An Efficient Mailbox-Based Algorithm for Message Delivery in Mobile Agent Systems*

Xinyu Feng[1], Jiannong Cao[2], Jian Lü[1], and Henry Chan[2]

[1] State Key Laboratory for Novel Software Technology
Dept. of Computer Science, Nanjing Univ.
Nanjing, China
fxy@softlab.nju.edu.cn, lj@nju.edu.cn

[2] Internet Computing and E-Commerce Lab
Dept. of Computing, Hong Kong Polytechnic Univ.
Hung Hom, Kowloon, Hong Kong
{csjcao | cshchan}@comp.polyu.edu.hk

Abstract. Agent mobility presents challenges to the design of efficient message transport protocols for mobile agent communications. A practical mobile agent communication protocol should provide location transparency to the programmer and thus need to keep track of the movement of an agent. In addition, because of the asynchronous nature of message passing and agent migration, how to guarantee the delivery of messages to highly mobile agents is still an active research topic in mobile agent systems. In this paper we propose an efficient mailbox-based algorithm for inter-mobile agent communications. The algorithm decentralizes the role of the origin (home) host in locating an agent. Furthermore, by separating the mailbox from its owner agent, the algorithm can be made adaptive and is efficient in terms of location updating and message delivery. In the cases that mobile agents migrate frequently but seldom communicate, our algorithm turns out to be preferable.

1. Introduction

In recent years, mobile agent computing has emerged as a new paradigm in developing applications in various areas including telecommunications, networking / distributed systems, and e-commerce. Mobile agents are active, autonomous objects or object clusters, which are able to move between locations in a so-called mobile agent system. A mobile agent system is a distributed abstraction layer that provides security of the underlying system on one hand and the concepts and mechanisms for mobility and communication on the other hand [1, 2].

Mobile agents used in various applications need to communicate with each other for different purposes such as exchanging information and/or co-operation [3, 4]. Although process communication has been a cliché in the research of distributed systems, agent mobility poses a number of problems in designing message delivery

---

* This research is partially supported by Hong Kong Polytechnic University Research Grant A-PC53.

mechanisms for effective and efficient communications between mobile agents. 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 partners' 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. On the other hand, the asynchronous nature of message passing and agent migration may cause the loss of messages being sent to an agent on its move. Thus, a reliable agent communication mechanism should also guarantee the delivery of messages to highly mobile agents. Besides, the agent location tracking and message routing algorithm should not introduce too much overhead or offset any of the merits of mobile agent technology.

Many currently available mobile agent systems do not provide solutions to these problems and leave the hard nuts to agent programmers [1, 5]. Although there are several protocols proposed trying to provide location transparency and reliable inter-agent communication [7~10], they either handle it in a way too complicated to be efficient in practical systems, or use home-registration and rely too much on agent home, which is improper when a disconnected execution is needed.

The mailbox-based algorithm proposed in this paper adopts a hybrid approach combining the registration and forwarding schemes to locate mobile agents and deliver messages. Using the algorithm, messages can be delivered in a reliable and location transparent way. By forwarding the message at most once, the algorithm resolves the problem that messages keep chasing their highly mobile target agents. Unlike the home registration method used in mobile computing, e.g., Mobile IP [11], the algorithm decentralizes the role of the origin (home) host in locating an agent. This reduces the reliance on a single host, so that the agent's ability to support disconnected operation, considered as an important advantage of mobile agent technology [12], can be achieved in a real sense. Furthermore, by separating the mailbox from its owner agent, the algorithm can be made adaptive and is efficient in terms of location updating and message delivery.

The remaining of this paper is organized as follows. Section 2 presents a brief review of related work. Section 3 describes our mailbox-based algorithm in detail and also presents a proof of its properties. In Section 4 we analyse the performance of the proposed algorithm. Section 5 describes the simulation results and discusses the relationship between the communication overhead and mailbox migration frequency. The final section provides some concluding remarks.

2. Related Work

To communicate with a remote mobile agent, we must find the location of the agent and route the message to it. A naming scheme is needed to identify agents in a unique fashion. The name should not change whenever the agent migrates to other hosts and it is up to the tracking algorithm to map the name to the agent's current address. The routing process can be done either in parallel with agents tracking [9] or in a second phase after the address has been got [7].

The usual way to name an agent [7, 9] is to append the address of an agent's origin host (i.e. agent home) with its title (a free form string used to refer to this agent). Thus it is impossible for agents born at different agent platforms to have the same name. For agents created at the same host, the origin host is responsible to manage the name space to ensure that each agent has a unique title. In this paper we adopt this naming scheme.

There are three basic schemes for locating agents, namely s*earching, logging* and *registration* [5]. In the searching approach, we either send an agent to visit every host that the target agent might reside in or broadcast locating messages to these hosts [8]. The overhead is unaffordable when the network is large. The logging method locates the mobile agent by following the trail information indicating its next destination, left in every host the agent has ever visited [9]. If the trail information is lost or if one of the hosts is down, the target agent would no longer be found. With the registration scheme, an agent needs to update its location in a predefined directory server (e.g., its home host) that allows agent to be registered, deregistered or located. The directory server can be either a central node, which may become the bottleneck of the system performance and/or a single point of failure, or the agent's home host, which follows the idea of Mobile IP [11].

Two common methods for message routing are *forwarding* and *locate-and-transfer*. Under the forwarding scheme (also called *path-extension*), locating a receiver agent and delivering a message to it are both done in a single phase. They are combined into one operation where an agent moved to a new host informs the previous resident host where it moves so that messages can be forwarded along the extended path. The disadvantage is that messages may take multi-hops before they reach the target agents. The performance is worsened when messages are large in size. On the other hand, locate-and-transfer locates the target agent first and then transfers the message directly to it. However, the message sender may get outdated address in cases that the receiver agent migrates frequently.

The Mobile IP [11] is the protocol designed for IP packets routing to mobile devices. A mobile host registers its care-of-address with its home host and it is the home host that forwards the IP packets to it. Although this home registration and forwarding method is easy to implement and has less location registration overhead, it is inappropriate in mobile agent systems. Since all the correspondents of an agent must find its address form its home host, the agent home host may be a performance bottleneck when a larger number of agents, each with many correspondents, are originated from that same host. Besides, the agent home host may sometimes break off from the network after the agent is dispatched. Disconnected computing cannot be supported under this scheme.

Among the mobile agent systems and programming environments that are currently available, few provides practical and efficient algorithms for mobile agent communications. In Mole [1] there is no solution to location transparent remote communication. An agent must give the current address of its correspondent explicitly in order to send a message. Aglets [5, 6] attempts to provide location-transparency via Aglet proxy, but the system does not provide APIs to support Aglet tracking. To obtain the receiver's proxy, Aglets programmers must implement tracking mechanism by themselves.[1]

The Mogent system [7] proposed a reliable inter-agent communication algorithm. All the agents need to register their locations with their homes. Before sending a message to another agent, the sender agent queries the recipient's current address from the target agent's home host. If the target agent is currently moving across the network, the reply to the location query is pending until the target agent registers its new location. Before an agent can move, it needs to ask for permission from the home host. If there are messages on their way targeting at the agent, the agent need to wait until these messages arrive. It is the responsibility of the agent home to synchronize the migration of the agents and the message passing. In this way, reliable message delivery can be guaranteed and no message forwarding is needed. However, the algorithm depends so much on the agent home that the agent cannot move and communicate if their home is down or disconnected.

Murphy and Picco [8] present a broadcast-based mobile agent communication scheme which is similar to a distributed snapshot. The scheme guarantees transparent and reliable inter-agent communication, and can also provide multicast communication to a group of agents. However, to search for the message recipient, it requires to contact every node in the network that has been visited by at least one agent, and thus generates an amount of traffic that is comparable to a broadcast. Same as in the "searching" scheme mentioned above, the traffic overhead is unaffordable when there are a large number of hosts and agents in the network.

In [9] a hierarchical infrastructure is proposed to name agents and to route messages. All the hosts in the network are organized into a tree. The agent moves along the nodes of the tree and on every node leaves a pointer to the next one in the path. Messages are forwarded along the same path according to these pointers. However, the hierarchy cannot always be easily constructed, especially in the Internet environment. Instead of sending the messages or agents directly to their targets, unnecessary hops need be taken along the tree. Besides, under this scheme, messages may be missed by their recipient agents and need to keep chasing the recipient.

A resending-based TCP-like message delivery mechanism, called MStream, for mobile agent communications is introduced in [10]. The mechanism assumes that losses and failures are possible in the network. MStream is the communication end-point that can be moved from host to host. When an MStream moves, a *Location Manager* will broadcast its new location to all other MStreams. If a message is sent to an outdated address of the target MStream, it will be retransmitted several times before the sender sends it to the Location Manager to be forwarded to the new destination. The paper does not mention how to avoid multiple forwarding for highly mobile agents.

Our mailbox-based algorithm adopts a hybrid approach combining the registration and forwarding schemes. It realizes location-transparency and ensures the message delivery. Under this communication scheme, most of the messages are sent to their recipients directly and others are forwarded at most once before they reach the

[1] In the latest release of Aglets system, namely ASDK V1.1 Beta3 [6], the MASIF interface MAFFinder was implemented over Java RMI. With the cooperation of the finder and the aglet server, the proxy of a remote aglet can be obtained in a location transparent way. However, there is no guarantee for message delivery. If the target aglet moves away, the message sending procedure will fail and an AgletNotFoundException will be thrown.

receiver agents. Besides, the movement of agents can be separated from that of their mailboxes, thus, by deciding adaptively when to move the mailbox to its owner agent, we can reduce the traffic overhead greatly. The details of our algorithm will be discussed in the next section.

3. The Adaptive Mailbox-Based Routing Algorithm

As we have discussed in Section 2, the home registration and forwarding method adopted by Mobile IP cannot be borrowed blindly by mobile agent systems. One possible solution to reducing the dependence of agent communication on agents' home hosts is to decentralize the role of the home host in locating a target agent. The responsibility of agent tracking is distributed to all the hosts (called "past hosts" hereafter) on the path traveled by the migrating agent. The location of the migrating agent is kept by all the past hosts. Once the agent has arrived at a new host, it multicasts its location to all the past hosts. This can reduce the agent tracking cost. However, the location updating cost can be too much to be acceptable if the agent visits a great number of hosts during its life cycle. As a matter of fact, in many applications, agents migrate from one host to another without communicating with others. In these cases, it is superfluous for agents to multicasting their locations to all past hosts. If we can find an adaptive way to avoid the superfluous address registration, the traffic overhead will be decreased considerably. As we will see, our mailbox-based algorithm can accomplish this goal by detaching the mailbox from its owner whenever possible.

3.1 System Model and Assumptions

In our system model, we assume that mobile agent communication is largely based on asynchronous messages. This is reasonable because, when mobile agents roaming the Internet, it is undesirable that two agents use synchronous communication to talk to each other [13], due to the large and unpredicted delays on the Internet, which can easily become several seconds.

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. Two modes of message delivery can be supported: *Push* and *Pull*. In the push mode, messages stored in the mailbox will be delivered to the mobile agent, while with the pull mode, the agent fetches messages from its mailbox any time it decides to do so. In this paper, we use the pull mode. A mobile agent is automatically initialized to check its mailbox whenever necessary. If the mailbox contains any messages, these messages are delivered. Otherwise, either a synchronous or an asynchronous 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).

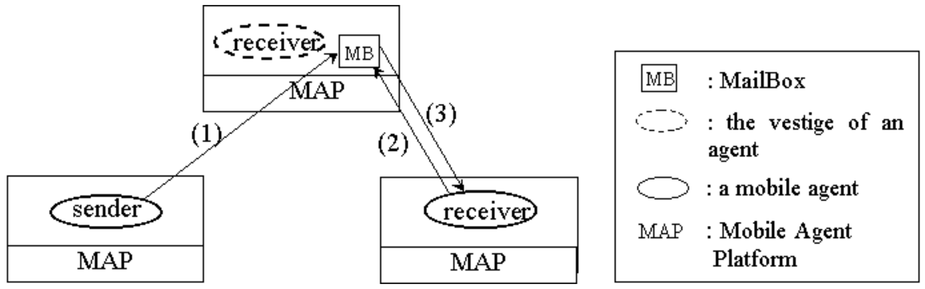

Fig. 1. Receiver and its mailbox residing at different hosts

As shown in Figure 1, in our algorithm, the mailbox can be detached from its owner agent in the sense that an agent and its mailbox can reside at different hosts. An agent can migrate to a new host while leaving its mailbox at a previous past host. When an agent M_A sends a message to another agent M_B , M_A sends the message to the host where M_B's mailbox currently resides (Step (1) in Figure 1). The agent M_B sends a request to its mailbox to fetch message (Steps (2) and (3) in Figure 1). Since the location of mailbox is unchanged, location updating is avoided and a considerable message passing cost can be saved. In location updating, the meaning of "past hosts" is also changed. It no longer refers to the hosts on the path of the migrating agent, but the hosts where the mailbox once resided, which may be much fewer in number. Thus the number of hosts that keep the agent's location information is decreased and the overhead of location updating is further reduced.

An *address table* is maintained in every host to record current addresses of mailboxes that have ever resided at this host. A "valid" tag is associated with every entry of the table to show whether the corresponding mailbox address is outdated. Another field in an entry is a *blocked message queue*, which is used to temporarily keep the messages sent to the corresponding mailbox if the "valid" tag is false, i.e. the mailbox is moving on its way to a new host.

We assume that our algorithm is built on a set of low-level location-dependent communication mechanisms, which can be directly implemented above standard network protocols using asynchronous and point-to-point messages [14]. It is assumed that these mechanisms abstract away the network failure for our high level location-independent algorithm. They also maintain the FIFO order of message delivery, which is critical to the proper execution of our algorithm. As Murphy and Picco indicated in [8], their algorithm also requires the FIFO property, which can be implemented straightforwardly in a mobile agent server by associating a queue that contains messages that must be transmitted to a remote server.

3.2 The Algorithm

The algorithm works in two phases, location-updating and message-routing. In the *location-updating* phase, if the agent decides to migrate with its mailbox, it will first de-register its mailbox address and then reregister the new address with all the past hosts after it reaches the new destination host. In the *message-routing* phase, messages are sent to the recipient's address cached by the sender. If the recipient has been moved to another host, the messages will be forwarded to the current address. The algorithm is presented more formally in pseudo code in the rest of this section.

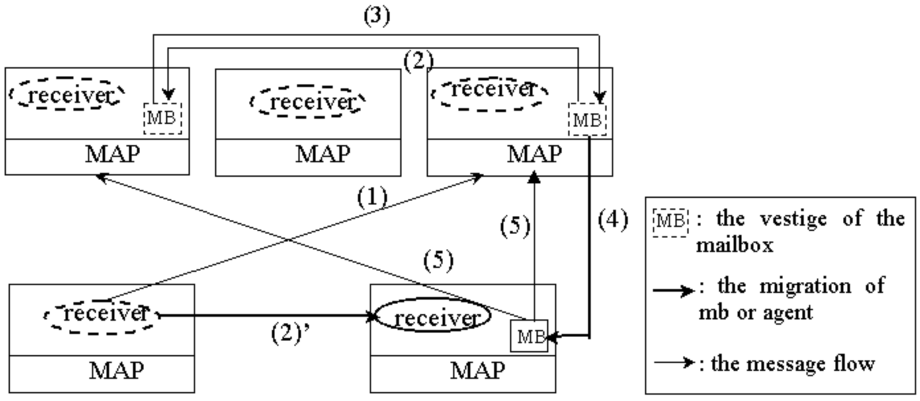

Fig. 2. Mailbox migration and registration

Location Updating. Before moving, the agent determines whether to migrate its mailbox to the new host. It sends a "MVMB" message to its mailbox host if it decides to do so. The "MVMB" message contains the address of the destination host that the agent is to migrate to (Step (1) in Figure 2). The pseudo code of this operation is shown in the function OnMigration_Agent().

```
OnMigration_Agent(){ //executed by agents before moving
  if(fetchMailbox()){
    // the agent decides to go with its mailbox
    String nextAddr = itinerary.getNextHost();
    sendMsgToMailBox("MVMB", getMBAddress(), nextAddr);
        //the underlying location-dependent primitive
  }
  migrateTo(nextAddr);
      //migrate to the target host, Step (2)' in
Figure2
}
```

On receiving the "MVMB" message, the mailbox host executes the function ProcessMVMBMsg_MB(). It sends "DEREGISTER" messages to all the hosts on its path, including the local host (Step (2) in Figure 2). After it has collected the "REPLY" messages from all the hosts, or when time out, it migrates the specified mailbox to the destination host (Step (4) in Figure 2).

```
ProcessMVMBMsg_MB(msg){   //executed by mailbox
  path = getPath();
  for(every host on the path) //including the local
host
```

```
    SendMsgToMAP(“DEREGISTER”, everyHost, localHost);
  wait until all REPLY msgs from these hosts arrive or
  time-out;
  targetHost = msg.getContent();
          //get the address of target host
  migrateTo(targetHost);
}
```

On arriving at the new host, the agent starts executing the function `OnArrival_Agent()` by checking whether its mailbox has been moved to the new host with it. If not, it does not need to register its new address. Otherwise it sends a “`REGISTER`” message to every past host where its mailbox has resided (Step (5) in Figure 2).

```
OnArrival_Agent()  //executed by the agent
{
  if (migrated without mailbox)
    return;      //do nothing
  setMBAddress(localAddress);
       //update the address of its mailbox
  append the localhost into its mailbox’s path;
  for (every host on the path of the mailbox)
    SendMsgToMAP(“REGISTER”, everyhost, localAddress);
}
```

The mobile agent platform (MAP) in a host is responsible of processing all the control messages. Its operation is illustrated in `MessageProcessing_MAP()` shown below.

```
MessageProcessing_MAP(msg)  //executed by MAP
{
  switch(msg.getKind()){
    case DEREGISTER:
      AddressEntry entry =
                 addressTable.getAddr(msg.getSender());
      entry.VALID = false;
      sendMsgToMailBox(“REPLY”, msg.getContent(),
                           null);    //step (3) in
Figure 2.
    case REGISTER:
      AgentID sender = msg.getSender();
      AddressEntry entry =
addressTable.getAddr(sender);
      if (entry == null){
      // REGISTER msg is from the local host, create a
      //new entry in address table for sender’s
address.
        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
  }
```

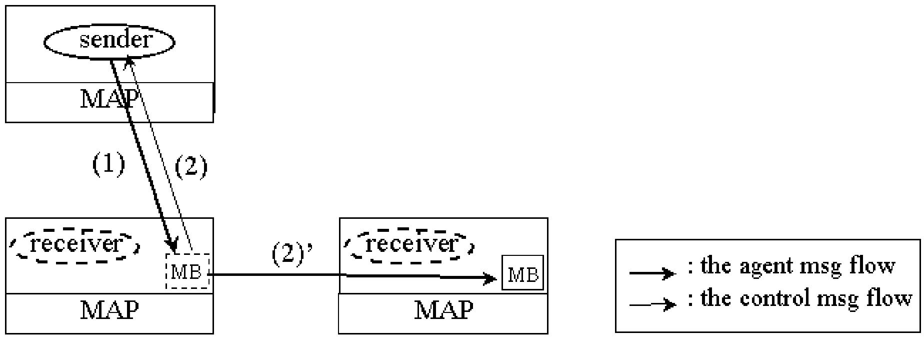

Fig. 3. Message forwarding after mailbox leaves

Message Routing. Figure 3 illustrates the message forwarding process. Suppose agent M_A wants to send a message to agent M_B. Referring to the function `SendMessage_Agents()`, M_A first checks whether the address of M_B's mailbox has been cached locally. If so, it sends the message to the cached address. Otherwise it sends the message to M_B's home host (Step (1) in Figure3).

```
  SendMessage_Agents(Message msg){
                //executed by mobile agent
    if (the receiver's address is in cache){
      sendMsgToMAP("AGENTMSG", address in cache, msg);
    }else{
      String homeAddress = msg.getReceiver().getHome();
      SendMsgToMAP("AGENTMSG", homeAddress, msg);
    }
  }
```

When a host receives a message destined to an agent M, it checks whether M's mailbox is currently resided locally. If so, it inserts the message to M's mailbox directly. Otherwise the message is forwarded to the M's current address recorded in the local address table. See the function `MessageRouting_MAP()` for details.

```
MessageRouting_MAP(agentMsg){
  AgentID receiver = the target agent of agentMsg;
  if (the receiver's mailbox is local){
    insert agentMsg to the mailbox;
  }else{
    AddressEntry entry =
             addressTable.getAddress(receiver);
    if (entry.VALID){
      sendMsgToMAP("AGENTMSG", entry.address,
agentMsg);
                                   //Step (2) in Figure 3
      sendMsgToMAP("UPDATE",agentMsg.getSender(),
                   entry.address); // Step (2)' in
figure3
    }else{  //valid tag is false: receiver is migrating
      entry.blockQueue.insert(agentMsg);
             //insert the message to the block queue;
    }
  }
}
```

Agent M_A caches the new address of agent M_B contained in the incoming "UPDATE" message. Next time when M_A sends messages to M_B, it will send the message to this new address.

3.3 Properties of the Algorithm

Before proving the correctness of our algorithm, we give a formal definition of the path of an agent mailbox.

Definition. Path(*mb*) is a sequence $<h_0, h_1, \ldots h_i, \ldots h_n>$ of hosts where the mailbox *mb* has been inhabited. For all h_i, h_j in the path, h_i is visited by *mb* earlier than h_j if $0 \leq i < j \leq n$. The host h_0, is the home of *mb*'s owner agent, and h_n is the host where *mb* is currently located.

Theorem1 shows that our algorithm can provide location transparency from the point of view of a sender agent.

Theorem1. With the proposed algorithm, 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 it has cached the receiver's address. If there is the receiver's address in its cache, it will send the message to this address without caring about whether it is outdated. Otherwise it will get the receiver's home address from its 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. **QED**

The following lemmas and theorem2 show the effectiveness of our algorithm, i.e., it can guarantee the delivery of messages. Besides, the message will be forwarded at most once so that it will not chase its recipient.

Lemma1. Suppose a mailbox *mb* is currently located at host h_j and Path(*mb*) is < h_0, h_1, …h_i ,… h_j >. For all h_i in Path(*mb*), if h_i was not down, *mb* must have received the REPLY message from h_i before it leaves h_j.

Proof. If *mb* wants to leave h_j, it must send DEREGISTER messages to all the hosts in Path(*mb*). Since h_i is not down and it is assumed that the underlying location dependent communication mechanisms can shield the network failure, h_i will at last receive the DEREGISTER message and send a REPLY message to *mb*. According to the function *ProcessMVMBMsg_MB(), mb* cannot leave h_j until it collects all the REPLY messages from all the hosts in Path(*mb*) or when time out. Since we need not worry about network failure, we can conclude that the REPLY message from h_i will arrive at *mb* before *mb* leaves. **QED**

Lemma2. For all h_i in Path(*mb*), the valid tag of *mb*'s address in the address table is true only if the address reflects exactly the current location of *mb*, i.e., the address kept in the address table is not outdated.

Proof. Suppose the address of *mb* kept in h_i's address table is h_j. If the valid tag is true, from the function *MessageProcessing_MAP()* we can conclude that h_i must have received *mb*'s REGISTER message from h_j, and the DEREGISTER message has not arrived yet. So the REPLY message has not been sent out from h_i. From *lemma1* we know that *mb* is still at h_j and cannot leave until it has colleted all the REPLY messages from hosts in Path(*mb*), including h_i. So the address h_j kept in the address table reflects the current location of *mb*. **QED**

Theorem2. All the messages can be delivered to their recipients' mailboxes by being forwarded at most once.

Proof. Suppose a sender agent *S* is sending a message *m* to a receiver agent *R*., and *R*'s mailbox MB_R is located at host h_j. Let Path(MB_R) be < h_0, h_1, …h_i ,… h_j >. Without loss of generality, we assume that *R*'s address kept in the cache of *S* is h_i and $0 \le i \le j$ (if there is no record of *R*'s address in the cache of *S*, the message will be sent to *R*'s home, which is h_0 in Path(MB_R)).

S will obtain *R*'s address, namely h_i, from its address cache and send the message *m* directly to h_i. When *m* arrives at h_i, 3 cases could happen:

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

Case 2: i < j and h_i has not received MB_R's DEREGISTER *message from h_j.* In this case, *m* will be processed before *R*'s DEREGISTER message. To deliver *m* to *R*, h_i will check its address table and find *R*'s address is h_j and the valid tag is "true". So *m* is forwarded to h_j. Since *m* is processed earlier than *R*'s DEREGISTER message, *m* is forwarded to h_j earlier than the REPLY to the DEREGISTER message. The FIFO property can guarantee that *m* arrives at h_j earlier than the REPLY message. From ***Lemma1*** we can conclude that MB_R cannot migrate to other hosts during the transmission of *m*. After *m* arrives at h_j, it will be inserted into MB_R. So *R* can receive *m* later from MB_R and *m* is forwarded only once.

Case 3: i < j and h_i has received MB_R's DEREGISTER *message from h_j, i.e., MB_R has left for h_{j+1}.* The host h_i checks the valid tag of MB_R's address. If it is "true", the address of MB_R kept in the address table must be h_{j+1} (warranted by *Lemma2*) and *m* is forwarded to MB_R in the same way discussed in the second case. If the valid tag is "false", we can conclude that MB_R is on its way to h_{j+1}. The message *m* will be put into the blocked message queue. It won't be forwarded until MB_R reaches h_{j+1} and its `REGISTER` message arrives at h_i. After the `REGISTER` message arrives, *m* will be forwarded to h_{j+1}. As discussed in the second case, MB_R will not leave during the transmission of *m*, since *m* will arrive at h_{j+1} earlier than the `REPLY` message. Therefore, in this case, *m* is also forwarded only once and *R* can get *m* later from MB_R.

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. **QED**

4. Performance Analysis

In this section we formulate the traffic cost of the location updating and message delivery of the proposed algorithm in terms of the number of messages required. To simplify the problem, we ignore the differences in the distances between hosts. Here we introduce 3 decision variables: x, x_0 and x_1, which are defined as follows:

$$x = \begin{cases} 1 & \text{The agent has left and messages should be forwarded to its new location.} \\ 0 & \text{Otherwise} \end{cases}$$

$$x_0 = \begin{cases} 1 & \text{The agent will move with its mailbox.} \\ 0 & \text{Otherwise} \end{cases}$$

$$x_1 = \begin{cases} 1 & \text{The agent and its mailbox reside at different hosts.} \\ 0 & \text{Otherwise} \end{cases}$$

The location updating cost and message delivery cost can be formulated as follows:

$$C_{update} = x_0\left(x_1 C_{ctrl} + (3N_{mb} + 1)C_{ctrl}\right) \tag{1}$$

$$C_{delivery} = C_{msg} + x\left(C_{msg} + C_{ctrl}\right) + x_1\left(C_{msg} + C_{ctrl}\right) \tag{2}$$

where C_{ctrl} and C_{msg} denote communication traffic of a control message and an agent message, respectively. Since control messages, such as "`MVMB`", "`REGISTER`" and "`UPDATE`" messages may be much shorter in length than agent messages, they should not be counted in the same way. N_{mb} denotes the number of hosts in Path(*mb*). As discussed in section 3, when an agent is leaving and decides to take its mailbox along with it to the new host (x_0 is 1), it sends "`MVMB`" message to its mailbox if the mailbox does not reside at the same host (x_1 is 1). The cost is denoted by the first term in parentheses of Formulation (1). Then it sends "`DEREGISTER`" messages to the N_{mb} hosts in Path(*mb*), collects N_{mb} "`REPLY`" messages and sends $N_{mb}+1$ "`REGISTER`" messages on arriving at the next destination (the second term in the parentheses of Formulation (1)).

When an agent sends a message to another agent, it sends the message to the receiver's location cached in its address table (first term in Formulation (2)). If the sender's knowledge about the receiver's location is out of date, the message has to be forwarded to the new location and the "UPDATE" message is returned to the sender. The cost is denoted by the second term of Formulation (2). If the receiver wants a message from its mailbox and it resides at a different host with its mailbox, it sends a control message to its mailbox and the mailbox returns the corresponding message to it (the third term of Formulation (2)).

From these two formulae, we can see that if an agent migrates without taking its mailbox (x_0 is 0), the location updating cost is 0. By deciding the value of x_0 to adjust location updating cost, our algorithm works in an adaptive way. There are two extreme cases.

1. The first one is that the mailbox never moves during the life cycle of its owner agent. In this condition, the mailbox always resides at the home of its owner. Messages is sent to the receiver's home and the receiver get the messages from its home. It is similar to the home registration and forwarding method. In this condition, the location updating cost is 0. But the message delivery cost is expensive ($2C_{msg}+C_{ctrl}$) and the home must be kept linked during agents' life cycle.
2. The other extreme case is that the mailbox is bound to its owner and they are always migrating together. Under this condition x_1 is always 0 and the message delivery is less expensive, but the location updating cost C_{update}, as shown in Formulation (1), is $(3N_{mb}+1)C_{ctrl}$ since x_0 is always 1.

To save the totle traffic cost, which includes the location updating and message delivery cost, compromise must be made between the two extremes according to specific applications. To determine whether moving with its mailbox or not, an agent can consider factors such as the number of messages it will receive in the next host and the distance between its next destination host and the current location of its mailbox. If an agent seldom receives messages from others in the next host, it doesn't need to take its mailbox to the new host. On the other hand, if an agent will receive messages frequently from others and the next host is far away from the host its mailbox currently resides at, it will be expensive to leave the mailbox unmoved and to fetch messages from the remote host. In this case the agent should migrate to the new host together with its mailbox.

5. Simulations and Observations

To evaluate the performance of the algorithm as formulated in Section 4 under various conditions, our algorithm is implemented in a simulated mobile agent environment. In our simulations we assume that the traffic cost for every agent message (C_{msg}) is 1 unit and the control message cost (C_{ctrl}) is one fourth that of C_{msg}. The cost is recorded automatically each time a message is sent out. We also assume that whenever an agent migrates, it will hop to a host different to all the hosts it has ever visited.

The first senario of our simulation involves one agent only. It migrates from one host to another without communication. The cost of the register, deregister and reply messages is recorded. We use the term "migration ratio" to denote the ratio of the mailbox migration number to the agents migration number. Figure 4 shows the average traffic costs per agent migration under different migration ratio and total hops numbers. We can see that as the migration ratio increases, the average traffic cost increases quickly. Since N_{mb} increases as the migration ratio rising, this result can be predicted from Formula (1) in Section 4.

As we have discussed in Section 4, message delivery is expensive if the mailbox stays at the agent's home host. Then what's the relation between the cost of the message delivery and the mailbox migration ratio? Result shown in Figure 5 can answer this question. This time a sender keeps sending messages and the interval of two messages is randomly set. The total number of messages is 600. The receiver receives several messages on every host and migrates to another. The migration intervals are set to 30, 10 and 6 messages respectively and the corresponding traffic costs under each interval and each migration ratio are recorded. As we can see, the average delivery cost per agent message is the highest when the migration ratio is 0, i.e., the mailbox stays at its owner's home all the time during the agent's life cycle. The cost decreases as the migration ratio increases. It reaches the lowest point when the migration ratio is 1. The mailbox is bound with its owner under this condition and the agent can get the message directly from its mailbox. We can also oberve that under the same migration ratio, the average delivery cost is a little higher when the move interval becomes shorter. The result is reasonable because the more frequently the mailbox migrates, the more messages must be forwarded.

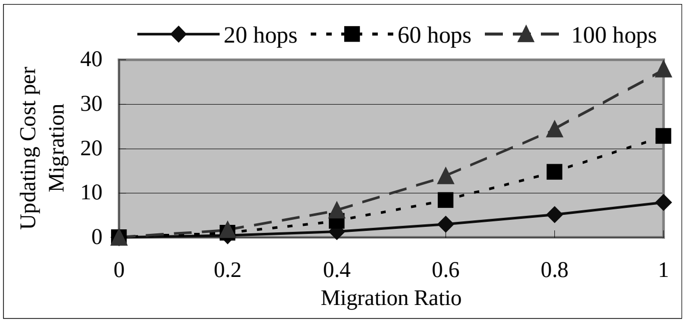

Fig. 4. Effect of Migration Ratio on the Updating Cost

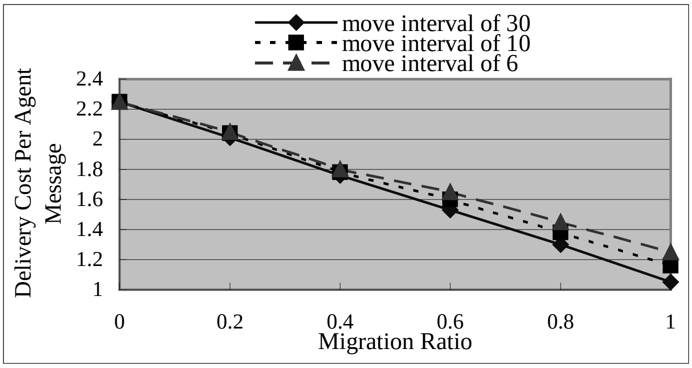

Fig. 5. Effect of Migration Ratio on the Message Delivery Cost

From figure 4 and 5 we can observe that the average location updating cost and message delivery cost varies in opposite directions as the migration ratio rises. There must be an optimal point on which the total traffic cost is the lowest. We introduce a sender agent and a receiver agent in our third simulation senario. As in the second one, the sender keeps sending messages at random intervals. The moving intervals of the receiver are also randomly set. They vary from 0 to 19 messages (inclusive). Whether the receiver migrates with its mailbox or not is determined by the moving interval and a pre-set threshold value. Before moving, the receiver estimates the number of messages it will receive in the next host (the number is generated by a random number generator in our simulation). If the number is less than the threshold value, the receiver will migrate without its mailbox. Otherwise the mailbox will be taken along. The total number of messages are set to 100, 300 and 500 respectively. The average total cost is shown in figure 6. We can see that the costs are higher in two extreme conditions. Since the moving intervals distribute evenly between 0 and 19, it reaches the lowest point when the threshold value is almost half of the highest interval, i.e. 8 or 12 messages. From this example we can conclude that by determining properly whether the agent will take its mailbox along, the communication overhead can be decreased considerablely.

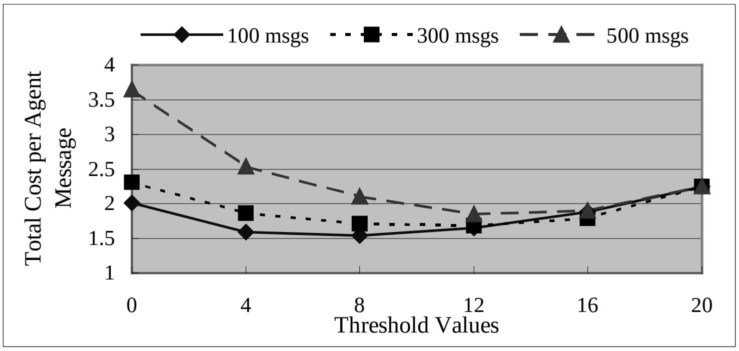

Fig. 6. There is an optimal point if the threshold value is properly set.

6. Conclusions and Future Work

We have proposed a mailbox-based approach to designing mobile agent communication protocols. In our design, a mobile agent and its mailbox can be separated in the sense that they can reside in different hosts. An agent can migrate to a new host while leaving its mailbox in a previously located host. This helps overcome the high location updating cost. An agent can decide whether to take with its mailbox along with it according to the number of messages in the network and its movement area. One of the two extreme cases of our algorithm is similar to the home forwarding scheme. If the decision is properly made, as shown by our simulation results, the lowest total traffic cost which is less than that of both extremes can be obtained.

A mailbox-based protocol still follows the registration-and-forwarding scheme but can be made to overcome many of the drawbacks. It can route the messages in a reliable and location transparent way. By forwarding the message at most once, the protocol avoids the problem that messages may chase forever its target agent that migrates frequently. Unlike the home registration method used in mobile computing, e.g., Mobile IP, the mailbox-based protocol decentralize the role of the home host and reduce the reliance on it, so that mobile agent's capability of supporting disconnected operations can be realized in real. Furthermore, the protocols are adaptive and can decrease the overhead of location registration by deciding whether a mobile agent will migrate with its mailbox.

Although in our algorithm the dependence and workload of the agent home have been distributed to all the hosts on the agent migration path, the agent home still has to work as a location server, especially when it's the first time that an agent sends a message to another born on it. To let the algorithm work even when some hosts on the agent migration path including the agent home are down or disconnected, one dedicated location server can be introduced in our framework. Since it is queried only when disconnection or system failure occurs, the dedicated location server will not be the performance bottleneck as a central one. Security issues should also be considered in our future work. Because the sender agent may accept "`UPDATE`" message from any host on the receiver migration path as discussed in Section 3, it is vulnerable to the address spoofing attack. Specifically a Bad Guy could simply send a bogus "`UPDATE`" message to the sender and cause all the messages to be sent to the Bad Buy instead of the receiver. To prevent such attacks, authentication schemes must be adopted.

References

1. M. Straßer, J. Baumann, F. Hohl, Mole - A Java Based Mobile Agent System In: *Special Issues in Object-Oriented Programming, Workshop Reader ECOOP'96,* p327-334, dpunkt.verlag, 1996
2. A. Pham and A. Karmouch, "Mobile Software Agents: An Overview", *IEEE Communications magazine,* Vol. 36, No. 7, July 1998, pp.26-37
3. Jiannong Cao, G.H. Chan, W. Jia, and T. Dillon, "Checkpointing and Rollback of Wide-Area Distributed Applications Using Mobile Agents", *Proc. IEEE 2001 International Parallel and Distributed Processing Symposium (IPDPS2001) (IEEE Computer Society Press),* April 2001, San Francisco, USA
4. Timothy K. Shih, "Agent Communication Network - A Mobile Agent Computation Model for Internet Applications", *Proc. 1999 IEEE Int'l Symp on Computers and Communications*, 1999. pp.425-431
5. D.B.Lange and M.Oshima, Programming and deploying Java mobile agents with Aglets. *Addison-Wesley,* 1998
6. http://www.trl.ibm.com/aglets/index.html
7. Tao Xianping, Jian Lu, et al. Communication Mechanism in Mogent System. In: *Journal of Software* 2000, 11(8): 1060~1065, P.R. China
8. Amy Murphy and Gian Pietro Picco, Reliable Communication for Highly Mobile Agents. In: *Agent Systems and Architectures/Mobile Agents (ASA/MA)'99*, pages 141-150, October 1999
9. Van Belle, W., Verelst, K., D'Hondt, T., Location transparent routing in mobile agent systems merging name lookups with routing. In: *Proceedings of the Seventh IEEE Workshop on Future Trends of Distributed Computing Systems (pp. 207-212).* 1999

10. Mudumbai Ranganathan, Marc Bednarek, and Doug Montgomery, A Reliable Message Delivery Protocol for Mobile Agents. In: *Agent Systems, Mobile Agents, and Applications, Lecture Notes in Computer Science, No. 1882, Springer-Verlag (D),* pp.206-220, September 2000
11. Charles. E Perkins. *IP Mobility Support* RFC2002. October 1996
12. D. Chess, C. Harrison, A. Kershenbaum. Mobile Agents: Are They a Good Idea? In: *Mobile Object Systems: Towards the Programmable Internet, Lecture Notes in Computer Science, No 1222, Springer-Verlag (D), pp.25-45*, February 1997
13. K. Verelst, "A Study of Communication Models for Mobile Multi-agent Systems", *Ph.D Thesis*. Department of Informatics, Vrije University of Brussel, Brussels, Belgium. May 1999
14. Peter Sewell, Pawel T. Wojciechowski and Benjamin C. Pierce. Location-Independent Communication for Mobile Agents: a Two-Level Architecture. *Technical Report 462,* Computer Laboratory, University of Cambridge, April 1999