The Journal of Supercomputing, 31, 79–99, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

Mobile Agent Connection Establishment and Management (CEMA)—Message Exchange for Pervasive Computing Environments

FARHAD KAMANGAR kamangar@cse.uta.edu
DAVID LEVINE levine@cse.uta.edu
GERGELY V. ZÁRUBA zaruba@cse.uta.edu
RENJITH THOMAS rthomas@cse.uta.edu
Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, TX-76019, USA

Abstract. Pervasive computing is an emerging technology that offers new possibilities to distributed computing and computer networking; it employs a wide variety of smart, ubiquitous devices throughout an individual's working and living environment. Mobile agents are software entities that can migrate between servers (mobile agent environments) of the network accomplishing various tasks on the behalf of their owners. The objective of this paper is to describe a test and prototyping environment for experimenting with mobile agents in pervasive environments. A prototype environment for a novel, proactive infrastructure is described for mobile agent assisted pervasive computing. In addition, a new message passing algorithm is provided for mobile agent connection establishment and management (CEMA). Simulation results show the performance of the proposed approach.

Keywords: mobile agents, pervasive computing, distributed computing

1. Introduction

This paper introduces a new mobile agent communication paradigm to be used in pervasive computing environments [1, 7, 10, 23, 27]. Mobile agents are software entities or programs that can migrate from host to host, at the instance and destination of their own choosing. A host refers to any node with computational capabilities providing an execution environment for mobile agents. Hosts can vary in size and computational power from tiny sensory devices through small portable computers to large servers. Typical examples include workstations, desktops, laptops, PDAs, and more sophisticated/dedicated devices such as routers, database servers, or access points. There are several wellknown mobile agent environments in existence today, including IBM Aglets [14] (Java bytecode based), Tcl agents [9, 11], and Telescripts [6] (both script based); all of these methods are interpreted by respective agent servers enabling code mobility.

Pervasive computing is a newly emerging paradigm to provide users with anytime anywhere access to information or computing resources. Pervasive computing enables convenient access to relevant information for users and applications through a class of intelligent and ubiquitous software and hardware entities that have the ability to come alive and become available when and where needed. Ubiquitous entities refer to small, mobile computing devices such as handheld phones, portable and wearable computers and appliances equipped with intuitive user interfaces to enhance information processing while being accessible in different types of networks. Many research groups and projects throughout the world are focusing on various aspects of pervasive computing, e.g., the Oxygen project at MIT [18].

The motivation behind this paper is to extend the mobile agent paradigm to pervasive computing environments, enabling the development of novel ubiquitous applications. One of the features of mobile agents is the asynchronous and autonomous behavior of mobile agents. Mobile agents are generally transport layer and architecture independent depending only on the availability of the execution environment. Mobile agents can be designed to be robust and fault tolerant to dynamically adapt to unfavorable conditions.

The prototype designs of the mobile agent architecture as well as the server for the mobile agent architecture are described in Section 2. Section 3 outlines a novel Connection Establishment and Management Algorithm CEMA for message passing among mobile agents in heterogeneous networks. Section 4 presents a simulation based performance analysis of CEMA. A very preliminary version of this research has been previously published in the PDPTA'03 Conference [15]

2. Background and previous work

There are several well known mobile agent and pervasive computing environments in existence, including IBM's Aglets [14], Tcl Agents (now known as D'Agents) [9, 11], Oxygen [18], DECAF [8], Java-To-Go [16], and Pico [12]. In addition agent communication languages such as MASIF (Mobile Agent Systems Interoperability Facility) [13, 20] and KQML (Knowledge Query Manipulation Language) [2] are used to enable agent communication and verification [26]. Experimental mobile agent infrastructures have been implemented and deployed in distributed systems [3]. Mobile agent systems provide the software environment for mobile agent programs to start running on a host computer and, at will, suspend their execution state and move to another host, where they may resume execution. Unlike aglets, which are small applications sent to a client and run once from start to finish, mobile agents continue execution where they left off and may move an arbitrary number of times to any number of hosts. Where code-migration systems attempt to distribute work load by moving running programs, mobile agents move according to their own needs and routes.

Stationary programs are compiled and executed or interpreted on a host computer while mobile agents must be transported from one host to another and run on hosts with possibly different processors and operating systems. These constraints usually encourage the building of mobile agent systems utilizing scripting languages which are directly interpreted (e.g., Tcl) or programs compiled to an interpreted intermediate form (e.g., Java bytecodes). Not only do these languages allow portable execution on heterogeneous hosts, the code transported is typically smaller than binary executables. Other benefits of interpreted forms include providing for some security considerations for running on a host (for example, Java sandbox or trusted/signed code) and the ease of freezing execution before transporting by capturing the runtime stack state not actual processor memory (which may be virtual, mapped, and contain pointers). The additional cost of interpretation may be offset by just-in-time compilers and smaller network data payloads.

Another critical issue in mobile agents is the ability to communicate with other agents (stationary or mobile, of the same type or different). This issue has resulted in OMG (Object Management Group) building a CORBA based standard MASIF which deals with managing (creating, suspending, resuming), transporting (between agent systems of different types) and naming mobile agents. While MASIF promises easier communication between agents it does not attempt to handle a mobile agent wanting to communicate with a non-agent source of information or handling the meaning of the data at a source (where the names of the data may be different but the data contents are the same), or where the host may have higher-level ways of managing data that the mobile agent was not originally programmed to handle (for example: advanced queries versus a simple string search).

Agent communication languages (ACL) such as KQML (a Lisp-like s-expression based language) are a multi-layered (content, communication and message are separated) way to describe information meta-data (e.g., by defining what is required and what are the required capabilities). KQML describes facilitators as agents that “match-make” connections between information requestors and providers and may optionally translate and mediate between clients and providers.

Researchers at Georgia Tech have been innovative in using mobile agents in pervasive computing environments, one approach has been linking agents with mobility [24], while work emphasizing mobile agent security has been described in [21]. Other groups have focused on using agents to provide collaborative as well as ubiquitous computing as in Berkeley’s Lawrence Berkeley’s National Laboratory Pervasive Collaborative Computing Environment Project [4].

Finally, pervasive computing environments employ agents to create an “anytime, anywhere” smart environment where specialized hardware and software interoperate and communicate to provide a community of agents. MIT’s Oxygen project is modeled after the Oxygen in the air (available for breathing but not visually noticeable). Special hardware, embedded devices (cameras and microphones, for example) will work with handheld devices (such as a personal digital assistant- PDA) utilizing a self-configuring network to automatically locate devices, people or services in an ever present environment that never shuts down or needs to be re-booted. Users are expected to communicate in natural language speech and through gesturing to cameras to devices or other people, special software and the network will provide security, support changes, and provide interoperability. Analogous to Oxygen, the PICO project at UTA is striving to develop a ubiquitous environment middleware providing a mission-oriented dynamic community performing tasks for people and devices. Standard or specialized hardware devices will have mobile (or stationary) agents placed on them that allow them to create very fast forming dynamic communities to respond to special needs, for example cameras that detect a car accident will find devices to call an ambulance and hospital, when needed.

Our proposed infrastructure works with mobile agent systems and agent communication languages to allow mobile agents in a pervasive environment to find other mobile or stationary agents and exchange messages (of any type) with them reliably and efficiently.

3. Prototype architecture

One of the main research thrusts in the field of pervasive computing is making the computations and functional behavior of the system invisible to users while providing smart spaces around ubiquitous devices [27]. The proposed software infrastructure relies on the operating system to provide the execution environment for mobile agents. The requirements for the proposed software infrastructure in a host are as follows:

- Allow multiple mobile agents to co-exist and execute simultaneously without interference.
- Provide means of communication between mobile agents and the host, and between mobile agents.
- Provide a transport mechanism to transfer and receive agents to or from other hosts.
- Provide mechanisms to save the state of an executing mobile agent.
- Be capable of receiving a mobile agent and resuming execution from the point it was suspended.

The agent environment consists of five software blocks as outlined in Figure 1.

The Data Router is the nerve center of the infrastructure. It is responsible for receiving, interpreting and exchanging data between the mobile agents and the outside world. The architecture of the data router is as shown in Figure 2.

All incoming data flow into the data router through the XML parser. At the XML parser, the XML tags are parsed and send to the interpreter. The interpreter consults with the directory services to determine whether the data is addressed to the current host and interprets the type of data. If the data is not addressed to the host, it is sent to the XML generator where the appropriate modifications are made to the data and then transmitted to the most appropriate host. The primary function of the data distributor is to deliver data to mobile agents within a host. It is also responsible for transferring mobile agents to the MA loader. The XML generator is responsible for generating XML tags for the data that is to be sent outside

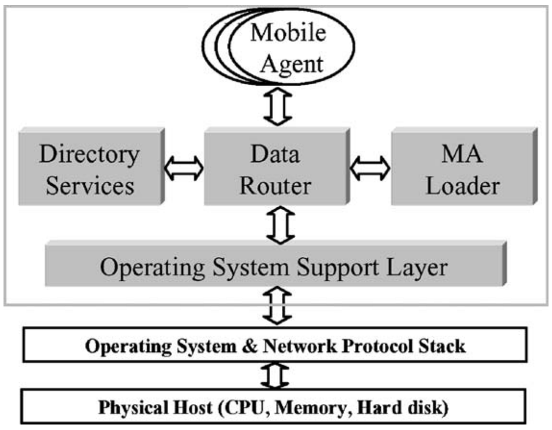

Mobile Agent
Directory Services
Data Router
MA Loader
Operating System Support Layer
Operating System & Network Protocol Stack
Physical Host (CPU, Memory, Hard disk)

Figure 1. Prototype architecture.

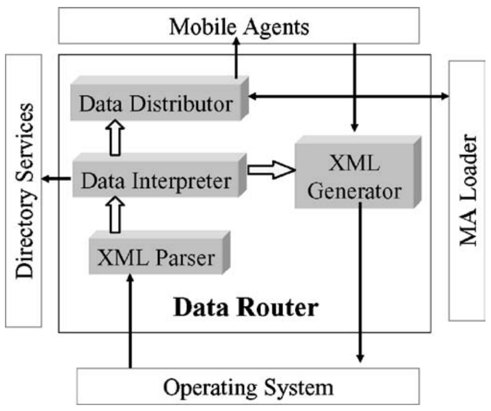


Figure 2. Architecture of data router.

the host. The routing functions are supported by the directory services. Directory Services provide lookup tables for inter-host as well as intra-host data transfer. Directory services implement three directory-tables for communication services: resource table, service table and lookup table:

The resource table maintains information on neighboring hosts. The service table stores information about mobile agents residing in the neighboring hosts as well as the native host. The lookup table implements a volatile cache of mobile agent addresses enabling the routing of messages among mobile agents.

The Mobile Agent Loader (MA Loader) provides the execution environment for the mobile agents. It maintains a process table with entries for each of the currently running mobile agents in the host, while tracking of the execution state of these agents. The MA loader is capable of (i) receiving mobile agents as a serialized binary objects; (ii) de-serializing the binary objects; (iii) allocating a process space and launching the mobile agents. It is also able to retrieve the previous context of the mobile agent and resume its normal execution. When a mobile agent wants to migrate from one host to another, the MA loader is capable of serializing the mobile agent and sending it to the XML generator, where the necessary XML tags are generated.

The Operating System Support Layer provides low-level services to the other components in the infrastructure; it maintains information about the host system and provides communication channels and basic input and output primitives. Figure 3 shows the details of the OS support layer. It is assumed that the OS support layer is also hardware dependant. Information on hardware components of the host is sent from the OS support layer. Mobile agents are capable of communicating with the operating system by means of OS Support Layer.

3.1. Mobile agent architecture

A mobile agent is a thread of execution so it has all characteristics of a thread with some enhancements. Mobile agents inherit all properties of threads; they are able to spawn parallel

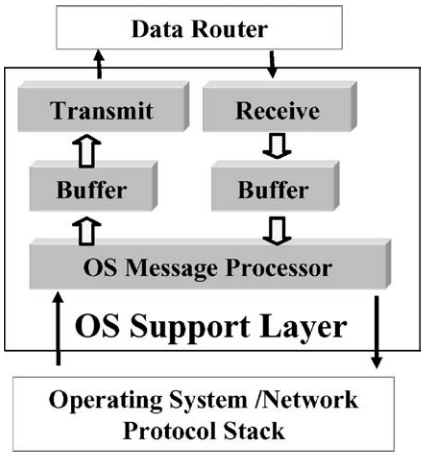


Figure 3. OS support layer.

threads on their own, sleep, suspend, stop and resume like any normal thread. In order to enable migration, mobile agents have a serializable interface that allows the process to be suspended and transported. Mobile agents should also have a uniform MA interface with a standard framework for mobile agent communication.

A UML representation of a mobile agent is given in Figure 4. All other mobile agents are inherited from this base mobile agent and may have additional functionality to meet special needs. The basic infrastructure remains the same in all mobile agents. The basic architecture of a mobile agent is depicted in Figure 5. Mobile agents consist of three modules: functional module, control module and communication module:

- The *functional module* contains the functions the mobile agent should perform; the functions can receive input from the host, or from other agents.

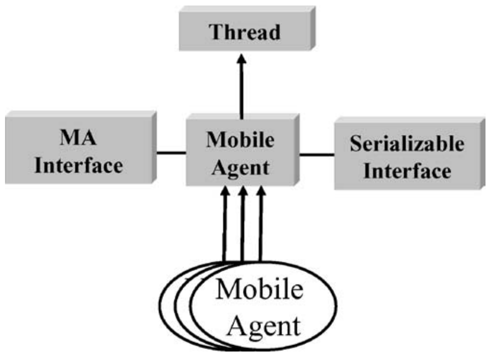


Figure 4. UML representation of mobile agents.

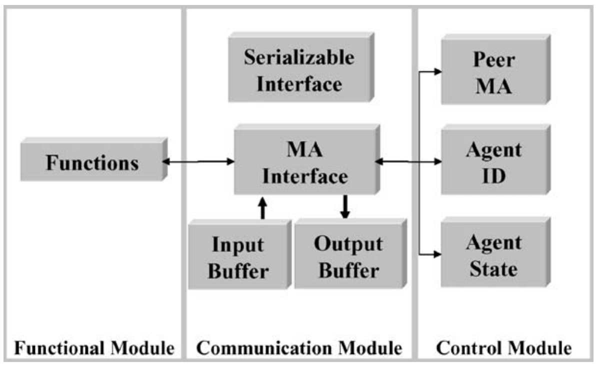


Figure 5. Basic mobile agent architecture.

- The *control module* consists of three submodules maintaining information about the mobile agent. The ID sub-module is responsible to carry the MA identity. The Peer sub-module maintains a list of peer mobile agents. The State sub-module stores the state information of the MA.
- The communication module maintains a common framework for communication with the software architecture. The MA Interface submodule allows simultaneous reception and transmission of messages between agents. The input buffer provides buffering for incoming data, while the output buffer does the same for outgoing data. The serializable interface serves the agent in preparing for migration to another host by suspending all parallelism.

4. CEMA

CEMA (Connection Establishment and Management Algorithm) is used to establish and manage connections between mobile agents. CEMA ensures that all mobile agents are able to communicate with peer agents even in sparsely connected pervasive networks. Previous research addressing communication among mobile agents has concentrated on fully connected networks, in which any two hosts can communicate with each other directly [23]. While an active message approach [24] uses an agent to route messages on-the-fly, CEMA calculates routes on demand.

4.1. *Design components*

The network between hosts is represented by an undirected graph $G = (V, E)$ where V is the set of vertices representing the hosts, and E is the set of edges representing a communication link between two hosts. E_{xy} denotes a connection between vertices V_x and V_y implying a full duplex communication channel. C_x represents the set of vertices connected directly to V_x. Communication between two hosts not directly connected with each other requires

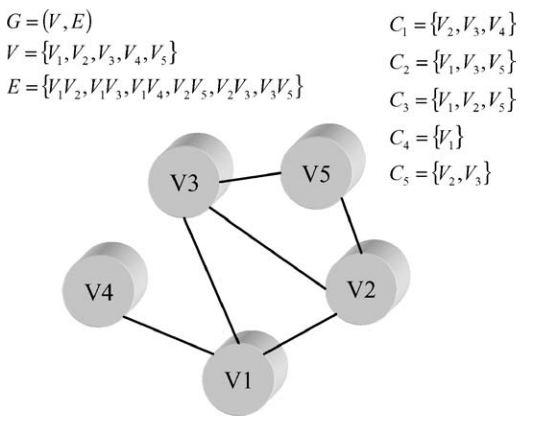


Figure 6. A sample network.

messages relayed by other intermediate nodes. A sample multihop network is depicted in Figure 6.

Key parameters of messages relayed from one host to another are: *mode*, *sid* (sender mobile agent id—the unique identifier of the originating MA), *shid* (sender host id—the unique identifier of the originating host), *rid* (receiver agent id—the unique identifier of the destination MA), *ts* (timestamp), *info* (the body of the message), *current* (the unique identifier of the host currently running CEMA on the message). Depending on the *mode* parameter, messages are divided into three categories:

- **Control messages** (*mode* = 0): contain routing information and are dispatched when an agent is launched in the agent loader. When an agent migrates, it will transmit control messages to its peer mobile agents.
- **Normal messages** (*mode* = 1): contain user data to be relayed between mobile agents.
- **Acknowledgement messages** (*mode* = 2): contain acknowledgments for messages.

As described in Section 3, each host maintains three tables in the directory services that help in sending messages:

- **Resource table:** is a hash table maintaining information on the physical neighbors of a specific host with unique identifier *hid (host id)*. The *hid*s are used as keys associated with corresponding values (*values*) (communication parameters).
- **Service table:** is a hash table maintaining information about mobile agents residing in the physical neighbors of hosts. A typical key in a service table is the unique identifier of mobile agents residing at the neighbors while the corresponding values are the unique identifiers of the serving hosts.
- **Lookup table:** is a hash table maintaining routing information on mobile agents in a host with unique identifier *hid*.

4.2. CEMA primitives

The following list contains the CEMA primitives:

CEMA-RESOURCES (*res* (*hid*)) is called to retrieve the unique identifiers of all the physical neighbors of a host *hid*. It has a complexity of $\boldsymbol{O}(N)$ where N is the number of hosts in the network.

CEMA-CONTAINS-MOBILE-AGENT-SERVICE (*ser*(*hid*), *id*) determines whether there is an entry key for an agent in service table of the host with unique identifier *hid*. It has a complexity of $\boldsymbol{O}(N)$.

CEMA-FIND-ADDRESS-SERVICE (*ser* (*hid*), *keys*) returns the unique identifier of the host where the mobile agent exists. It has a complexity of $\boldsymbol{O}(1)$.

CEMA-CONTAINS-MOBILE-AGENT-LOOKUP (*lookup*(*hid*), *keys*) determines if there is an entry for an agent in the lookup table of host with unique identifier *hid*. It has a complexity of $\boldsymbol{O}(M)$ where M is the number of mobile agents in the network.

CEMA-FIND-ADDRESS-LOOKUP(*ser* (*hid*), *keys*) returns the unique identifier of the host where the mobile agent is presumed to exist. It has a complexity of $\boldsymbol{O}(1)$.

CEMA-VERIFY-MESSAGE-LOOKUP (*lookup*(*hid*), *keys*, *newts*) verifies if a message it received is an old message; if so then the message should be discarded. It has a complexity of $\boldsymbol{O}(M)$.

CEMA-UPDATE-LOOKUP (*lookup*(*hid*), *keys*, *newts*, *level*) is used to add/update entries in the lookup table *hid* when it receives a message from an. It has a complexity of $\boldsymbol{O}(M)$.

4.3. Description of CEMA

A high level pseudo code description of the CEMA algorithm is given in Figure 7. CEMA runs in the data router module of each host. It consults with the directory services to determine what function should be performed on each message it receives. The *current* parameter refers to a unique identifier of the host where the algorithm is currently executing. The output of CEMA is the most optimal neighbor to route the message to. The variable *level* refers to the host from where the message was received.

Lines 1–4 initialize the variables that are being used in CEMA. The Boolean variable *found* is set to true when the location of the destination agent *rid* is found in the directory services during the execution of the algorithm. Control messages ($mode = 0$) do not rely on the lookup table (*lookup*(*current*)) to route from the source mobile agent to the destination agent. Control messages are used to build the lookup table, so they consult only the service table (*ser*(*current*)) and the resource table (*res*(*current*)). Line 5 ensures that the lookup table is queried only if the message is not a control message.

In line 8, CEMA checks whether the message is an old message; if the message is old then it is discarded. When a message is received, only the information regarding the sender mobile agent (*sid*) is updated. It has to be ensured that no entry for *sid* is added in the lookup of the host where *sid* is residing (implemented by lines 11–14). An update is

```
CEMA(current, lookup(current), service(current),resource(current), sid ,rid, mode, ts, level)

1  destination ← NULL
2  found ← FALSE
3  continue ← FALSE
4  temp ← NULL
5  if CEMA-CONTAINS-MOBILE AGENT-LOOKUP(lookup(current),rid)) = TRUE and mode ≠ 0
6    then found = TRUE
7         destination ←CEMA-FIND-ADDRESS-LOOKUP(lookup(current), rid)
8  if CEMA-VERIFY-MESSAGE-LOOKUP(lookup(current),sid))
9    then continue = TRUE
10 if continue = TRUE and level ≠ current
11   then if CEMA-CONTAINS-MOBILE AGENT-SERVICE(ser(current), sid) = TRUE
12          then temp←CEMA-FIND-ADDRESS-SERVICE(ser(current), sid)
13        if temp ≠ current
14          then CEMA-UPDATE-LOOKUP(lookup(current), sid ,ts, level)
15 if continue = TRUE and found = FALSE
16   then if CEMA-CONTAINS-MOBILE AGENT-SERVICE(ser(current), rid) = TRUE
17          then found = TRUE
18               destination←CEMA-FIND-ADDRESS-SERVICE(ser(current), rid)
19 if continue = TRUE and found = TRUE
20   then if current = destination
21          then if mode = 1 or mode = 2 or mode = 0
22                 then print "Destination reached"
23               if mode = 1
24                 then send an acknowledgement for the received message
25        else
26          send the message towards destination
27 if continue = TRUE and found = FALSE
28   then for each element in keys in CEMA-RESOURCES(res(current))
29          do if element ≠ level
30 then send the message towards element.
```

Figure 7. CEMA algorithm.

made for the location of *sid* along with the timestamp (*ts*) of the received message in line 15.

When mobile agents are dispatched to a pervasive network the first task they perform is to transmit control messages to their peer mobile agents. These control messages populate the lookup tables initially. The resource tables are populated when the network is formed. The service table is populated when the host receives the mobile agent. Figure 8 shows how the data in various tables are affected by the various events.

If a message is a control message ($mode = 0$), CEMA first checks if the message is outdated. It then updates the information in the lookup table for the originating agent. The service table is queried to determine the location of the destination agent. If the query is unsuccessful then CEMA will multicast the message to all of its neighbors, except for the neighbor from which it has received the message.

If a message is a normal message ($mode = 1$), then the lookup table is queried for the location of the destination host. CEMA then proceeds the same way as in the case of control messages. A fallback to search the service table is made only if CEMA fails to find the location of the destination host in the lookup table. Acknowledgement messages are generated for every normal message. Acknowledgement messages ($mode = 2$) are similar to normal messages except that they do not require an acknowledgement.

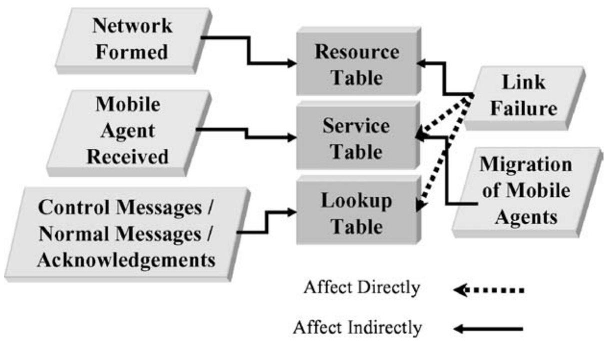


Figure 8. Events affecting tables in a host.

4.4. Link failure and migration of agents

When a mobile agent wishes to move from one node to another (from node x to y) it must remove itself from node x before going to node y. Formally, if MA_z wants to migrate from V_x to V_y then it must remove itself from $ser(V_x)$ before moving to V_y to ensure proper working of the routing algorithm. Upon reaching its destination, MA_z adds/updates an entry for itself in $ser(V_y)$, the service table for node y. It also removes any old entries in *lookup* (V_y) referring to MA_z. In the next step, MA_z transmits control messages to all of its peer mobile agents. Failure of edge E_{xy} mandates V_x and V_y to remove entries corresponding to V_x and V_y, from their resource, service, and lookup tables. If a new link is not registered by CEMA, the path remains undiscovered until a mobile agent migrates and sends out the control messages using (and detecting) the new link. As an example, suppose there are two mobile agents under consideration namely, MA_x and MA_y as shown in Figure 9(a). The

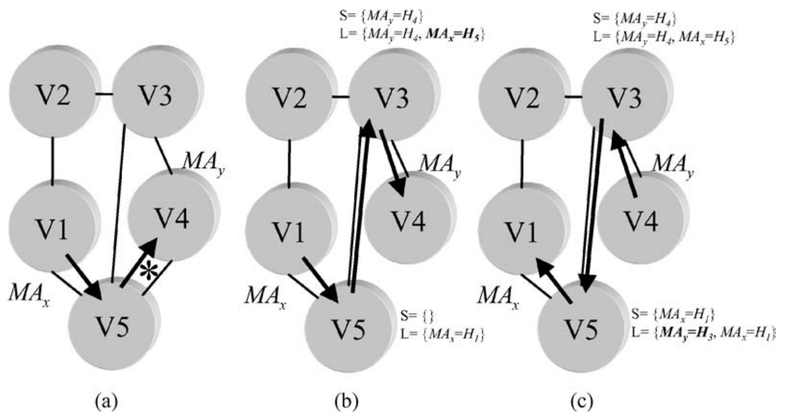


Figure 9. Illustration of link failure: (a) optimal path (b) new path (c) changes in the tables.

optimal path for communication between MA_x and MA_y is $V_1 - V_5 - V_4$. If the link between V_5 and V_4 fails, V_5 will remove all references to V_4 from its resource, service, and lookup tables. Suppose a message from MA_x addressed to MA_y reaches V_5 (Figure 9(b), it finds there is no references to MA_y since all entries leading to V_4 have been removed. V_5 will be left with no option other than to multicast the message to its physical neighbor V_3. Once the message is received by V_3, since MA_y is there in the service table of V_3 it will directly send the message to V_4 and a new entry will be made in the lookup table of V_3 for MA_x. This is shown in Figure 9(b). MA_y sends an acknowledgement message back to MA_x, which then makes a new entry in V_5 for MA_y as shown in Figure 9(c). Therefore, a new path is created from MA_y to MA_x and vice versa. Next time the message will be able to reach the destination without any multicasting.

5. Analysis of CEMA

This section presents a performance analysis of CEMA. The network is represented by an undirected graph $G = (V, E)$ where V is the set of vertices that represent hosts while E is the set of edges representing communication links between two hosts. E_{xy} depicts a full duplex connection between vertices V_x and V_y. The following assumptions are made:

- The communication channel remains duplex at all times.
- All link bandwidths and power sources of devices are infinite.
- The cost of all links are equal.
- There is equal delay on all links and there is no loss during transmissions.
- Graph G remains connected at all times.
- V_x and V_y are notified if E_{xy} is removed.

In a time interval τ, if n agents are launched then the total number of control messages generated is equal to $n * (n - 1)$. If, within the time internal τ, there are m migrations of mobile agents then the total number of control messages generated due to the migrations is: $m * (n - 1)$. If x number of messages are transmitted in a time interval τ, then the total number of normal messages is x and the total number of acknowledgement messages is x. Therefore in a time interval τ, if n mobile agents are launched followed by m number of migration and they send x messages among themselves, the total number of messages is: $n * (n - 1) + m * (n - 1) + 2x$.

The algorithm has a message complexity of $\boldsymbol{O}(N + M)$ where N is the number of hosts in the network and M is the number of mobile agents in the network. Messages in CEMA traverse the network in a breadth-first fashion; T is a breadth-first generated spanning-tree of nodes from V_x to V_y. The maximum number of hops a message (normal message) can take to reach its destination depends on the number of levels in T (the location of the mobile agent).

The algorithm allows only forward movement of messages and does not allow a message to revisit an edge it has already visited. Thus, the numbers of levels in T denoted by x determine the average number of hops a message takes to reach from a source to a destination. The worst-case scenario occurs when the graph G is a linear graph having a

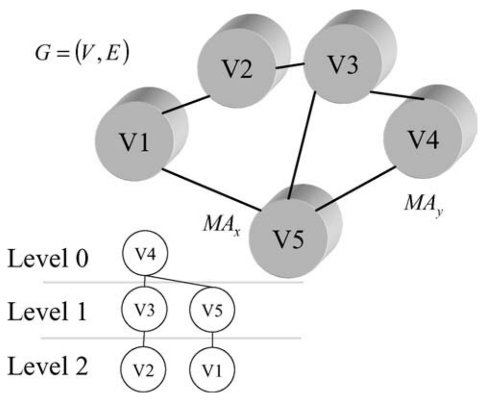


Figure 10. CEMA traversing the network.

worst-case complexity of $\boldsymbol{O}(|V|)$. In such a case, the number of *levels* in *T* is equal to the number of edges. The best-case scenario is when all the mobile agents are concentrated on a single host, while in a fully connected *G*, to reach the destination the message has to take either zero or one hop.

The CEMA algorithm is analyzed to obtain the number of hops it takes to reach a destination host. CEMA traverses the network in a breadth first traversal fashion as shown in Figure 10. Let us denote the sub graph created by the traversal by *T*. The traversal goes one *level* at a time, left to right within a *level* (where a *level* is defined simply in terms of hops from the root of the tree).

Lemma 1 *When CEMA is executed, it runs in a breadth first traversal fashion.*

Proof: The proof that vertices are in breadth first traversal fashion is made by induction based on the level number. By the induction hypothesis, CEMA first reaches all one-hop neighbors followed by the two hop neighbors. It reaches all vertices at level $k-1$ before those at level k. Therefore, it will reach all vertices at level k before all those of level $k+1$. □

Lemma 2 *The sub graph T generated by CEMA is a tree.*

Proof: Let us consider each edge *vw* in the breadth first traversal; *vw* can be seen as if it was pointing "upward" from *v* to *w*. Then, each edge points from a vertex visited later to one visited earlier. Following successive edges upwards can only be rooted by *x* (which has no edge going upward from it) so every vertex in *T* has a path to *x*. This means that *T* is at least a connected sub graph of *G*. A tree is just a connected and acyclic graph, so we need only to show that *T* has no cycles. In any cycle, no matter how edges are oriented there is always a "bottom" vertex having two upward edges out of it. However, in *T*, each vertex has at most one upward edge, so *T* can have no cycles. Therefore, *T* is a tree. □

Lemma 3 *T is a spanning tree of graph G.*

Proof: If the sub-graph *T* is connected (every vertex has a path to the root *x*) then every vertex of *G* will be present in *T*. We use induction on the length of the shortest path to *x* to prove this Lemma. If *v* has a path of length *k*, starting $v - w - \cdots - x$, then *w* has a path length of $k - 1$, and by induction would be included in *T*. Thus, when *w* is visited, edge *vw* has already been visited, and if *v* was not already in the tree, it would have been added. □

Lemma 4 *T is the shortest path tree starting from its root.*

Proof: Breadth first search trees have the property of every edge of *G* being classified into one of three groups: (i) some edges of *G* are in *T*; (ii) some edges connect two vertices at the same level of *T*; (iii) remaining edges connect two vertices on two adjacent levels. It is not possible for an edge to skip a level; therefore, the breadth first search tree *T* is a shortest path tree starting from its root. □

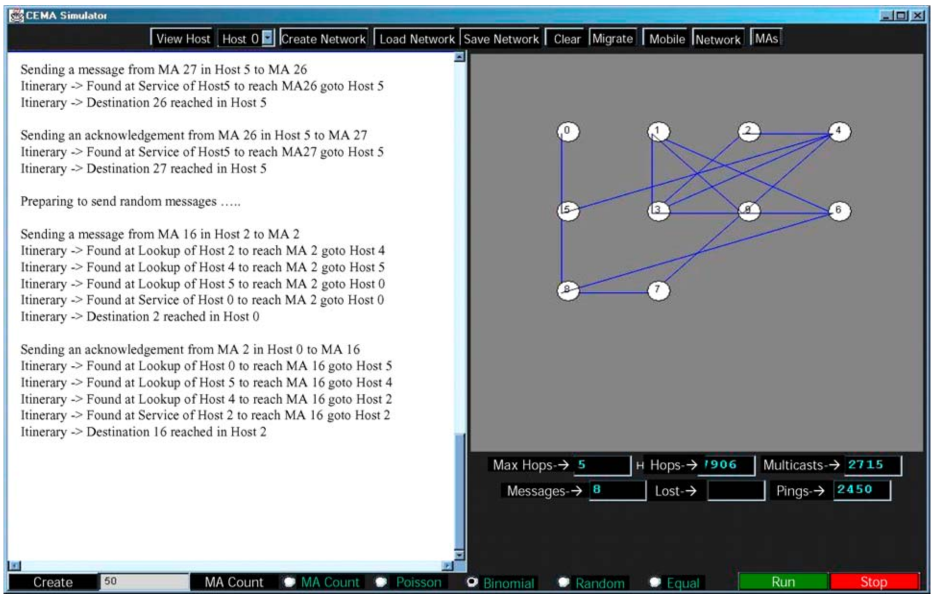
Every vertex has a path to the root, with path length equal to its level, and no path can skip a level. The maximum number of hops a normal message can take to reach its destination is dependent on the number of levels in *T* and on the location of the mobile agent. The average case for the message complexity is $O(x)$ where *x* is the number of levels in *T*. This can be proven from Lemma 4. The algorithm allows only forward movement of messages and does not allow a message to revisit an edge already visited. Thus, the numbers of levels in *T* denoted by *x* determine the average number of hops a message takes to reach from a source to a destination.

In summary, the CEMA algorithm is efficient in terms of number of messages generated (for n mobile agents launched and m agent migrations, the number of messages are slightly more than $\mathrm{m}^2 + \mathrm{mn}$. Since the algorithm basically views the nodes in the network as a tree (spanning tree) and then does a breadth-first traversal, the worst-case scenario are completely linear nodes where nodes *x* and *y* are at the opposite ends of the tree (network). Average case performance depends on the depth of the tree (from *x* to *y*) and is usually much better than worst case, while best case is the trivial case of *x* and *y* being the same node.

6. Experimental results

To investigate the performance of CEMA, a discrete event simulation has been developed in Java (Figure 11). The simulator is capable of mimicking a network with an arbitrary number of hosts, a user specified network topology and an arbitrary number of mobile agents. Events such as migrations and link failures can be created by defining their probability density function (pdf) and its parameters.

Four sets of simulation experiments have been completed, with sufficient runs for each simulation step to claim a 95% confidence level that the relative error is less than 5%, based on large generated random samples and calculations of means and deviation.



```
Sending a message from MA 27 in Host 5 to MA 26
Itinerary -> Found at Service of Host5 to reach MA26 goto Host 5
Itinerary -> Destination 26 reached in Host 5

Sending an acknowledgement from MA 26 in Host 5 to MA 27
Itinerary -> Found at Service of Host5 to reach MA27 goto Host 5
Itinerary -> Destination 27 reached in Host 5

Preparing to send random messages .....

Sending a message from MA 16 in Host 2 to MA 2
Itinerary -> Found at Lookup of Host 2 to reach MA 2 goto Host 4
Itinerary -> Found at Lookup of Host 4 to reach MA 2 goto Host 5
Itinerary -> Found at Lookup of Host 5 to reach MA 2 goto Host 0
Itinerary -> Found at Service of Host 0 to reach MA 2 goto Host 0
Itinerary -> Destination 2 reached in Host 0

Sending an acknowledgement from MA 2 in Host 0 to MA 16
Itinerary -> Found at Lookup of Host 0 to reach MA 16 goto Host 5
Itinerary -> Found at Lookup of Host 5 to reach MA 16 goto Host 4
Itinerary -> Found at Lookup of Host 4 to reach MA 16 goto Host 2
Itinerary -> Found at Service of Host 2 to reach MA 16 goto Host 2
Itinerary -> Destination 16 reached in Host 2
```

Max Hops-> 5   H Hops-> 1906   Multicasts-> 2715
Messages-> 8   Lost->   Pings-> 2450

Create 50

Figure 11. Screenshot of the CEMA simulator.

6.1. Experiments with random population and 20 agents

The first set of simulations employed 20 mobile agents with varying populations ($n = 4, \ldots, 20$) and a randomized network topology. A total of 100 messages were exchanged in each simulation step. The results of the experiment are shown in Figures 12 and 13. The maximum hops for each of the messages to reach the destination was proportional to number of levels in the BFS tree T.

The worst-case scenario occurs when the graph G is a linear graph. This has a worst-case complexity of ***O(E)***. In such a case, the number of *levels* in T is equal to the number of edges as shown in Figure 14. When two mobile agents are located on extreme ends of a linear graph, their messages will have to traverse through all the edges in order to reach the other extreme end.

6.2. Linear graph with varying populations

The second set of experiments were performed in a setup similar to that of the first set of experiments except for the topology which has been implemented as a linear graph. The results of the experiment are shown in Figures 15 and 16. The maximum hops for each of the messages to reach the destination was proportional to the number of edges in the network.

The best-case scenario is when all the mobile agents are concentrated on a single host or when G is a fully connected graph. To reach the destination a message has to take a single

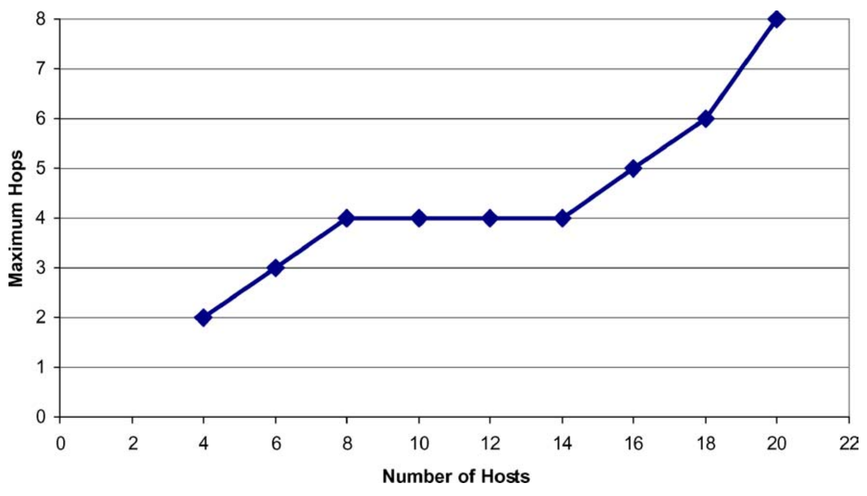


Figure 12. Number of hosts vs. maximum hops for average-case.

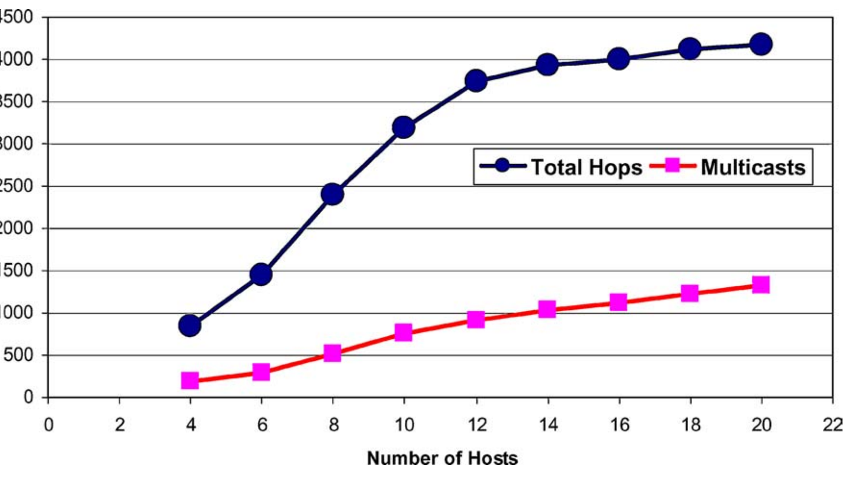


Figure 13. Number of hosts vs. total hops/multicasts for average-case.

hop or no hop at all (complexity of $\boldsymbol{O}(1)$). In this scenario, the message has to move only one level as shown in Figure 17(a) and (b).

6.3. Varying number of mobile agents

In the third set, the number of mobile agents ($m = 10, \ldots, 60$) has been varied, for different population of hosts ($n = 5, \ldots, 50$). Readings were taken for transmission of 100 randomly generated normal messages. It can be observed that as the number of hosts increase there is an increase in number of hops. In addition, there is an increase in the total number of hops

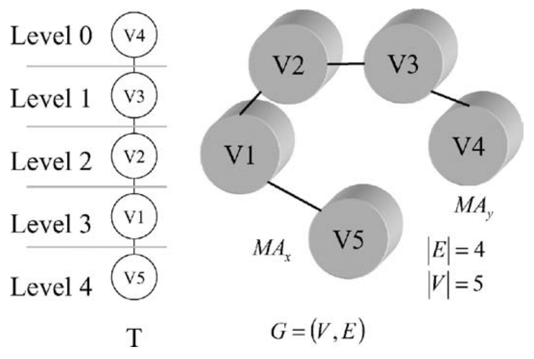


Figure 14. Worst-case scenario.

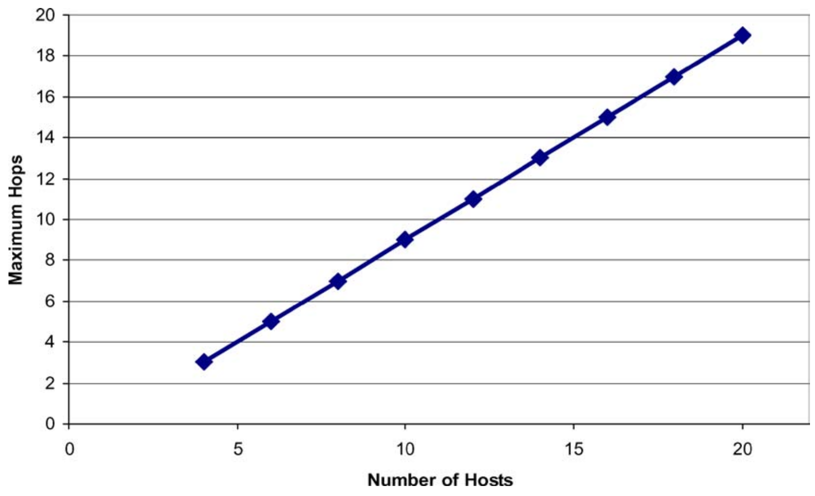


Figure 15. Number of hosts vs. maximum hops for worst-case.

as the number of mobile agents increase, but this is mainly attributed due to the increase in the number of control messages created. Figure 18 depicts the total number of hops versus to the number of hosts for the three runs with 10, 20 and 30 randomly distributed mobile agents. Please note that the hops indicated on the graph are the total number of hops (summed) for all mobile agents in the system.

6.4. *Effects of mobile agent migration*

The fourth set of experiments deployed 20 hosts with a varying number of mobile agents ($m = 5, \ldots, 30$). Three different runs were performed with 10, 20 and 30 occurrences of mobile agent migrations during the transmission of 100 randomly generated messages. Table 1 shows the various readings taken during the experiment and Figure 17 depicts the total hops taken versus the number of mobile agents. It is seen as the number of migrations

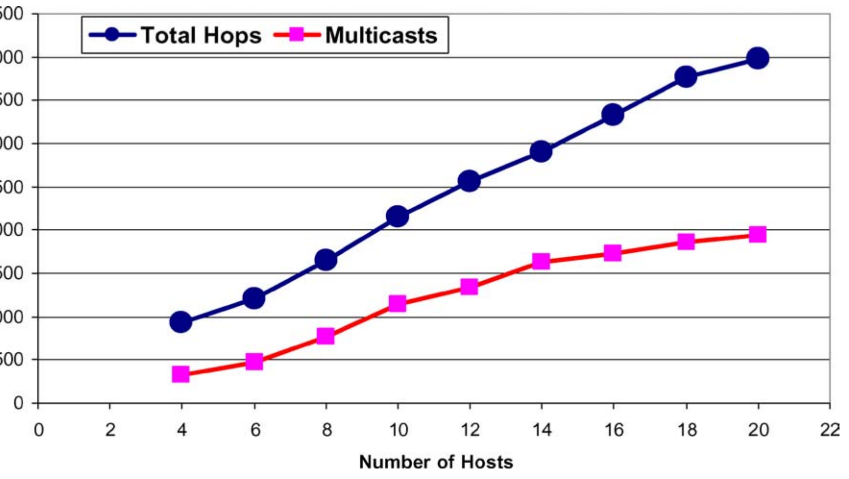


Figure 16. Number of hosts vs. total hops/multicasts for average-case.

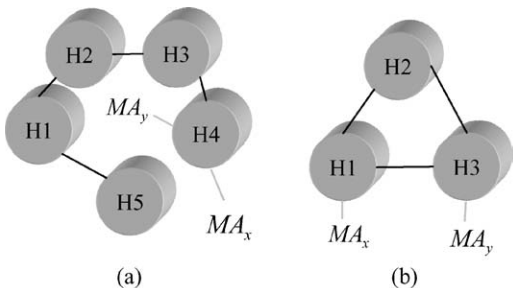


Figure 17. Best-case scenario: (a) same host and (b) fully connected graph.

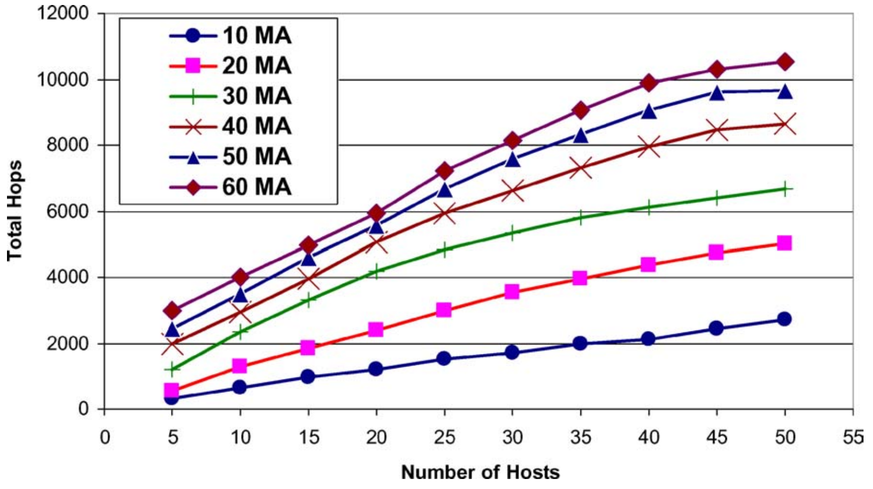


Figure 18. Number of hosts vs. number of hops.

Table 1. Experimental results of mobile agent migration simulations

Hosts	Mobile agents	Migrations	Normal messages	Control messages	Maximum hops	Total hops	Multi-casts
20	5	10	100	60	6	260	283
20	5	20	100	100	8	300	443
20	5	30	100	140	8	340	577
20	10	10	100	180	9	380	562
20	10	20	100	270	8	470	1261
20	10	30	100	360	8	560	1651
20	15	10	100	350	8	550	1461
20	15	20	100	490	8	690	1561
20	15	30	100	630	7	830	2299
20	20	10	100	570	9	770	1583
20	20	20	100	741	9	941	3076
20	20	30	100	950	8	1150	4259
20	25	10	100	864	9	1064	2922
20	25	20	100	1080	8	1280	3914
20	25	30	100	1320	9	1520	4178
20	30	10	100	1160	8	1360	3612
20	30	20	100	1450	8	1650	4352
20	30	30	100	1711	8	1911	5522

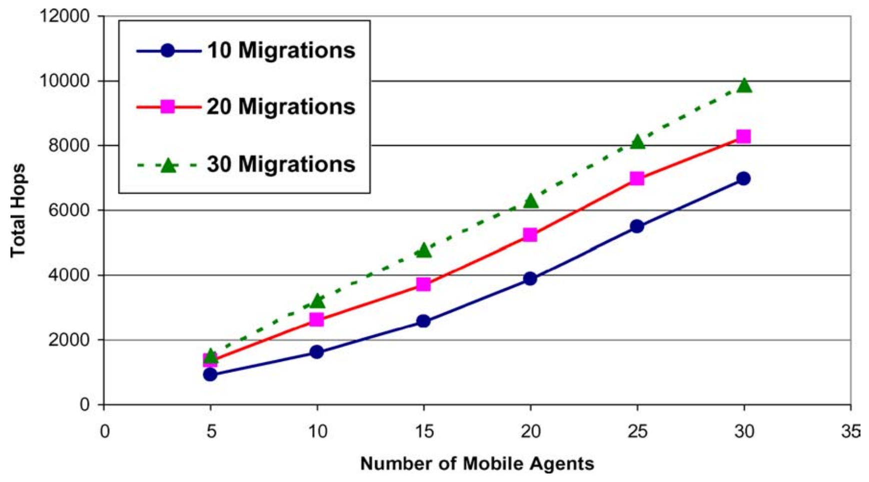


Figure 19. Number of mobile agents vs. total hops.

increase the total number of hops increase. This increase can be attributed to the extra number of control messages that are generated because of migration.

6.5. Limitations of CEMA

The CEMA algorithm cannot automatically detect link failures. If a link fails the updates to the tables have to be done by an external entity. Work is underway to extend CEMA for these scenarios. Additionally, messages can only move forward from the source to the destination due to the timestamp and due to the multicast restriction towards the originator. If a message crosses a bridge, it will not be able to return across the bridge; if meanwhile the destination mobile agent happens to migrate to the other side of the bridge, then the message is lost since it cannot go back across the bridge. This situation cannot be handled by the current algorithm and will be addressed in a subsequent paper.

7. Conclusions

This paper presented the design of a prototype software infrastructure for pervasive computing utilizing mobile agents. Mobile agents are being used in many new, ubiquitous computing systems as well as Internet computing, distributed processing and other novel domains. Architectural components of the software infrastructure were described and a novel algorithm (CEMA) for establishing connections between mobile agents in a pervasive network was presented. Analysis of the algorithm showed that worst-case performance is linear with respect to the number of nodes between a source and destination. For m mobile agent migrations and n mobile agents the number of messages sent is on the order of $\mathrm{m}^2 + \mathrm{mn}$. Two limitations that are inherent in the CEMA method are: link failures are not quickly found and some network topologies are slow to be updated. In the first case link failures are not found until an agent migrates to a node where that link is part of the communication path. In actuality, this is a reasonable penalty to pay for reduced message overhead in the normal case, and bad links will be found when there is an attempt to use one. The second case is the network topology where two sub-networks are connected via a single link (bridge). In this case CEMA's one-directional flow of messages (through the spanning tree) will cause messages to be lost and must be timed out on a node to determine that this situation has occurred. A Java based simulator was designed to evaluate the message complexity performance of CEMA. Various experiments conducted by the simulator have shown the scalability of the infrastructure as well as the limitations of the infrastructure. In experiments both average case and worst-case simulations show reasonable performance, as seen in the analytical model. Experiments also show that as agents are randomly dispersed on a network topology, and then migration is simulated, the total number of cumulative hops is better than linear (in relation to number of hosts), especially as the number of mobile agents increases (from 10 to 60). Experiments show that as the mobile agents move (migrate) through the network the total number of hops, on average, is only a little worse than linear in relation to the number of mobile agents in the network.

References

1. T. C. Agoston, T. Ueda, and Y. Nishimura. Pervasive computing in the networked world. *Proceedings of INET2000*, Yokohama, Japan, 2000.
2. ARPA External Interfaces Working Group. ARPA knowledge sharing initiative. specification of the KQML agent-communication language. ARPA Knowledge Sharing Initiative, July 1993.
3. K. S. Barber, A. Goel, D. C. Han, J. Kim, D. N. Lam, T. H. Liu, M. MacMahon, C. E. Martin, and R. McKay. Infrastructure for design, deployment and experimentation of distributed agent-based systems: The

requirements, The technologies, and an example. *Journal of Autonomous Agents and Multi-Agent Systems*, 7:49–69, 2003.

4. K. Berket and D. Agarwal. Enabling secure ad-hoc collaboration. *Proceedings of the Workshop on Advanced Collaborative Environments*, Seattle, WA, June 22, 2003.
5. J. M. Bradshaw (ed.). *Software Agents*, AAAI/MIT Press, 1995.
6. W. Cockayne and M. Zyda. *Mobile Agents*, Manning Publications, 1998.
7. G. H. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computer*, 27(6), 1994.
8. J. R. Graham, K. S. Decker, and M. Mersic. DECAF—A flexible multi agent system architecture. *Journal of Autonomous Agents and Multi-Agent Systems*, 7:7–27, 2003.
9. R. S. Gray. Agent Tcl: A transportable agent system. *Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95)*, Baltimore, Maryland, 1995.
10. A. C. Huang, B. Ling, S. Ponnekanti, and A. Fox. Pervasive computing: What is it good for? *Proceedings of the Workshop on Mobile Data Management (MobiDE) in Conjunction with ACM MobiCom*, 1999.
11. D. Kotz, R. Gray, S. Nog, D. Rus, S. Chawla, and G. Cybenko. AGENT TCL: Targeting the needs of mobile computers. *IEEE Internet Computing*, 1(4):58–67, 1997.
12. M. J. Kumar, B. A. Shirazi, S. K. Das, B. Sung, D. Levine, and M. Singhal. PICO: A middle framework for pervasive computing. *IEEE Pervasive Computing Magazine*, Oct. 2004.
13. Y. Labrou, T. Finin, and Y. Peng. The interoperability problem: Bringing together mobile agents and agent communication languages. *Proceedings of the Hawaii International Conference On System Sciences*, Jan. 1999, Maui, Hawaii, 5–8.
14. D. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley Publishers, 1998.
15. D. Levine, R. Thomas, F. Kamangar, and G. V. Zaruba. Mobile agents for pervasive computing using a novel method of message passing. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'03)*, Las Vegas, June 2003.
16. W. Li. *Java-To-Go*, UC Berkeley, http://ptolemy.eecs.berkeley.edu/∼ wli/group/java2go/java-to-go.html.
17. W. Li. *Java-To-Go*, Univ. of California, Berkeley, URL = http://ptolemy.eecs.berkeley.edu/ ∼wli/group/java2go/java-to-go.html.
18. MIT Oxygen project,. http://oxygen.lcs.mit.edu, 2003.
19. C. Okino and G. Cybenko. Modeling and analysis of active messages in volatile networks. *Proceedings of the 37th Allerton Conference on Communication Control & Computing*, Monticello, IL, Sept. 1999.
20. OMG Mobile Agent Systems Interoperability Facilities Specification (MASIF), OMG TC Document ORBOS/97-10-05
21. A. Orso, G. Vigna, and M. J. Harrold. MASSA: Mobile Agents Security through Static/Dynamic Analysis. *ICSE Workshop on Software Engineering and Mobility*, May 2001.
22. K. Paul and S. Bandyopadhyay. Evaluating the performance of mobile agent based message communication among mobile hosts in large ad-hoc wireless networks. *Proceedings of the Second ACM International Workshop on Modeling and Simulation of Wireless and Mobile Systems* (In conjunction with IEEE/ACM MobiCom'99), Seattle, WA, Aug. 1999.
23. M. Satyanarayanan. Pervasive computing: Visions and challenges. IEEE Personal Communications, 2001.
24. T. Starner. Wearable agents. *IEEE Pervasive Computing*, 1(2):90–92, 2002.
25. R. Thomas. Mobile agents for pervasive computing. Master's Thesis, The University of Texas at Arlington, May 2002.
26. R. M. Van Eijk, F. S. De Boer, W. Van Der Hoek, and J.-J. Ch. Meyer. A verification framework for agent communication. *Journal of Autonomous Agents and Multi-Agent Systems*, 6:185–219, 2003.
27. M. Weiser. The computer for the twenty-first century. *Scientific American*, 94–104, 1991.