AJIS vol 9 no. 1 September 2001

SCALABLE FAULT-TOLERANT LOCATION MANAGEMENT SCHEME FOR MOBILE IP

JinHo Ahn, Sung-Gi Min and ChongSun Hwang
Dept. of Computer Science and Engineering, Korea University
5-1 Anam-dong, Sungbuk-gu, Seoul 136-701, Republic of Korea

ABSTRACT

As the number of mobile nodes registering with a network rapidly increases in Mobile IP, multiple mobility (home or foreign) agents can be allocated to a network in order to improve performance and availability. Previous fault-tolerant schemes (denoted by PRT schemes) to mask failures of the mobility agents use passive replication techniques. However, they result in high failure-free latency during registration process if the number of mobility agents in the same network increases, and force each mobility agent to manage bindings of all the mobile nodes registering with its network. In this paper, we present a new fault-tolerant scheme (denoted by CML scheme) using checkpointing and message logging techniques. The CML scheme achieves low failure-free latency even if the number of mobility agents in a network increases, and improves scalability to a large number of mobile nodes registering with each network compared with the PRT schemes. Additionally, the CML scheme allows each failed mobility agent to recover bindings of the mobile nodes registering with the mobility agent when it is repaired even if all the other mobility agents in the same network concurrently fail.

INTRODUCTION

A Mobile IP based system extends an IP based distributed system to support mobility of nodes by providing Mobile Nodes (MNs) with continuous network connections while changing their locations [Ghosh (1998), Johnson (1994), Perkins (1996b), Solomon (1998)]. In other words, it transparently provides mobility for nodes while backward compatible with the current IP routing scheme by using two kinds of Mobility Agents (MAs), home agent and foreign agent. In the system, each home agent must maintain information about the current care-of address of each MN registering with the home agent and forward packets, destined to the MN, to the care-of address for the MN. Each foreign agent should offer a care-of address for each visiting MN and forward packets to the MN. If each MN uses a collocated care-of address, it must have the same function of a foreign agent.

As mobile computing is increasingly gaining popularity, MAs should serve a large number of MNs. Thus, they may be single points of failure and potential performance bottlenecks. Especially, if a home agent fails, all the MNs served by it can't communicate with other nodes. The problems can be solved by allowing multiple MAs to be assigned to the same network. Previous fault-tolerant schemes (denoted by PRT schemes) [Ghosh (1998)] to mask failures of MAs in Mobile IP use passive replication techniques. In the PRT schemes, each MA must always maintain bindings of all MNs registering with its network. Moreover, if each MA receives a registration request message from a MN in these schemes, it should process the message, forward the message to its peers and then wait until it has received all the acknowledgement messages from them. Thus, the PRT schemes result in high failure-free overhead during registration process if the number of MAs in the same network increases and are not scalable to the number of MNs registering with each network. Log-based rollback recovery schemes, which use checkpointing and message logging techniques, are inexpensive during failure-free operation compared with the schemes using passive replication techniques [Alvisi (1993)]. In this paper, we present a new fault-tolerant scheme (denoted by CML scheme) for MAs using checkpointing and message logging techniques. The CML scheme reduces the failure-free latency even if the number of MAs in a network increases, and improves scalability to a large number of MNs registering with each network compared with the PRT schemes and provides fast recovery for taking over failed MAs. Additionally, the CML scheme allows each failed MA to recover bindings of the MNs registering with the MA when it is repaired even if all the MAs in the same network fail.

The rest of the paper is organized as follows. In section 2, we describe the overview of Mobile IP based Systems and problems of the previous fault-tolerant schemes respectively. In section 3, we present our fault-tolerant scheme, explain the description of the scheme in details and prove its correctness. Section 4 compares our scheme with others and then, in section 5, we conclude this paper.

PRELIMINARIES

System Model

In Mobile IP, each MN must have a unique home address and a home agent on its home network. If a MN moves from its home network to a foreign network, the current location of the MN is identified as a care-of address (COA), and the mapping between the home address and the COA of the MN is called a binding [Solomon (1998)]. Whenever the MN enters into a new foreign network, it must register by sending a registration request message to a foreign agent (FA) in the new network. The request message includes the home address of the MN and the IP address of its home agent (HA). Then, the FA sends a registration request message to the HA. The message contains the home address and COA of the MN, the IP address of the HA, a registration lifetime and an identifier which uniquely identifies the registration request message. When the HA receives the request message, it updates the binding of the MN, and then sends a registration reply message back to the FA. When the FA receives the reply message, it updates its own table and forwards the message to the MN. A MN in a foreign network can obtain a COA in one of two ways as follows. First, if there is a FA in the foreign network, the MN will attempt to obtain a care-of address from the agent by using an agent discovery protocol [Johnson (1994), Solomon (1998)]. In this case, the IP address of the FA is used as the COA of the node. Second, if there is no FA in the network, the MN can obtain a collocated COA in the network using a DHCP-like protocol [Droms (1993)]. If a correspondent node (CN), which may be a MN or a FN, sends a packet to a MN, this packet is routed to the home network of the MN when the MN is in the network. When the MN is not in its home network, its HA intercepts, encapsulates and then tunnels the packet to the FA in the foreign network where the MN is currently located [Perkins (1996a), Perkins (1996c), Simpson (1995)]. Then, the FA de-tunnels the packet to the MN. If the MN currently uses a collocated COA, de-capsulation of the packet must be carried out by the MN rather than the FA. However, this triangle routing scheme may be inefficient because the messages, destined to each MN, should be first routed to its HA. In order to solve the problem, the route optimization scheme [Perkins (1996d)] was proposed in which the messages are routed directly to the COA of the MN. In this scheme, each CN maintains a binding cache containing the COAs of the communicating MNs. The drawback of the scheme is that it forces the CN to be aware of mobility of the MNs. Each FA or HA is connected by a fixed wired network, which provides reliable FIFO delivery of messages. Each MN can directly communicate with its local FA or HA via a reliable FIFO wireless network only if the MN is in the network covered by the mobility agent. We assume that nodes, including FAs, HAs, other FNs and MNs, fail, in which case they lose the contents of their volatile memories and stop their executions, according to the fail stop model [Schlichting (1985)]. Multiple FAs or HAs can be allocated to each network in order to serve a large number of MNs on the network like in Figure 1 [Binkley (1997)]. Separate nodes or a single node on a network may have the function of HA and FA. Similarly, the existing IP routers or separate nodes on a network may implement either function or both.

We assume that the communication network is immune to partitioning and there is a stable storage that every MA can always access that persists beyond processor failures, thereby supporting recovery from failure of an arbitrary number of processors [Elnozahy (1992)]. The execution of each MA is piecewise deterministic [Elnozahy (1999), Strom (1985)]: At any point during the execution, a state interval of the MA is determined by a non-deterministic event such as delivering a registration request message. The *k*-th state interval of a MA *p*, denoted by si_p^k ($k > 0$), is started by the delivery event of the *k*-th registration request message *m* of *p*, denoted by $dev_p^k(m)$. Let *p*'s state, $s_p^i = \{si_p^0, si_p^1, \ldots, si_p^i\}$, represent the sequence of all state intervals up to si_p^i. Therefore, given *p*'s initial state, si_p^0, and the non-deterministic events, $[dev_p^1, dev_p^2, \ldots, dev_p^i]$, its corresponding state s_p^i is uniquely determined. All information needed for replaying the delivery event of a message during recovery is called *determinant* of the event.

Definition 1. si_p^i is *stable* if a determinant of $dev_p^i(m)$ is saved on stable storage.

Definition 2. si_p^i is *volatile* if it is not stable.

Definition 3. s_p^i is *recoverable* if the system has sufficient information for replaying its failure-free execution up to si_p^i in any future failures.

Lemma 1. *si_p^i is stable if s_p^i is recoverable.*

Proof: We prove this lemma by contradiction. Assume that si_p^i is unstable, i.e., volatile if s_p^i is recoverable. *p* can replay $dev_p^i(m)$ in any failure by Definition 3. To do so, it has to obtain the determinant of $dev_p^i(m)$ during recovery. The stable storage is the only one that survives in an arbitrary number of failures by the fail-stop model and the property of the storage. Thus, the determinant should have been saved on stable storage in a previous failure free execution. Therefore, si_p^i is stable by Definition 1. Hence, this contradicts the hypothesis.

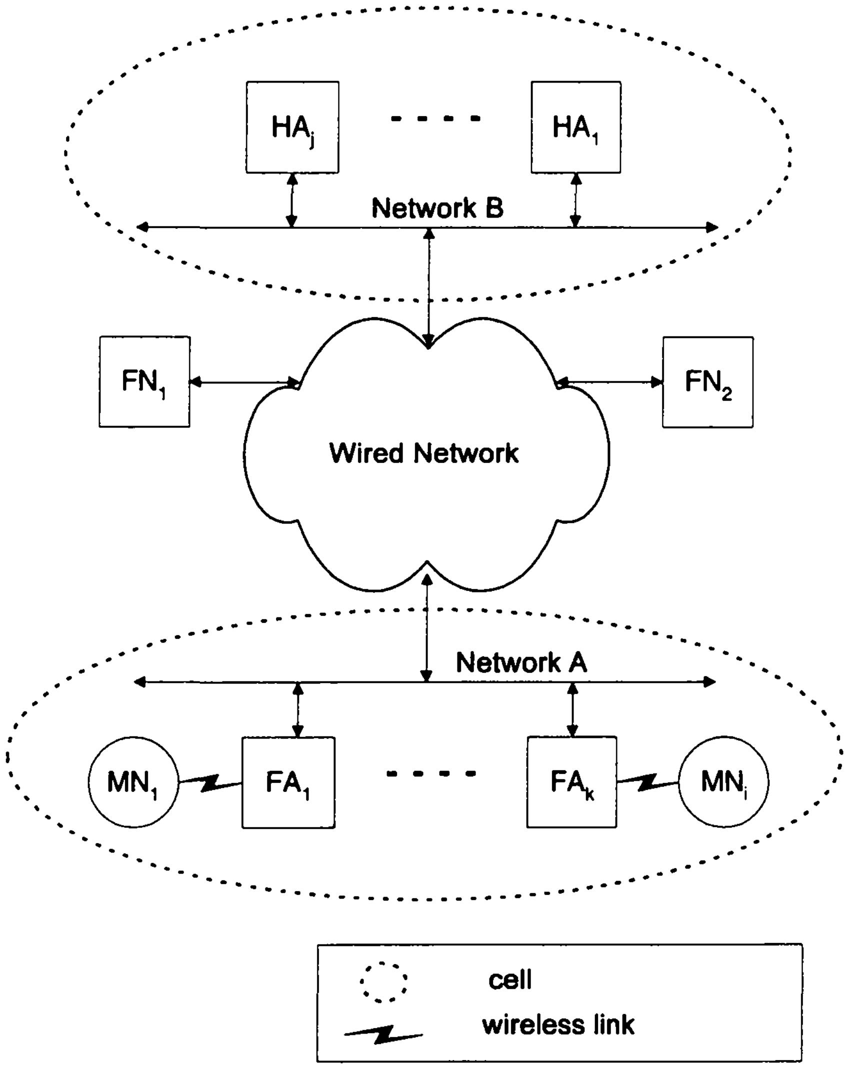

Figure 1 Mobile IP Infrastructure

Lemma 2. s_p^i is *recoverable* if $\forall k$ $(0 \le k \le i)$: si_p^k is *stable.*
Proof. The proof proceeds by induction on i, the index of the state interval of each mobility agent p.
[Base case]
If si_p^0 is stable in this case, s_p^0 is trivially recoverable because si_p^0 is the initial state interval of p and deterministic.
[Induction hypothesis]
We assume that the theorem is true in case that $i = n$.
[Induction step]
If s_p^n is recoverable and there is the determinant of $dev_p^{n+1}(m)$ on stable storage, the theorem is true for p in case that $i = n + 1$. In this step, s_p^n is recoverable by induction hypothesis because $\forall k$ $(0 \le k \le i)$: (si_p^k is stable), and si_p^{n+1} is stable. Therefore, s_p^{n+1} is recoverable.
By the induction, s_p^i is recoverable if $\forall k$ $(0 \le k \le i)$: si_p^k is stable.

Problems of PRT Schemes

PRT Schemes [Ghosh (1998)] to mask failures of multiple MAs in a network use passive replication techniques. In the PRT schemes, each MA in the network must always maintain bindings of all MNs registering with the network. Moreover, if each MA receives a registration request message from a MN in the schemes, it should process the message and forward the message to its peers and wait until it has received all the acknowledgement messages from them. Thus, the PRT schemes result in high failure-free latency during registration process as the number of MAs allocated to a network increases, and are not scalable to the number of MNs registering with each network. Additionally, in the schemes, each failed MA cannot recover bindings of the MNs registering with the MA when it is repaired if all the MAs in the same network fail.

To illustrate the stated problems of the schemes, consider the examples shown in Figure 2 and 5. In Figure 2, there are three MAs, MA_1, MA_2 and MA_3 in network A. If MA_1 receives a registration request message from a MN, named MA_1, it updates MA_1's binding using the message and then forwards the message to MA_2 and MA_3, respectively. After MA_2 and MA_3 have received the message from MA_1, they update MA_1's binding using the message, and then send each an acknowledgement message to MA_1. After MA_1 has received the acknowledgement messages from MA_2 and MA_3, it sends a registration reply message to MA_1. In this figure, we can see that the total number of messages generated per registration request message and the number of messages on the critical path for registration operation in the previous schemes increases as the number of MAs allocated to a network increases. Thus, they result in high latency during registration process.

Additionally, in the PRT schemes, each MA in a network should still maintain bindings of all the MAs registering with the network. For examples, in Figure 3, 6000 MNs register with network A, and MA_1 serves 2000 among them, MA_2 serves 1000 and MA_3 serves 3000. However, MA_1, MA_2 and MA_3 should respectively maintain the bindings of 6000 MNs registering with network A. Therefore, we can see that the traditional schemes are not scalable to a large number of MNs registering with the same network even if there are multiple MAs in the network. Moreover, in the schemes, if MA_1, MA_2 and MA_3 all concurrently fail in Figure 3, each of the three MAs cannot recover bindings of the MNs registering with it from any other MA when it is repaired.

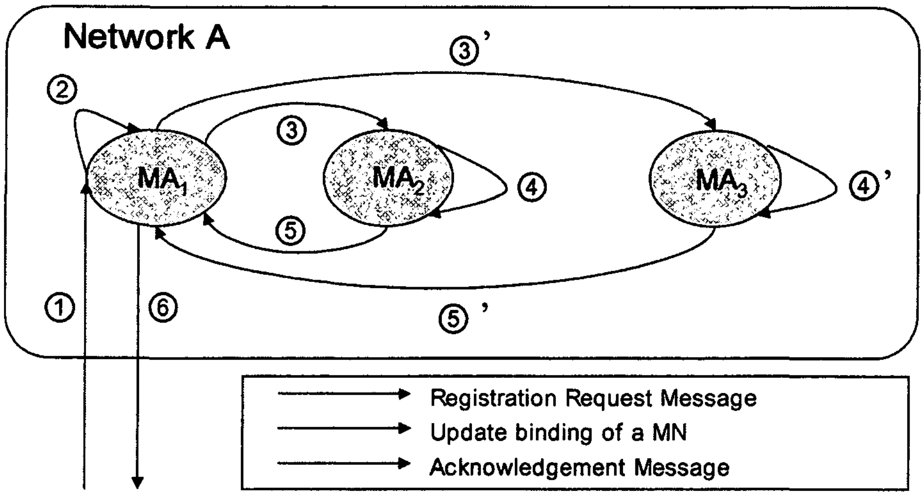

Figure 2 In case of performing a registration process in the PRT schemes

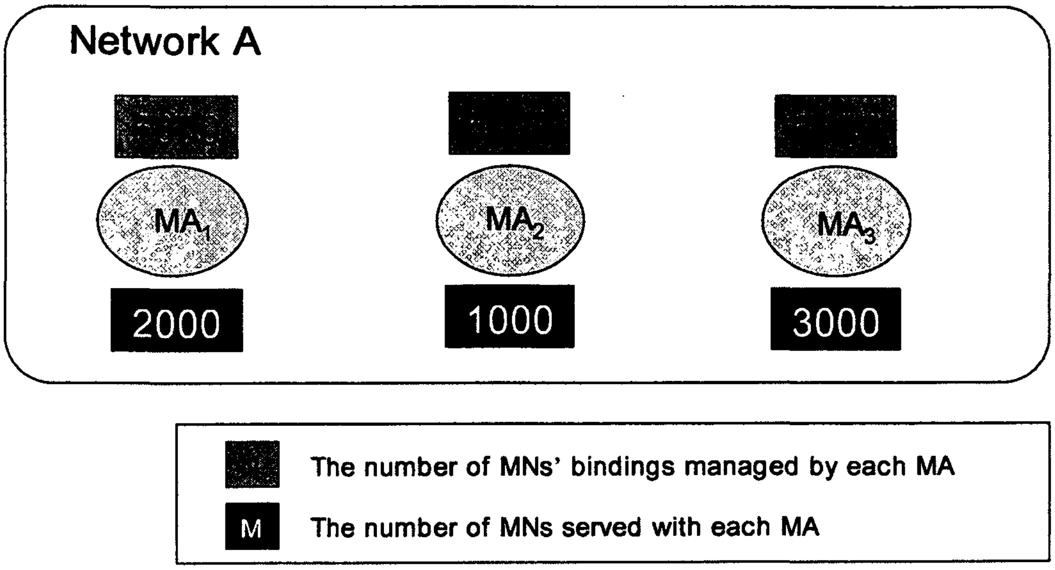

Figure 3 In case of scalability problems of the PRT schemes

THE CML SCHEME

Basic Idea

To solve the stated problems of the PRT schemes in this paper, we present the CML scheme using checkpointing and receiver-based pessimistic message logging techniques. For example, suppose HA_l is the home agent of MN_l and MN_l currently obtains a care-of address from FA_l on the foreign network as in figure 1. In this case, MN_l must send a registration request message to HA_l through FA_l. Receiving the message, HA_l authenticates the message. If the message is invalid, HA_l sends MN_l a registration reply message for rejection. Otherwise, HA_l concurrently performs the following two executions: logging the message into the stable storage and updating the binding of MN_l using the message. After having completed both executions, it sends MN_l a registration reply message for acceptance. This step ensures that even if HA_l fails, one among other home agents on the network, named HA_i, can recover the bindings of all the mobile nodes registering with HA_l by restoring the logged registration request messages from the stable storage and replaying them. Moreover, each home agent should periodically save the bindings of all the mobile nodes registering with it on the stable storage and remove all the logged messages beyond the previous checkpoint from the stable storage. Therefore, the CML scheme can reduce the failure-free overhead compared with the PRT scheme presented in [Ghosh (1998)] because in the CML scheme, each MA need not forward each registration request message to the other MAs and wait for each an acknowledgement message from them. Furthermore, the CML scheme improves the scalability to a large number of mobile nodes registering with each network compared with the PRT scheme because each home or foreign agent need to maintain the bindings of only the mobile nodes registering with it.

In Mobile IP, each MA broadcasts an agent advertisement message via its wired or wireless network interface every few seconds. Therefore, in the CML scheme, each MA detects if other MAs fail or not by monitoring their agent advertisement messages. For example, if HA_l in network B fails in figure 1, live HAs on the network can detect its failure because they may receive no agent advertisement message from it. In the CML scheme, one among the live MAs, namely HA_j, which currently has the minimum number of registering mobile nodes, takes over HA_l. This step ensures that the CML scheme is scalable even during takeover compared with the PRT schemes using passive replication techniques. HA_j restores the bindings of all the mobile nodes registering with HA_l and obtains the logged messages for HA_l from the stable storage. Then, it can recover the latest bindings of all the MNs, which HA_l has managed before it failed, by replaying the messages in receive sequence order. After that, HA_j performs a gratuitous address resolution protocol mapping HA_l's IP address to HA_j's hardware address to take over HA_l [Plummer (1982)]. Then, HA_j serves the MNs on behalf of HA_l. Therefore, the CML scheme provides the MNs with transparency of their MA's failure and replacement.

If a failed MA, named MA_i, is repaired in the CML scheme, it should monitor any agent advertisement message, including its IP address, for a few seconds and perform a gratuitous address resolution protocol mapping its IP address to its hardware address. If no other MA has taken over the role of the repaired agent, it should restore the bindings of all the MNs managed before it failed and obtain its logged messages from the stable storage. Then, it can recover the latest bindings of the MNs served in its pre-failure state by replaying the messages in receipt sequence order. If it receives an agent advertisement message including its IP address from a live MA, named MA_k, it should require from MA_k the bindings of the MNs served in its pre-failure state. If MA_k fails during its recovery, MA_i can recover the latest bindings of the MNs using its checkpoint and logged messages on the stable storage. Therefore, the CML scheme allows each failed MA to recover bindings of the MNs registering with the MA when it is repaired even if all the MAs in the same network fail.

The Description

In this part, we will first describe our fault-tolerant scheme for home agents and then for foreign agents.

The Description for Home Agents

(1) Data Structures

Every home agent in Mobile IP has the following data structures for our scheme.

- ***HATable$_i$***: It is a vector for saving the timer of each home agent clustered in a network. The timer of each home agent is used so that HA_i detects whether the agent is alive or failed currently. The timer of each home agent is initialized to *INIT_TIME*.
- ***RSN$_i$***: It is the receive sequence number of the latest registration request message which was delivered to HA_i. It is initialized to 0.
- ***Loc_Info$_i$***: It is a vector for saving the bindings of mobile nodes registering with HA_i.

- ***Log_Info***$_i$: It is a set for saving every permitted registration request message delivered to HA_i beyond the latest checkpoint and RSN_i of the message. Its element is of the form $e = (msg, rsn)$. It is initialized to $\varnothing$.

(2) Checkpointing and Message Logging Algorithm

```
procedure Register_MN(m, mn)
if(mn's home agent is HA_i) then {
  authenticate m ;
  if(m is permitted) then {
parallel
psections
    section
    RSN_i <- RSN_i + 1 ;
      save (m, RSN_i) into Log_Info_i at the stable storage ;
section
update mn's binding in Loc_Info_i using m ;
psections end
parallel end
     send a registration reply message for acceptance to mn ;
   }else
     send a registration reply message for rejection to mn ;
}
procedure Checkpointing()
save Loc_Info_i and RSN_i into the stable storage ;
remove all e ∈ Log_Info_i at the stable storage ;
```

Figure 4 Procedures for HA_i's logging each registration request message during registration process and periodically saving bindings of MNs registering with HA_i into stable storage

Figure 4 shows a formal description of our checkpointing and receiver-based pessimistic message logging algorithm. Whenever a home agent HA_i receives a registration request message m from a mobile node mn, it calls procedure **Register_MN**(), which first authenticates m. If m is valid, HA_i performs two executions in parallel: incrementing RSN_i by one and saving (m, RSN_i) into *Log_Info*$_i$ at the stable storage, and updating mn's binding in *Loc_Info*$_i$ using m. After having completed both executions, it sends mn a registration reply message for acceptance. Otherwise, HA_i sends a registration reply message for rejection to mn.

If HA_i attempts to take its local checkpoint, it calls procedure **Checkpointing**(). In this procedure, HA_i saves *Loc_Info*$_i$ and its current receive sequence number into the stable storage. Then, it removes all the messages logged in *Log_Info*$_i$ beyond the previous checkpoint.

(3) Failure Detection Algorithm

```
procedure Recv_AAM(m, j)
HATable_i[j] <- INIT_TIME ;
procedure Failure_Detect()
for all k ∈ other home agents in the same network of HA_i st (HATable_i[k] > 0)
  HATable_i[k] <- HATable_i[k] - 1 ;
for all k ∈ other home agents in the same network of HA_i st (HATable_i[k] = 0)
  if(Election_For_Takeover(k) = i) then take over HA_k ;
```

Figure 5 Procedures for HA_i's detecting failures of other home agents in the same network using agent advertisement messages

The home agent failure detection algorithm using agent advertisement messages is given in figure 5. If a home agent HA_i receives an agent advertisement message from another home agent HA_j, it calls procedure **Recv_AAM**() to set the timer for HA_j in *HATable*$_i$ to *INIT_TIME*.

Whenever *ADVERTISING_INTERVAL* (2 ~ 3) seconds have elapsed, HA_i calls procedure **Failure_Detect**(). In this procedure, it decrements the timer for every other home agent by one. If among the other home agents in the same network, there are ones for which timers expire, procedure **Election_For_Takeover()** is called so that the remaining home agents determine which live home agents take over the failed ones. In this procedure, among the remaining live agents, one, which serves the minimum number of mobile nodes, takes over a failed agent.

(4) Takeover and Recovery Algorithm

Figure 6 and 7 show a formal description of our takeover and recovery algorithm of a home agent, HA_i. If HA_i takes over a failed agent HA_j after having completed **Election_For_Takeover()**, it calls procedure **Take_Over()** in figure 6. In this procedure, HA_i restores *Loc_Info*$_j$, *Log_Info*$_j$ and RSN_j from the stable storage and then, it updates all the bindings in *Loc_Info*$_j$ to the latest by replaying each logged message in receive sequence order. Afterward, it performs a gratuitous address resolution protocol mapping HA_j's IP address to its physical hardware address. Then it should serve the MNs having registered with HA_j, send an agent advertisement message for HA_j to other home agents every *ADVERTISING_INTERVAL* seconds and respond to every address resolution protocol request message sent for HA_j.

When a failed home agent HA_i is repaired and rebooted, it calls procedure **Recover()** in figure 7. In this procedure, it invokes procedure **Listen_To()** to listen to any agent advertisement message including its IP address for sometime (i.e., *INIT_TIME* × *ADVERTISING_INTERVAL* seconds). Then, it performs a gratuitous address resolution protocol mapping its IP address to its physical hardware address. If it receives no agent advertisement message including its IP address, it restores *Loc_Info*$_i$, *Log_Info*$_i$ and RSN_i from the stable storage and then updates all the bindings in *Loc_Info*$_i$ using *Log_Info*$_i$. If it receives an agent advertisement message including its IP address from HA_k, it requires *Loc_Info*$_i$ and RSN_i from HA_k by remotely calling procedure **Req_Bindings()** at HA_k. If HA_k has failed before completing the procedure, HA_i restores *Loc_Info*$_i$, *Log_Info*$_i$ and RSN_i from the stable storage and then updates all the bindings in *Loc_Info*$_i$ using *Log_Info*$_i$.

The Description for Foreign Agents

In Mobile IP, failures of foreign agents are less critical than those of home agents. It means that if a mobile node has registered with a foreign agent in a network and the agent fails currently, the mobile node can re-register with another foreign agent in the same network [Perkins (1996b)]. However, to do so, it should send a registration request message to and receive a registration reply message from its home agent through the new foreign agent.

```
procedure Take_Over(j)
restore Loc_Info_j, Log_Info_j and RSN_j from the stable storage ;
for all e ∈ Log_Info_j in e.rsn order
update Loc_Info_j replaying e.msg ;
perform a gratuitous ARP binding HA_j's IP address to HA_i's physical hardware address ;
```

Figure 6 Takeover Procedures for a home agent HA_i

```
procedure Recover()
k ← Listen_To(i) ;
perform a gratuitous ARP binding $HA_i$'s IP address to $HA_i$'s physical hardware address ;
if($k \neq i$) then {
    ($Loc\_Info_i$, $RSN_i$) ← remote call at $HA_k$ : Req_Bindings(i) ;
    if($HA_k$ has failed before completing Req_Bindings(i)) then {
        restore $Loc\_Info_i$, $Log\_Info_i$ and $RSN_i$ from the stable storage ;
        for all $e \in Log\_Info_i$ in e.rsn order
            update $Loc\_Info_i$ replaying e.msg ;
    }
} else {
    restore $Loc\_Info_i$, $Log\_Info_i$ and $RSN_i$ from the stable storage ;
    for all $e \in Log\_Info_i$ in e.rsn order
        update $Loc\_Info_i$ replaying e.msg ;
}
procedure Req_Bindings(j)
    return ($Loc\_Info_j$, $RSN_j$) ;
```

Figure 7 Recovery procedure for a home agent HA_i

The re-registration process across the Internet may require high latency for recovering binding of the mobile node. Thus, this method forces other nodes not to communicate with the mobile node until the registration process is completed. To solve the problem of the traditional method, live foreign agents can recover bindings of all the mobile nodes having registered with failed agents faster than the traditional method by using the same scheme as the scheme for home agents described in the previous section. However, unlike a home agent, a foreign agent logs a registration request message, sent from each mobile node having registered with the agent, into the stable storage only after it has received a registration reply message for acceptance from the home agent of the mobile node. If the foreign agent fails, another live foreign agent in the same network restores bindings of the mobile nodes having served by the failed agent and logged messages. In this case, the live foreign agent need not forward each logged message to the corresponding home agent when replaying the message. If the failed foreign agent attempts to recover, but there is no live agent having taken over the failed agent, it replay each logged message in the same manner.

CORRECTNESS

In this section, we prove the correctness of our checkpointing and message logging algorithms, takeover algorithm and recovery algorithm for mobility agents.

Theorem 1 Our checkpointing and message logging algorithms ensure that the current state of p, s_p^k $(0 \leq k)$, is recoverable.

Proof: The proof proceeds by induction on k, the index of the state interval of each mobility agent p.

[Base case]

In this case, s_p^0 is the initial state interval of p and deterministic. Therefore, s_p^0 is trivially recoverable by lemma 2 because si_p^0 is stable.

[Induction hypothesis]

We assume that the theorem is true in case that $k = n$.

[Induction step]

If there is the determinant of $dev_p^{n+1}(m)$ on stable storage because the algorithms allows s_p^n to be recoverable by induction hypothesis, the theorem is true for p in case that $k=n+1$. In this step, when p receives the $(n+1)$-th registration request message m beyond s_p^n, it saves the determinant of $dev_p^{n+1}(m)$ on stable storage by calling procedure **Register_Mn()**. Afterwards, if p calls **Checkpointing()**, it saves s_p^{n+1}, i.e., the current bindings of the mobile nodes registering with it on stable storage, and then removes all the determinants for p from stable storage. Therefore, our algorithms allow s_p^{n+1} to be recoverable.

By the induction, our checkpointing and message logging algorithms ensure that the current state of p, s_p^k $(0 \leq k)$, is recoverable.

Theorem 2 If there are n $(2 \leq n)$ redundant mobility agents in a network and even n-1 among them fail concurrently, all the mobile nodes registering with the failed agents can communicate with other nodes in the system after a live mobility agent has completed our takeover algorithm.

Proof: The proof proceeds by induction on n, the number of redundant mobility agents in a network.

[Base case]

In this case, there are a failed and a live mobility agent, named MA_{fail} and MA_{live}, respectively. In our protocol, MA_{live} detects the failure of MA_{fail} when MA_{live}'s timer for MA_{fail} expires, and then take over MA_{fail} by executing procedure **Election_For_Takeover()**. Then, it restores from the stable storage the bindings of all the mobile nodes having registered with MA_{fail}, the logged messages beyond the latest checkpoint for MA_{fail}, and the receive sequence number of the latest logged message delivered to MA_{fail}. Then, it can recover the latest bindings of the mobile nodes by replaying each logged message in receive sequence number order. Afterward, it performs a gratuitous address resolution protocol mapping MA_{fail}'s IP address to its physical hardware address. Then it serves the MNs having registered with MA_{fail}, sends an agent advertisement message for MA_{fail} to other home agents every *ADVERTISING_INTERVAL* seconds and responds to every address resolution protocol request message sent for MA_{fail}. Thus, MA_{live} enables the mobile nodes to communicate with other nodes in the system.
[Induction hypothesis]
We assume that the theorem is true in case that $n = k$.
[Induction step]
For $n=k+1$, there are k failed mobility agents and a live one in a network. If the live mobility agent MA_{live} has only to take over the k-th failed mobility agent MA_{failk} because MA_{live} can take over k-1 failed mobility agents by induction hypothesis, the theorem is true in case that $n = k+1$.
In our takeover algorithm, MA_{live} can recover bindings of the mobile nodes having registered with MA_{failk}, and then serve the mobile nodes on behalf of MA_{failk} in the same manner like base case. Thus, MA_{live} enables all the mobile nodes having registered with k failed mobility agents to communicate with other nodes in the system.
By the induction, if there are n redundant mobility agents in a network and even n-1 among them fail concurrently, all the mobile nodes registering with the failed agents can communicate with other nodes in the system after a live mobility agent has completed our takeover algorithm.

Theorem 3 Our recovery algorithm allows every failed mobility agent to recover the latest bindings of all the mobile nodes served by it before it failed.
Proof: We prove this theorem by contradiction.
Assume that every failed mobility agent cannot recover the latest bindings of all the mobile nodes after having completed our recovery algorithm. When a failed mobility agent MA_{fail} is repaired and rebooted, it calls procedure **Recover()**. In this procedure, it listens to any agent advertisement message including its IP address for some seconds and then performs a gratuitous address resolution protocol mapping its IP address to its physical hardware address. There are two cases:
Case 1: There is no live mobility agent having taken over MA_{fail}.
In this case, the information is in the stable storage that is needed for recovering the latest bindings of the mobile nodes served by MA_{fail}. The information consists of bindings of the mobile nodes, logged messages for MA_{fail} and the receive sequence number of the latest message delivered to its IP address. In **Recover()**, MA_{fail} restores the information from the stable storage and then recovers the latest bindings of the mobile nodes by replaying each logged message in receive sequence order.
Case 2: A live mobility agent MA_{live} has taken over MA_{fail}. In this case, MA_{live} has the latest bindings of the mobile nodes served by MA_{fail}. In **Recover()**, MA_{fail} requires from MA_{live} the bindings of all the mobile nodes served by it and the receive sequence number of the latest message delivered to its IP address by remotely calling procedure **Req_Bindings()** at MA_{live}. There are two sub-cases:
Case 2.1: MA_{live} successfully completes **Req_Bindings()**.
In this case, MA_{fail} obtains the latest bindings of the mobile nodes from MA_{live}.
Case 2.2: MA_{live} has failed before it completes **Req_Bindings()**.
In this case, MA_{fail} recovers the latest bindings of the mobile nodes in the same manner like case 1.
Hence, both case 1 and 2 contradict the hypothesis.

Table 1. Parameters and their meanings

Parameter	Meaning
NOMN	Number of mobile nodes registering with a network
NOMA	Number of redundant mobility agents in a network

COMPARISONS

In this section, we intend to compare the CML scheme with the PRT scheme presented in [Ghosh (1998)] briefly. Generally, performance indices used for evaluation of scalability of the two schemes are the number of MNs whose bindings are managed by each MA in a network (denoted by *NOMNMA*) and the latency time for its processing a registration request message from each MN (denoted by *LTRR*). For simplicity, we suppose that each MA in a network serves the same number of MNs. It means that if the number of MNs registering with a network is *n* and the number of MAs in the network is *m*, the number of MNs served by each agent is (*n* / *m*).

First, we evaluate scalability of the two schemes with respect to *NOMNMA* during failure-free operation. Table 1 shows the parameters, which *NOMNMA* depends on, and their meanings. If *NOMNMA*s of the two schemes are calculated using the parameters respectively, *NOMNMA* of the CML scheme is *NOMN* / *NOMA* whereas that of the PRT scheme is *NOMN* during failure-free operation. Figure 8 illustrates *NOMNMA* versus *NOMN* when *NOMA* is 10. In this figure, *NOMNMA* increases as *NOMN* increases in the two schemes. However, we can see that the increasing rate of *NOMNMA* in the CML scheme is significantly lower than in the PRT scheme. Figure 9 illustrates *NOMNMA* versus *NOMA* when *NOMN* is 6000. As *NOMA* increases in this figure, *NOMNMA* of the CML scheme decreases whereas *NOMNMA* of the PRT scheme is always equal to *NOMN*. The reason for these results is that each mobility agent in a network must maintain bindings of the mobile nodes registering with all the redundant mobility agents in the same network in the PRT scheme whereas it has only to maintain bindings of the mobile nodes registering with only it in the CML scheme. Therefore, we can see that the CML improves scalability to a large number of mobile nodes managed by each mobility agent compared with the PRT scheme during failure-free operation.

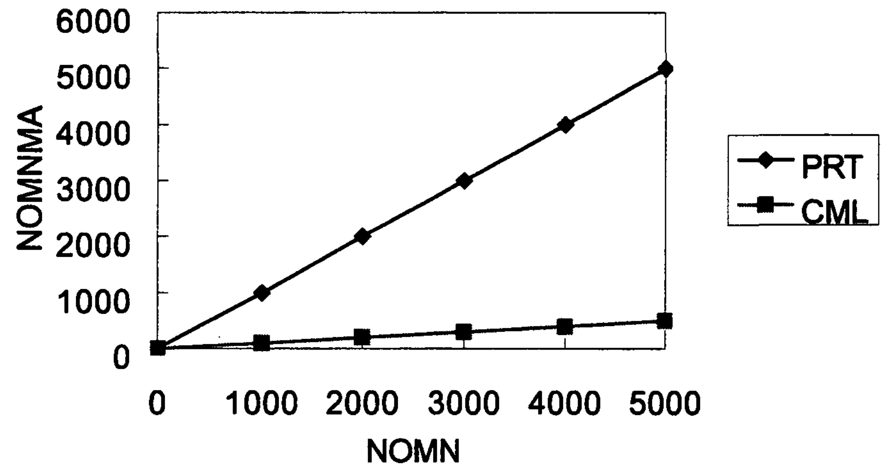

Figure 8 ***NOMNMA*** **versus** ***NOMN*** **(*NOMA*=10)**

Next, we evaluate scalability of the two schemes with respect to *LTRR* during failure-free operation. If each mobility agent receives a registration request message from a mobile node in the PRT scheme, it should process the message and forward the message to its peers, and wait for receiving all the acknowledgement messages from them.

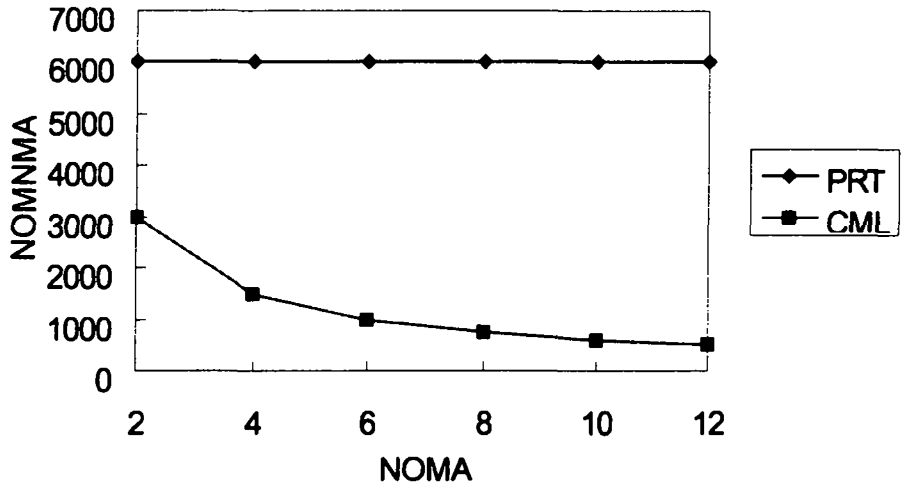

Figure 9 ***NOMNMA*** **versus** ***NOMA*** **(*NOMN*=6000)**

Thus, the total number of messages generated per registration request message in the PRT scheme is (2 × (*NOMA*-1)) and the number of messages on the critical path is *NOMA*. However, in the CML scheme, it should process the message and require the stable storage server to save the recovery information of the message into stable storage, and wait for receiving an acknowledgement message from it. Thus, the total number of messages generated per registration request message in the CML scheme is 2 and the number of messages on the critical path is 2. Therefore, we can see that the total number of messages and the number of messages on the critical path generated per registration request message in the CML scheme, not in the PRT scheme, are always constant.

Next, we evaluate overheads of the two schemes for taking over or recovering failed mobility agents. In the PRT scheme, live mobility agents can take over failed agents fast because they always maintain bindings of all the mobile nodes registering with not only itself but also its peers. Therefore, the takeover time of the CML scheme may be longer than that of the PRT scheme because each mobility agent has only to maintain the bindings of the mobile nodes registering with it and live mobility agents should recover the bindings of failed ones from the stable storage. However, the takeover time of the CML scheme can be reduced using two methods. The first method is that each mobility agent maintains only the latest in the stable storage among registration request messages sent from each mobile node registering with the agent. This method decreases the number of logged messages that each live mobility agent should replay when it takes over a failed agent. The second method is implementing the stable storage as a high-speed and scalable storage system such as Storage Area Network (SAN) [IBM (1999)].

If there are live mobility agents in a network, the recovery time of failed and repaired agents are the same in the two schemes because they can recover their bindings from the live mobility agents in the schemes. However, otherwise, each failed agent can recover its latest bindings from the stable storage in the CML scheme whereas it cannot recover them in the PRT scheme.

CONCLUSION

In this paper, we identified the problems in the PRT schemes using passive replication techniques; high failure-free latency during registration process if the number of MAs in the same network increases and forcing each MA to manage bindings of all the MNs registering with its network. Then, we presented the CML scheme using checkpointing and receiver-based pessimistic message logging techniques. The CML scheme achieves low failure-free latency even if the number of MAs in a network increases, and improves scalability to a large number of MNs registering with each network compared with the PRT schemes. Additionally, the CML scheme allows each failed MA to recover bindings of the MNs registering with the MA when it is repaired even if all the MAs in the same network fail. However, the takeover time of the CML scheme may be longer than that of the PRT scheme like in section 4. The takeover time of the CML scheme can be reduced using the two methods mentioned in section 4.

REFERENCES

Alvisi, L., Hoppe, B. & Marzullo, K. (1993) "Nonblocking and Orphan-Free Message Logging Protocols", **Proc. the 23rd Symposium on Fault-Tolerant Computing**, pp. 145-154.

Binkley, J. R. & McHugh, J. (1997) **Secure Mobile Networking: Sixth Quarterly Report – Winter 1997,** Portland State University.

Dixit, A & Gupta, V. (1996) **Mobile-IP for Linux (ver 1.00),** Technical Report, SUNY Binghampton.

Droms, R. (1993) **Dynamic Host Configuration Protocol,** RFC 1541.

Elnozahy, E. N. & Zwaenepoel, W. (1992) "Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output commit", **IEEE Transactions on Computers,** Vol 41, pp. 526-531.

Elnozahy, E. N., Alvisi, L., Wang, Y. M. & Johnson, D. B. (1999) **A Survey of Rollback-Recovery Protocols in Message-Passing Systems,** Technical Report CMU-CS-99-148, Carnegie-Mellon University.

Ghosh, R. & Varghese, G. (1998) **Fault-Tolerant Mobile IP**, Technical Report WUCS-98 -11, Washington University.

IBM Corporation and International Data Group. (1999) **Survey says Storage Area Networks may unclog future roadblocks to e-Business,** News Release.

Johnson, D. B. (1994) "Scalable and Robust Internetwork Routing for Mobile Hosts", **Proc. the 14th International Conference on Distributed Computing Systems,** pp. 2-11.

Perkins, C. (1996) **IP Encapsulation Within IP,** RFC 2003.

Perkins, C. (1996) **IP Mobility Support,** RFC 2002.

Perkins, C. (1996) **Minimal Encapsulation With IP,** RFC 2004.

Perkins, C. & Johnson, D. B. (1997) **Route Optimization in Mobile-IP,** Mobile IP Working Group, Internet Draft.

Plummer, D. C. (1982) **An Ethernet Address Resolution Protocol - or – Converting Network Protocol Address to 48 bit Ethernet Address for Transmission on Ethernet Hardware. STD 37,** RFC 826.

Postel, J. B. (1981) **Internet Protocol,** Internet Request For Comments RFC 791.

Rose, M. T. (1990) **The Open Book: A Practical Perspective on OSI,** Prentice Hall, Englewood Cliffs, NJ.

Schlichting, R. D. & Schneider, F. B. (1985) "Fail-stop processors: an approach to designing fault-tolerant distributed computing systems", **ACM Transactions on Computer Systems,** Vol.1, No.3, pp. 222-238.

Simpson, W. (1995) **IP in IP Tunneling,** RFC 1853.

Solomon, J. D. (1998) **Mobile IP, The Internet Unplugged,** Prentice Hall Series.

Strom, R. B. & Yemeni, S. (1985) "Optimistic recovery in distributed systems", **ACM Transactions on Computer Systems,** Vol.3, No.3, pp. 204-226.

Turner, P. (1990) **NetWare communications processes,** NetWare Application Notes, Novell Research, pp. 25-81.