumstances through simulations. The performance metrics used is the communication cost, which is characterized by the number of messages sent as well as the message size.

### 5.1 Simulation Model

Our simulation is built on the Network Simulator 2 (*ns2*) developed by the Lawrence Berkeley National Laboratory [9]. We have incorporated into the simulator the original adaptive routing protocol as well as the path pruning algorithm proposed in this paper.

(1,10)	(2,10)	(3,10)	(4,10)	(5,10)	(6,10)	(7,10)	(8,10)	(9,10)	(10,10)
(1,9)	(2,9)	(3,9)	(4,9)	(5,9)	(6,9)	(7,9)	(8,9)	(9,9)	(10,9)
(1,8)	(2,8)	(3,8)	(4,8)	(5,8)	(6,8)	(7,8)	(8,8)	(9,8)	(10,8)
(1,7)	(2,7)	(3,7)	(4,7)	(5,7)	(6,7)	(7,7)	(8,7)	(9,7)	(10,7)
(1,6)	(2,6)	(3,6)	(4,6)	(5,6)	(6,6)	(7,6)	(8,6)	(9,6)	(10,6)
(1,5)	(2,5)	(3,5)	(4,5)	(5,5)	(6,5)	(7,5)	(8,5)	(9,5)	(10,5)
(1,4)	(2,4)	(3,4)	(4,4)	(5,4)	(6,4)	(7,4)	(8,4)	(9,4)	(10,4)
(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	(6,3)	(7,3)	(8,3)	(9,3)	(10,3)
(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	(6,2)	(7,2)	(8,2)	(9,2)	(10,2)
(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)	(7,1)	(8,1)	(9,1)	(10,1)

Fig. 5. Network topology setting.

The network configuration of the simulation is shown in Fig. 5. There are all together 100 hosts, each of which is associated with a coordinate  $(x, y)$ . They are all interconnected with each other. As the two most important arguments required by *ns2*, we configure the bandwidth and the propagation delay between any two hosts in the following way. Any two adjacent hosts are considered to be within a local LAN environment which has a bandwidth of 10Mbps and a propagation delay of 2ms. The two hosts with the largest distance, i.e., the host at  $(1, 1)$  and the host at  $(10, 10)$ , are treated as if they reside on opposite sides of the earth with a small bandwidth of 64Kbps and a large propagation delay of 100ms. To make it simple, we assume that both the bandwidth and the propagation delay between any two hosts are proportional to their distance. So, we can derive the bandwidth and the propagation delay between host  $A$  and  $B$  as

$$Bandwidth(A, B) = (Dist(A, B) - 1) \frac{64Kbps - 10Mbps}{10\sqrt{2} - 1} + 10Mbps \quad (1)$$

$$Delay(A, B) = (Dist(A, B) - 1) \frac{100ms - 2ms}{10\sqrt{2} - 1} + 2ms \quad (2)$$

$$Dist(A, B) = \sqrt{(A(x) - B(x))^2 + (A(y) - B(y))^2} \quad (3)$$

Next we present the definitions of parameters used in the simulation model.

- The receiver and its mailbox are denoted  $A$  and  $M_A$ , respectively.
- $S$ : the set of senders in the network that might send messages to  $A$ .
- $t_{s_i}$ : the intervals during which the sender  $s_i \in S$  sends two consecutive messages to  $A$ .
- $f_{s_i}(t)$ : we assume a negative exponential probability distribution function of the message sending interval of  $s_i$ .
- $p_{s_i}(t, n)$ : the probability distribution function (Poisson distribution) of the number of messages sent from  $s_i$  during the time interval  $t$ .
- $\lambda_{s_i}(t)$ : the mean message sending rate of  $s_i$ , i.e.,  $f_{s_i}(t) = \lambda_{s_i} e^{-\lambda_{s_i} t}$ ,  $p_{s_i}(t, n) = (\lambda_{s_i} t)^n e^{-\lambda_{s_i} t} / n!$ .
- $t_r$ : the residence time  $A$  spends in a host.
- $f_r(t)$ : we also assume a negative exponential probability distribution function for an agent's residence time.
- $1/\mu$ : the mean residence time,  $f_r(t) = \mu e^{-\mu t}$ .
- $p_i$ : the probability that  $A$  will not take  $M_A$  to the new location during its  $i$ th migration,  $p_i = Prob(f_A(h_{ai}) = f_A(h_{a(i-1)}))$ .
- $p_{hit}$ : the probability that the mailbox's location information cached by the underlying host of the sender is correct.
- $p_a$ : the probability that the forwarding host has not been prematurely removed from the migration path of  $M_A$ . In the ARP,  $p_a$  is always 1. In the ARP with path pruning, we have proved that  $p_a$  is always 1 if *Assumption 1* and *Assumption 2* are satisfied. Otherwise,  $p_a$  may be less than 1.

By removing the redundant hosts, the migration path maintained by  $M_A$  is reduced. To differentiate the actual migration path of  $M_A$  denoted as  $S_m(A)$  and the path maintained by  $M_A$  in which the redundant hosts have been removed, we create another symbol  $S_p(i)$  to denote the set of hosts maintained by  $M_A$  after the  $i$ th migration of  $A$ . Therefore, in the ARP we have

$$S_p(i) = \begin{cases} S_p(i-1) & i > 0 \text{ and } f_A(h_{ai}) = f(h_{a(i-1)}); \\ S_p(i-1) \cup \{h_{ai}\} & i > 0 \text{ and } f_A(h_{ai}) = h_{ai} \end{cases};$$

and in the ARP with path pruning, we have:

$$S_p(i) = \begin{cases} S_p(i-1) & i > 0 \text{ and } f_A(h_{ai}) = f(h_{a(i-1)}) \\ S_p(i-1) \cup \{h_{ai}\} - R(i) & i > 0 \text{ and } f_A(h_{ai}) = h_{ai} \end{cases},$$

where  $R(i)$  denotes the set of redundant hosts identified during the  $i$ th migration of  $A$ .

If  $A$  takes  $M_A$  to its target host in the  $i$ th migration, the migration overhead involves the communication cost of the ‘‘MVMB’’ message, the ‘‘DEREGISTER’’ messages, the ‘‘REPLY’’ or ‘‘NAK’’ messages, and the ‘‘REGISTER’’ messages. Otherwise, the migration overhead is zero. Therefore, the overhead of  $A$ 's  $i$ th migration can be denoted as

$$C_{mig}(i) = (1 - p_i)(p_i - 1)C_{mvm} + (|S_p(i-1)| - 1)C_{deregister} + (|S_p(i-1)| - 1)C_{reply/nak} + (|S_p(i) - 1)C_{register}, \quad (4)$$

where  $|S_p(i)|$  means the number of hosts existing in the set  $S_p(i)$ . The total migration overhead during the life cycle of  $A$  is then given by

$$C_{mig} = \sum_{i=1}^N C_{mig}(i), \quad (5)$$

where  $N$  is the total number of migrations during the life cycle of  $A$ .

While  $A$  is residing at  $h_i$ , the cost of message delivery from sender  $s_j$  involves the cost of message passing from  $s_j$  to the cached address (or the home of  $A$ ), the cost of once again a forwarding message to the home of  $A$  should the forwarding host have been removed prematurely as a redundant host, the cost of message forwarding to  $M_A$  and the ‘‘UPDATE’’ message should a cache miss occur, and finally the cost of message retrieval by the receiver. Thus the delivery cost is

$$C_{del}(i,j) = n_{i,j} (C_{s_j \rightarrow h_{mk}} + (1 - p_{hit})((1 - p_a)C_{h_{mk} \rightarrow h_{m0}} + C_{h_{mk} \rightarrow M_A/h_{m0} \rightarrow M_A} + C_{update}) + p_i(C_{M_A \rightarrow receiver} + \alpha C_{query})), \quad (6)$$

where  $n_{i,j}$  denotes the number of messages sent to  $A$  from  $s_j$  while it is residing at  $h_i$ , and  $\alpha$  is the ratio of the number of query messages to the number of messages obtained from  $M_A$ . The total message delivery cost is

$$C_{del} = \sum_{i=1}^N \sum_{j=0}^M C_{del}(i,j), \quad (7)$$

where  $M$  is the number of mobile agents in  $S$ .

The total communication cost is therefore

$$C_{total} = C_{mig} + C_{del}. \quad (8)$$

Before migrating to the new host  $h_{ai}$ , there are many factors  $A$  might consider in deciding whether or not its mailbox should move. For example, if  $A$  seldom receives messages from others at  $h_{ai}$ , it does not need to take its mailbox. On the other hand, if  $A$  is expected to frequently receive messages from others, leaving the mailbox unmoved will waste resources on the suboptimal message delivery route and it would be much better for  $A$  to collocate with the mailbox. In this simulation we use a fixed threshold  $T$  for the number of messages to make the decision. Before the  $i$ th migration,  $A$  estimates the number of messages it will receive at the new host, which is based on the mean message arrival rate as well as the mean residence time at a host. If the estimated number exceeds the threshold  $T$ , it will send an “MVMB” message to fetch its mailbox. Otherwise, it does nothing.  $T$  remains unchanged during the life cycle of  $A$ .

## 5.2 Simulation Results

In the simulation, we let  $S = \{s_0, s_1, \dots, s_9\}$ . Receiver  $A$  migrates sequentially from host (1, 1) to host (10, 10). MTL is set to be the same as the largest propagation delay, i.e., 100ms. The ratio between the number of query messages and the number of messages from the mailbox ( $\alpha$ ) is set 1.5. The agent’s mean residence time  $1/\mu$  is set to 500ms. We use  $C_{ctrl}$  and  $C_{msg}$  to denote the cost of a control message (e.g.,  $C_{mvmb}$ ) and a data message (e.g.,  $C_{s_i \rightarrow h_{mk}}$ ), respectively. Since control messages such as “MVMB” and “REGISTER” may be much smaller in size than data messages, they should not be counted in the same way and are assigned to be 128 bytes and 512 bytes, respectively.

Figs. 6, 7 and 8 show the migration cost of agent  $A$ , the message delivery cost between senders and  $M_A$ , and the total communication cost, respectively. Curves with different  $1/\lambda$  are illustrated. The value of TTL varies from 0 to 1000. To see the performance results more clearly, the X-axis is expressed in terms of  $\log(TTL + 1)$ . We use  $\log(TTL + 1)$  instead of  $\log(TTL)$  because  $TTL$  starts at 0.

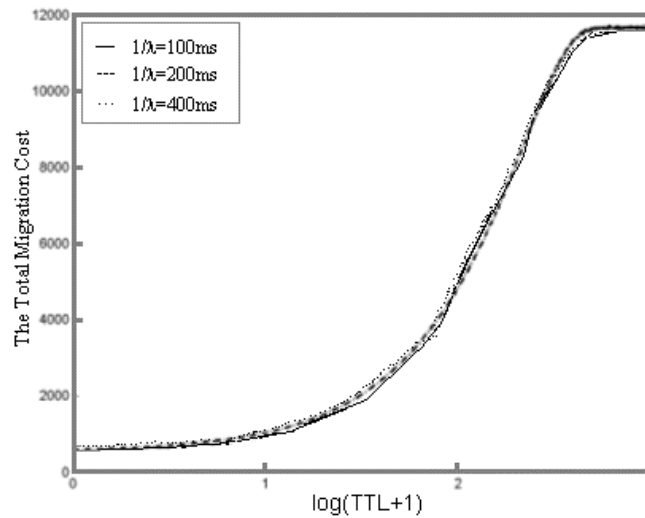


Fig. 6. The total migration cost.

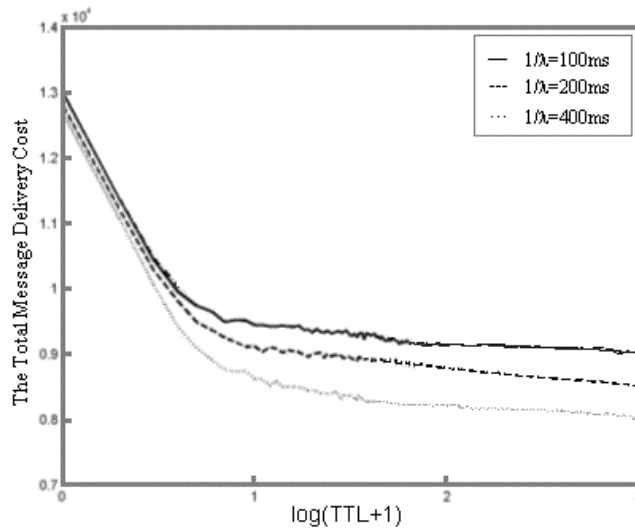


Fig. 7. The total message delivery cost.

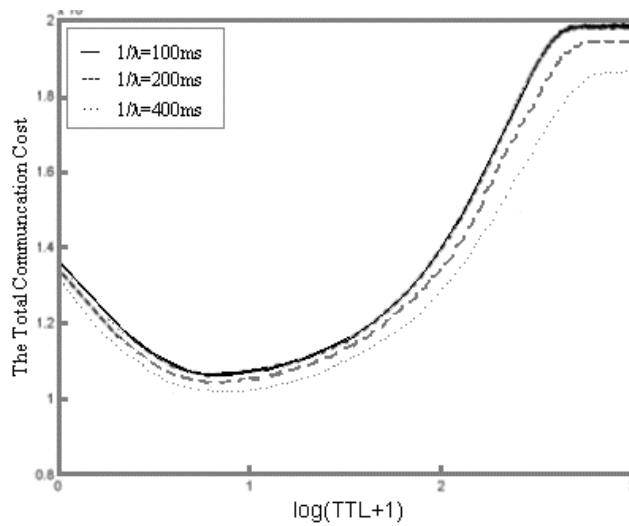


Fig. 8. The total communication cost.

The migration cost and message delivery cost of the original ARP without path pruning are shown in the above figures when TTL is large enough, and is represented by the right end of each curve. Similarly, the migration cost and message delivery cost of the home server based message forwarding protocol are shown when TTL is 0, and is represented by the left end of the curves.

As shown in Fig. 6, the agent migration cost can be reduced by using the path pruning algorithm. The migration cost is minimized when TTL is 0 because only one host, the

home of the receiver, is in  $S_p(i)$ . The migration cost increases as TTL increases. The upper bound is the migration cost of the original ARP. Also we can see that  $1/\lambda$  does not affect the migration cost because it influences only the message delivery cost and has nothing to do with the number of hosts maintained in  $S_p(i)$ .

However, the reduction in migration cost is at the expense of an increase in message delivery cost. From Fig. 7, we can see that the message delivery cost is large when TTL is small because more messages have to be forwarded by the agent home. We can also draw this conclusion from (6) because the more quickly the cache is cleared, the smaller  $p_{hit}$  will be. When TTL gets larger, the sender can take full advantage of the cached address of  $M_A$  and  $p_{hit}$  will get larger. As mentioned in Section 4.2, the smaller  $1/\lambda$  is, the more likely senders will send messages to hosts that have already been removed from  $S_p(i)$  and the more cost will be wasted on the once again a forwarding message to the home of the receiver. When TTL is 0, all messages are sending with two steps through the home and thus there makes no difference with different  $1/\lambda$ . With increasing TTL as well as decreasing  $1/\lambda$ , more packets will face the circumstance of once again a forwarding message.

Fig. 8 illustrates the total communication cost, i.e., the sum of the total migration cost and total message delivery cost. As we can see, in most cases, the communication cost of the adaptive protocol can be reduced by properly setting the value of TTL.

In section 4.2, we proved the correctness of the path pruning algorithm with two basic assumptions. However, in real situations these two assumptions, especially the second one, cannot be guaranteed. In Fig. 8, the curve with  $1/\lambda = 100\text{ms}$  ( $\ll 2\text{MTL} = 200\text{ms}$ ) demonstrates the performance of the path pruning algorithm without guaranteeing the two assumptions; while the curve with  $1/\lambda = 400\text{ms}$  ( $\gg 2\text{MTL} = 200\text{ms}$ ) illustrates the performance with strong fulfillment of the assumptions. We can see that the performance shows only a slight degradation by relaxing the requirement of the two assumptions. This result also exhibits the fact that it is really low for the probability of once more message forwarding to the receiver's home due to a premature removal of a host from the mailbox's migration path.

## 6. RELATED WORKS

Many mobile agent/object tracking schemes have been proposed in the last several years in different contexts, including mobile agents, wireless communications and wide-area distributed systems. The major schemes include home server, forwarding pointer, hierarchical location directory and broadcast. Readers are referred to [10] and [11] for excellent surveys of these schemes. Our proposed mailbox-based framework not only covers these schemes as discussed in our previous work [5], but also allows new schemes to be designed.

To achieve optimized performance, it is required that the sender send its messages to the receiver with as few intermediate hops as possible because any intermediate hops requires one forwarding table lookup and might result in a suboptimal detour. Also, without a proper management strategy, substantial cost will be wasted on the maintenance of the tracking information maintained in the network. Thus path pruning is a necessary for the success of a forwarding-based mobile communication protocol. Various

path pruning techniques have been developed. For example, in [12], forwarding pointer loops are detected and removed so as to shorten the forwarding path. In [13], path pruning is achieved by introducing forwarding pointers to the base hierarchical architecture. A sender will not have to traverse the hierarchical tree before tracking the receiver. It only needs to follow a shortcut of the forwarding pointer. Also pointer loops are detected. Hierarchical structure refreshing is periodically or dynamically determined by considering a trade-off in mobile tracking performance.

Our proposed path pruning algorithm is a new approach and reinforcement to further reduce the tracking path should any forwarding-based message delivery be used in the mobile tracking system. This is because our algorithm could effectively identify and remove those redundant hosts in the tracking path. By carefully selecting some parameters, our algorithm could also behave in an efficient and adaptive way. Therefore, our proposed algorithm is orthogonal to any other path pruning techniques and can even work in parallel with them for any forwarding pointer scheme.

## 7. CONCLUSION AND FUTURE WORK

In this paper we proposed a path pruning and garbage collection algorithm to improve the efficiency of a mailbox-based mobile agent communication protocol. Simulations show us a very sound performance improvement achieved by the algorithm. Actually, the proposed path pruning technique can be used in any forwarding-based message delivery protocol to prune away useless forwarding pointers in the forwarding path. Future work includes a deep adaptability study about the dynamical selection of the optimal TTL so that the communication cost can always be kept at a minimum.

## ACKNOWLEDGMENT

This work is partially supported by The Hong Kong Polytechnic University under the HK PolyU Research Grants A-PC53 and G-YD63.

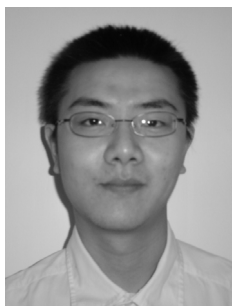
## REFERENCES

1. D. B. Lange and M. Oshima, "Seven good reasons for mobile agents," *Communications of the ACM*, Vol. 42, 1999, pp. 88-89.
2. J. Cao, L. Zhang, J. Yang, and S. K. Das, "A reliable mobile agent communication protocol," in *Proceedings of 24th International Conference on Distributed Computing Systems (ICDCS '04)*, 2004.
3. A. Murphy and G. P. Picco, "Reliable communication for highly mobile agents," *Agent Systems and Architectures/Mobile Agents (ASA/MA)'99*, 1999, pp. 141-150.
4. B. Awerbuch and D. Peleg, "Online tracking of mobile users," *Journal of the ACM*, Vol. 42, 1995, pp. 1021-1058.
5. J. Cao, X. Feng, J. Lu, and S. K. Das, "Mailbox-based scheme for mobile agent communications," *IEEE Computer*, 2002, pp. 54-60.
6. J. Cao, L. Zhang, X. Feng, and S. K. Das, "Adaptive and reliable message delivery for

- mobile objects,” in *Proceedings of 2003 IEEE Global Communications Conference (Globecom 2003)*, 2003, pp. 3196-3200.
7. M. D. Gallagher and R. A. Snyder, “Mobile telecommunication networking with IS-41,” McGraw-Hill, Inc., 1997.
  8. C. E. Perkins, “IP mobility support,” RFC2002, October 1996.
  9. Network Research Group, Lawrence Berkeley National Laboratory, ns-LBNL Network Simulator, URL: <http://www-nrg.ee.lbl.gov/ns/>.
  10. P. T. Wojciechowski, “Algorithms for location-independent communication between mobile agents,” Technical Report 2001/13, Communication Systems Department, EPFL, March 2001.
  11. E. Pitoura and G. Samaras, “Locating objects in mobile computing,” *IEEE Transactions on Knowledge and Data Engineering*, Vol. 13, 2001, pp. 571-592.
  12. J. Desbiens, M. Lavoie, and F. Renaud, “Communication and tracking infrastructure of a mobile agent system,” in *Proceeding 31st Hawaii International Conference on System Sciences, Agent Mobility and Communication*, 1998, pp. 54-63.
  13. E. Pitoura and I. Fudos, “Distributed location databases for tracking highly mobile objects,” *The Computer Journal*, Vol. 44, 2001, pp. 75-91.



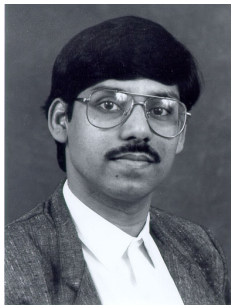
**Jiannong Cao (曹建農)** received the B.S.C. degree in Computer Science from Nanjing University, China and the M.S. and Ph.D. degrees from Washington State University, U.S.A., all in Computer Science. Before joined the Hong Kong Polytechnic University in 1997, where he is currently an associate professor, he has been on faculty of Computer Science in James Cook University and The University of Adelaide in Australia, and the City University of Hong Kong. Cao's research interests include parallel and distributed computing, networking, mobile computing, fault tolerance, and distributed programming environments. He has authored or co-authored more than 100 journal and conference papers in the above areas. Cao is a member of the IEEE Computer Society, IEEE, ACM. He has served as an editor for several international journals, a reviewer for journals / conferences, and also as an organizing / program committee member for many international conferences.



**Liang Zhang (張良)** received the B.S. degree in the Department of Computing from the Hong Kong Polytechnic University in 2002. He is now an M.Phil. student in the same department. His research interest is the design of reliable and efficient communication protocols for mobile computing.



**Xinyu Feng (馮新宇)** was born in Heze, China on July 2, 1978. He got his B.S. degree in July 1999, and M.S. degree in 2002, both in Computer Science, in Nanjing University, Nanjing, China. From June 2001 to December 2001 he worked as a visiting research assistant in the Department of Computing, The Hong Kong Polytechnic University. Currently Xinyu Feng is a Ph.D. student in the Department of Computer Science at Yale University. Xinyu Feng is interested in distributed systems and programming languages.



**Sajal K. Das** is a Professor of Computer Science and Engineering and also the Founding Director of the Center for Research in Wireless Mobility and Networking (CReWMaN) at the University of Texas at Arlington (UTA). His current research interests include resource and mobility management in wireless networks, mobile and pervasive computing, wireless multimedia and QoS provisioning, sensor networks, mobile Internet protocols, distributed processing and grid computing. He has published over 250 research papers, directed numerous funded projects, and holds 5 US patents in wireless mobile networks. He received the Best Paper Awards in ACM MobiCom'99, ICOIN-2001, ACM MSWIM-2000, and ACM/IEEE PADS'97. Dr. Das is also a recipient of UTA's Outstanding Faculty Research Award in Computer Science in 2001 and 2003, and UTA's College of Engineering Excellence in Research Award in 2003. He serves on the Editorial Boards of IEEE Transactions on Mobile Computing, ACM/Kluwer Wireless Networks, Parallel Processing Letters, Journal of Parallel Algorithms and Applications. He served as General Chair of IEEE PerCom-2004, CIT-2003 and IEEE MASCOTS-2002; General Vice Chair of IEEE PerCom-2003, ACM MobiCom-2000 and HiPC 2000-01; General Chair of ACM WoWMoM 2000-02; Program Chair of IWDC-2002, WoWMoM 1998-99; TPC Vice Chair of ICPADS-2002; and as TPC member of numerous IEEE and ACM conferences. He is the Vice Chair of IEEE TCPP and TCCC.