Configurable Network Processing for Mobile Agents on the Internet

Ichiro Satoh

National Institute of Informatics /
Japan Science and Technology Corporation
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
E-mail: ichiro@nii.ac.jp

Abstract

This paper presents a framework for building and deploying protocols for migrating mobile agents over the Internet. The framework enables network protocols for agent migration to be naturally implemented within mobile agents and then dynamically deployed at remote hosts by migrating the agents that perform the protocols. It is built on a hierarchical mobile agent system, called MobileSpaces, and several protocols for migrating agents for managing cluster computing systems have been designed and implemented based on the framework. This paper describes the framework and its prototype implementation, which uses Java as both the implementation language and the protocol development language.

1 Introduction

Mobile agent technology can play an important role in the management of cluster computing. Mobile agents are autonomous programs that travel from computer to computer under their own control. Their abilities to cope with system heterogeneity and to deploy user-customized procedures at remote sites are well suited for cluster computing environments. By interacting with a remote host after migrating to it, an agent can perform complex operations on remote data without transferring them, because the agent can carry the application logic to where it is needed. A mobile agent for monitoring a cluster computing system can travel to multiple nodes in a network to observe the components locally. This emerging technology will make it much easier to manage and customize cluster computing systems. Although this technology has been expected to be used to manage cluster computing on the Internet, there have few attempts to apply it there. The reason for this is a lacking for the ability to customize network processing for migrating agents on the Internet as well as the problem of protecting against malicious agents and malicious hosts. That is, most existing mobile agent systems are essentially designed to be used within an intranet, instead of on the Internet; they are also often dependent on the network protocols such as Java RMI and HTTP. In almost all intranet situations, there is a firewall that prevents users from establishing a direct socket connection to/from an external node, except for certain ports, so some of the systems cannot migrate any mobile agents through the firewall. Moreover, several applications require application-specific network processing for migrating their agents over a

network. For example, a mobile agent for electronic commerce needs to be transformed into an application-specific encrypted bit stream before transferring itself over a network, including the Internet. However, existing mobile agent systems cannot dynamically adapt their network processing to the requirements of each visiting agent because their network processing is statically embedded with them.

This paper describes a mechanism for dynamically configuring network processing for agent migration in cluster computing environments on the Internet. A framework is described for dynamically deploying and changing network protocols for agent migration according to the requirements of each visiting agent and the external environment. This framework introduces the notion of *network processing of mobile agents, by mobile agents, for mobile agents*. The notion enables network processing for migrating mobile agents to be implemented as a set of mobile agents. That is, mobile agent-based protocols transmit mobile agents as first-class objects to their destinations. Dynamic deployment of these protocols can be naturally and easily performed by the migration of the agents that support them. Therefore, this framework provides a useful infrastructure and testbed for implementing and evaluating different types of network processing for migrating agents. Moreover, it enables network processing for mobile agents to be adapted to the requirements of visiting agents and to changes in the environment.

This paper is organized as follows. Section 2 explains our framework of customizable network processing for migrating agents for managing cluster computing environments. Section 3 briefly reviews our mobile agent system, *MobileSpaces*, and Section 4 presents several mobile agent-based protocols. Section 5 illustrates an application to cluster computing to demonstrate the usability of the framework. Section 6 surveys related work, and Section 7 provides a summary and mentions future issues.

2 Architecture

The framework presented in this paper provides a self-configuring infrastructure of network processing for mobile agents.[1] It can deploy and configure network protocols for agent migration according to the requirements of visiting agents and to changes in the network environment. This section outlines the overall architecture of the framework and describes the basic idea of adaptive protocols based on the framework.

2.1 Configurable Network Processing

The framework is built on a mobile agent system, called MobileSpaces [12]. The system is characterized by two novel concepts: **agent hierarchy** and **inter-agent migration**, which are illustrated in Figure 1. The former means that one mobile agent can be contained within another mobile agent. That is, mobile agents are organized in a tree-like structure. The latter means that each mobile agent can migrate to other mobile agents along with all its inner agents. Each agent can freely move into any agent in the same agent hierarchy except into itself or its inner agents, as long as the destination agent accepts it. A container agent is responsible for automatically offering its own services and resources to its inner agents and can control its inner agents. Therefore, an agent can directly instruct its inner agents to move to another location.

[1] This paper focuses on network processing for agent migration among nodes in a network instead of for exchanging data and system management messages.

Our mobile agents can transmit other mobile agents as first-class objects [4], in the sense that mobile agents can be passed to and returned from other mobile agents as values by their container agents. Such container agents are still mobile and thus can transport their inner agents to other agents. As a result, our framework allows various operations for mobile agents, including network processing for agent migration, to be naturally constructed and performed by other mobile agents. That is, network protocols for agent migration are implemented within mobile agents. Accordingly, mobile agents are introduced as the only constituent of our network processing for mobile agents. Moreover, such protocols can be dynamically and easily changed by migrating agents that implement the protocols at destinations and intermediate nodes. Therefore, our framework offers a self-configurable infrastructure of network processing for mobile agents. Furthermore, the agents can be constructed and reused through a single programmable abstraction for composition and refinement of mobile agents. As a result, the framework greatly simplifies the development of active networks.

2.2 Agent Migration Protocols for the Internet

Current protocols for data transmission are often arranged in a hierarchy of layers. Each layer presents an interface to the layers above it and extends services provided by the layer below it. The hierarchical structure of mobile agents enables network protocols for agent migration to be organized hierarchically. That is, each agent hierarchy consisting of mobile agent-based protocols can be viewed as a protocol stack for agent migration, and agent migration in an agent hierarchy is introduced as a basic mechanism for accessing services provided by the underlying layers. Mobile agent-based protocols in the bottom layer correspond to data-link layer protocols. They are responsible for establishing point-to-point channels for agent migration between neighboring computers. The middle layer corresponds to routing protocols for agent migration and provides two mechanisms for transmitting mobile agents beyond the channels between directly connected nodes. The framework enables routing protocols for agent migration to be handled by mobile agents. The first mechanism allows a mobile agent to carry other mobile agents among nodes agents according to its own itinerary and the second allows a mobile agent to forward other mobile agents to nodes. Both mechanism are built on transmitter agents running on nodes.

3 MobileSpaces: A Runtime System for Hierarchical Mobile Agents

Here we briefly review the MobileSpaces system [12], which provides an infrastructure for building and executing mobile agents for network processing in addition to mobile agent based-applications. It supports mobile agents that obey the notions of agent hierarchy and inter-agent migration presented in the previous section.

This mobile agent system can dynamically adapt itself to changes in the network environment because it is based on a micro-kernel architecture, like several operating systems. That is, the system consists of two parts: a core system and subcomponents. The former offers only minimal, common functions, independent of the underlying environment. It is thus independent of the network infrastructure.[2] The latter is a collection of subcomponents outside the core system that

[2]The current implementation of the core system has a built-in mechanism for transmitting agents over the network

provide other functions, such as agent migration between computers and persistence of secondary storage, which may depend on the surrounding environment. These subcomponents, including network processing for agent migration, are implemented as mobile agents so that they can be dynamically added to and removed from the system by migrating and replacing the corresponding agents.

3.1 Core System

Each core system runs on a computer and is made as small as possible. It offers only three facilities.

Agent Hierarchy Management: Each runtime system has an agent hierarchy, which is maintained as a tree-like structure in which each node contains a mobile agent and its attribute. The system itself corresponds to the root node of its own agent hierarchy. Agent migration in an agent hierarchy is performed simply as a transformation of the tree structure of the hierarchy. A container agent is introduced as a service provider for its inner agents. Each agent offers a collection of service methods that can be accessed by its inner agents. Each agent is active but subordinate to its container agent. That is, a container agent can instruct its inner agents to move to other agents or computers, and it can marshal and terminate them.

Agent Execution Management: Each mobile agent can have more than one active thread under the control of the core system. The core system maintains the life-cycle state of mobile agents. When the life-cycle state of an agent is changed, for example, creation, termination, or migration, the core system issues events to invoke certain methods in the agent and its contained agents. The core system can explicitly limit the length of an agent's visit and the number of staying agents. When the time limit of a staying agent expires, it automatically terminates the agent. This limitation offers a mechanism for caching network protocols.

Agent Verification Management: The current implementation of the system uses the Java object serialization package for marshaling the states of agents, so agents are transmitted based on the notion of weak mobility [5]. The core system checks whether or not a marshaled agent is valid to protect the system against invalid and malicious agents, by means of Java's security mechanism. Enriched security mechanisms for agents can be implemented by mobile agents outside the core system. Also, the core system provides a facility for marshaling agents into bit streams and unmarshaling them later.

3.2 Mobile Agent Program

The mobile agents introduced here are programmable entities like other mobile agents. As shown in Figure 2, each agent consists of three parts: a body program, context objects, and inner agents. The body program is an instance of a subclass of abstract class `Agent`.[3] The `Agent` class provides a command for agent migration in an agent hierarchy, written as `go(AgentURL destination)`. When an agent performs the command, it migrates itself to the destination agent

by using an extension of HTTP running on TCP/IP.

[3]Examples of mobile agent programs are given in the Appendix.

specified as the argument of the command. An inner agent cannot access any methods defined in its container agent. Instead, each container agent is equipped with a context object that offers service methods in a subclass of the `Context` class, like the `AppletContext` class of Java's Applets. These methods can be indirectly accessed by its inner agents to get information about and interact with the environment, such as with their container agent, their sibling agents, and the underlying computer system. Each inner agent can invoke the public methods defined in the context of its container agent via several built-in application programming interfaces. Each agent must be informed of lifecycle state changes so that it can release various resources, such as sockets and files, which are reserved by the agent. To hook events invoked by the core system, before or after the life-cycle of a mobile agent changes, the agent provides listener objects, which define callback methods invoked by the core system and its container agents at certain timings.

Each agent is associated with a resource limit that functions as a generalized Time-To-Live field. This limit is carried with the agent and is decremented by nodes as resources are consumed when the agent arrives at a new place. Nodes can discard agents when their limit reaches zero. To restrict the total resource bounds, when one agent creates another inside the network, the resources allocated to each created agent must be less than those of the creating agent.

4 Agent Migration Protocols in the Internet

Since the MobileSpaces core system supports only functions independent of the underlying environment, other functions, including agent migration between different computers, must be provided by mobile agents outside the core system. The core system introduces each mobile agent as a service provider, because a mobile agent is designed to provide service to its inner agents. When a mobile agent is preparing to traverse over a network, the agent migrates itself into a mobile agent that provides appropriate network processing in the same agent hierarchy, and then the agent automatically transfers the visiting agent (or migrates itself) to its destination, or delegates the task to another mobile agents in the same agent hierarchy.

4.1 Point-To-Point Channels for Agent Migration

The framework introduced here enables point-to-point agent migration to be provided by mobile agents, called *transmitters*, instead of by the core system. Transmitter agents correspond to a data-link layer or a network layer and are responsible for establishing point-to-point channels for agent migration between the source host and destination host through a (single-hop or multiple-hops) data transmission infrastructure, such as TCP/IP, as shown in Figure 3. They abstract away the variety in the underlying network infrastructure and exchange their inner agents with coexisting agents running at remote computers through their favorite communication protocols. Furthermore, transmitter agents are implemented as mobile agents so that they can be dynamically added to and removed from the system by migrating and replacing the corresponding agents, enabling them to keep up with changes in the network environment. After an agent arrives at a transmitter agent from the upper layer, the arriving agent indicates its final destination. The transmitter suspends the arriving agent (including its inner agents), then requests the core system to serialize the state and code of the arriving agent. Next, it sends the serialized agent to a coexisting transmitter agent located at the destination. The transmitter agent at the destination receives the data and then reconstructs the agent (including its inner agents) and migrates it to the destination or to specified

agents for offering upper-layer protocols.

Since each runtime system can be equipped with more than one transmitter agent, upper-layer protocols can dynamically specify a suitable agent and migrate their inner agents to the selected transmitter agents. Several transmitter agents have already been implemented based on data communication protocols widely used in the Internet, such as TCP, HTTP, and SMTP. Authentication services normally available in secure communications infrastructure include this functionality. Therefore, our secure transmitter agents, which can exchange agents with each other, are implemented with secure socket layer (SSL), which is one of the most popular secure communication protocols in the Internet. A virtual class is provided in Java that can be specialized to create transmitter agents for various protocols. Therefore, point-to-point channels can easily be implemented based on other secure communication protocols for data transmission.

Example: Authentication for Agent Migration

Before migrating an agent to its destination host, the moving agent, the source host, and the destination host should be authenticated. The current implementation provides a Kerberos-based authentication mechanism for agent migration. Kerberos [17] provides secure authentication services, provided the Kerberos server itself is trusted. It authenticates users without exposing their passwords on the network and generates secret encryption keys that can be selectively shared between mutually suspicious parties. It also enables roaming mobile agents to authenticate themselves in foreign domains where they are unknown, thus enhancing the scale of mobility. The methods used here have also been devised to use Kerberos for authorization control and accounting before establishing a TCP connection to transmit mobile agents from the source host to the destination host. This system consists of authentication servers and a ticket-granting server. Each authentication server verifies users during login, and the ticket-granting server issues proof of identifier tickets. Transmitter agents are assumed to be allocated at the source host and the destination host. A transmitter agent at the source host requests the authentication server and the ticket-granting server for a session key. After receiving the key from the ticket-granting server, the transmitter agent migrates its inner agents to the destination. The current implementation is susceptible to off-line password guessing attacks and to replay attacks for a limited time window.

4.2 Application-Specific Routing for Agent Migration

A mobile agent often must visit multiple nodes to perform its task, so it must make an application-specific and network-dependent itinerary. On the other hand, channels between transmitter agents support point-to-point agent migration. Therefore, mechanisms are needed to migrate an application-specific mobile agent among multiple nodes so it can perform its task. However, it is difficult to determine the itinerary at the time the agent is designed or instantiated. In addition, even if an agent were optimized for a particular network, it might not be reusable in another one. Therefore, we introduce two approaches for determining and managing the itinerary of agents. Both approaches are built on transmitter agents running on nodes.

- The first approach provides a function similar to that of routers. A service provider, called a *forwarder* agent, redirects moving agents to new destinations. Forwarder agent stays at nodes. When receiving agents, it redirects the agents to their destinations through point-to-point channels established among multiple nodes as shown in Figure 4. Each forwarder

agent has a table describing part of the network structure. It is built by using management rules based on existing routing algorithms for packets, including shortest path routing and distance vector routing. The forwarder agents are dynamically deployed at nodes and coordinate with each other to redirect moving agents to their destinations. However, if a destination is not reachable, they try to transfer the moving agents to other forwarder agents running on intermediate nodes as near the destinations as possible. Each forwarder agent will repeat the entire process in the same way until it arrives at the destination. The current implementation allows forwarder agents to negotiate with each other through data transmission protocols such as TCP/IP.

- The second approach is similar to the notion of an active packet (also called a programmable capsule) in active network technology. Existing mobile agents can move from one node to another under their own control, just as active packets can define their own routing. A service provider, called a *navigator*, conveys inner agents over a network, as shown in Figure 5. Each navigator agent is a container of other agents and travels with them to a sequence of nodes statically or algorithmically determined, or dynamically based on the agent's previous computations and the current environment. That is, a navigator agent can migrate itself to the next node along with all its inner agents. Each navigator has a routing mechanism for managing a routing table consisting of nodes that the navigator agent needs to visit. It maintains a list of nodes to be visited and provides methods for dynamically adding and removing elements from this list.

The interaction between a forwarder agent or navigator agent and its inner agents is based on event-based communication. Upon receiving agents, a forwarder propagates certain events to its visiting agents instructing them to do something during a given time period. After the events have been processed by the inner agents, the forwarder agent transmits the agents another forwarder agent running at the next node. Upon arriving at a node, a navigator agent propagates certain events to its inner agents. After the events have been processed by the inner agents, the forwarder or navigator continues on its itinerary. Since the both approaches hide the description of an agent's itinerary from its behavior, mobile agents become independent of the network structure and the modularity and reusability of application-specific mobile agents are enhanced. Although forwarder agents must be deployed at many intermediate nodes, they do not seriously affect the performance of the whole system, including scalability, because they basically perform their tasks only when receiving agents.

The forwarder and navigator agents are useful in the authentication of mobile agents. This is because each forwarder agent can limit the forwarding range of its inner agents and receive only those agents moved by authorized forwarder agents. Therefore, each node can explicitly reject any agents from unauthorized nodes. Each navigator agent can define its own reachable nodes, and each node can accept only authorized navigator agents. That is, when an agent is moved by a navigator agent whose reachable nodes are limited, it can travel only among the reachable nodes of the navigator agent.

Example: Locating Mobile Agents

When an agent wants to interact with another agent, it must know the current location of the target agent. Therefore, a mechanism is needed for tracking a moving agent. An extension of the forwarder agent offers such a mechanism as shown in Figure 6. Immediately before an agent moves

into another agent, it creates and leaves a forwarder agent behind. The forwarder agent inherits the name of the moving agent and transfers visiting agent to the new location of the moving agent. Therefore, when an agent wants to migrate to another agent that has moved elsewhere, it can migrate into the forwarder agent instead of the target agent. The forwarder agent then automatically transfers it to the current location of the target agent. Each forwarder agent can record the identifiers of its moving agents that it receives so that it prevents circular forwarding of agents. Several schemes for efficiently locating mobile agents have been explored in the field of process/object migration in distributed operating systems. Such forwarder agents can easily support most of these schemes because they are programmable entities that can flexibly negotiate with each other. The current implementation provides forwarder agents for two efficient schemes: 1) each forwarder's registering the new location of every agent moved to a specified forwarder, which corresponds to a name server and 2) a specified forwarder querying other forwarders about the presence of every agent, as well as simply forwarding moving agents based on the trails left during an agent's migration.

Forwarder agents can be also used to handle network disconnection by using the notion of store-and-forward migration, which is similar to the process of transmitting electronic mail by using SMTP. When an agent requests such a forwarder agent at the source to migrate to its destination, the forwarder agent tries to transmit the moving agent to the destination through transmitter agents. If the destination is not reachable, the forwarder agent automatically stores the moving agent in its queue and then periodically tries to transmit the waiting agent to either the destination or another forwarder agent on a reachable intermediate node as close to the destination as possible. These relay agents repeat the process until the agent arrives at its destination.

Example: Encrypting Mobile Agents

An agent should be encrypted before migrating itself over the Internet for security reasons. Since each navigator agent can treat its inner agents as first-class data, it can transform them before carrying them to their destinations. The current implementation provides a special navigator agent, called a safe agent, which can be viewed as a cash transport truck. It is a container of other agents and has a secret-key based cryptographic procedure inside it. When an agent visits a safe agent, the safe agent automatically serializes and encrypts the visiting agent under a secret key. Next, it migrates itself to the destination as a whole with all its inner agents and its cryptographic procedure except for its secret keys. After arriving at the destination, the safe agent keeps its inner agents inside it. If the destination provides its privacy to the visiting safe agent, the agent decrypts its inner agents. Safe agents can be implemented independently of cryptographic algorithms because the algorithms should be selected according to the requirements of the application. Each safe agent has an interface for hiding the differences between secret-key based cryptographic algorithms and can support any algorithms that satisfies the interface. A non-standardized cryptographic algorithm can be embedded into a safe agent without losing any interoperability, because the agent can carry the procedure for the algorithm and perform it at both the source node and destination node.

4.3 Protocol Distribution and Configuration

Given a dynamic network infrastructure, a mechanism is needed for propagating mobile agents that support protocols to where the agents are needed. The current implementation of this frame-

work provides three mechanisms: (1) mobile agent-based protocols autonomously migrate to nodes at which the protocols may be needed and remain there in a decentralized manner; (2) mobile agent-based protocols are passively deployed at nodes that may require them by using forwarder agents prior to using the protocols as distributors of protocols; and (3) moving agents can carry mobile agent-based protocols inside themselves and deploy the protocols at nodes that the agents traverse. This last mechanism improves performance in the expected common case of agent migration, i.e., a sequence of agents that follow the same path and require the same processing. All the mechanisms are managed by mobile agents instead of by the runtime system and thus can be customized easily.

Since the purpose of our framework is to support mobile agent-based management for cluster computing systems rather than data transmission, it basically uses an on-demand approach: mobile agents can be downloaded on demand and cached for future use, as in several existing active networks. Mobile agent-based protocols identify their type and the protocol groups to which they belong as they travel. When a mobile agent arrives at a node, the cache of agents is checked. If the required protocol is not present, a request to fetch the missing protocol is sent to the node from which the agent arrived. The transmission of the arriving agent is suspended, awaiting the protocol, for a finite time. Transmitter agents are deployed by other transmitter agents. Furthermore, the framework supports a static scheme in which agents for processing network protocols are downloaded before they are needed, or simultaneously migrated with the agents that need to be processed by the protocols. For practical reasons, the current implementation provides a built-in transmitter agent that can deploy other agents to specified nodes through extended HTTP running on TCP/IP. It offers the bootstrapping capability needed to install other protocols.

4.4 Current Status

The framework presented here and its mobile agent-based protocols were implemented on MobileSpaces in the Java language. The framework and protocols can be run on any computer with a JDK 1.2-compatible Java runtime system. The framework provides several useful libraries for constructing network protocols within mobile agents. Several mobile agent-based protocols were developed, in addition to the protocols presented in the next section. They include agents for establishing channels through TCP, HTTP, and SMTP, navigator for carrying other agents according to their own itineraries, and forwarder agents for migrating other agents among multiple computers according to their own static routing tables, and SNMP agents at each computer. The current implementation of this framework was not built for performance. However, to compare the forwarder and the navigator agent protocols we measured the per-hop latency and the throughput of a single node in agents per second in a network consisting of ten PCs (Intel Pentium III-600MHz with Windows 2000 and JDK 1.3) connected by 100-Mbps Ethernet via a switching hub.

Table 1 shows the basic performance of agent migration over a network. The latency through two computers and the throughput of a single node in agents per second were measured. In both cases, we measured minimal agents, which consist only of the common callback methods invoked at the changes of their life-cycle states by the core system.[4] They correspond to a null RPC and have a data size of about 2.5 Kbytes (zip-compressed).

The first row in Table 1 shows the cost of agent migration between two simple transmitter agents, the code of which is shown in the Appendix. Transmitter agent running on nodes ex-

[4] A moving agent is a simple implementation of the `DefaultEventListener` interface presented by the author [12].

changes the code and state of their inner agents with each other through an application-level protocol for agent transmission over a TCP channel. A marshaled agent consists of its serialized state, its code, and its attributes, such as name and capability; it is packed and compressed into a bit-stream. The latency shows the total time of the marshaling, zip-based compression, TCP connection opening, transmission, security verifications, decompression, and unmarshaling. The cost of sequentially relaying a minimal agent from one forwarder agent to another forwarder agent was measured when running eight computers. The second low in Table 1 shows the average cost of one-hop transmission based on forwarder agents. Each forwarder agent determined the computers that its inner agents was to visit at their next hops, according to its own routing tables maintained by periodically polling the routing table of the SNMP agent, and then delegated a transmitter agent to dispatch each inner agent. The third row in Table 1 shows the cost of one-hop agent migration by using a simple navigator agent which has an itinerary list of eight computers. It migrates itself and its inner agents to the computers sequentially by using the transmitter agents described above. The code of the navigator agents is shown in the Appendix.

The latency of agent migration in this framework is reasonable for a high-level prototype that uses adaptive protocols for agent migration instead of data communication. However, the latency costs can be reduced because the latencies in Table 1 basically depend on the protocols rather than the MobileSpaces system and all the protocols used in these experiments were made as simple as possible without any optimization, since the intention of the current implementation was to estimate the basic costs of our framework. The measured throughputs are limited by the MobileSpaces system and the underlying operating system.[5] Also, when any of these approaches migrate more than one mobile agent, congestion at each computer is occasionally unbalanced between some computers, because the agent-based protocols are performed asynchronously.

The performance of our mobile agent-based protocols is inferior to that of conventional protocol implementations for data transmission, including TCP/IP. However, the goal of the framework is to provide an infrastructure for dynamically customizing network processing for agent migration and the latency and throughput of the current implementation are sufficient for the migration of application-specific agents, which are programmable entities rather than passive data. Comparison of the forwarder and the navigator agent approaches showed that the former is better because the latter needs to transfer two mobile agents to the destination. Although the former needs more agent migration in agent hierarchies than the latter, the overhead of agent migration in a hierarchy is less than a few milliseconds.

5 Application to Cluster Computing

To evaluate the effectiveness of this framework, we developed a network management system for a cluster computing environment consisting of three sub-networks and each sub-network has from four to eight processor elements distributed geographically.[6] The management system dynamically deploys user customized procedures at remote nodes and monitors network and computational resources at the nodes.

The management system divides mobile agents into two types: task agents and routing agents.

[5]The performance of the current implementation highly depends on the cost of thread creation in the underlying operating system.

[6]The environment is small in scale because it is implemented as a testbed for developing middleware and applications for cluster computing rather than as a computational infrastructure.

The former are application-specific agents that perform management tasks at each of the nodes it visits. They correspond to user customized procedures that should be dynamically installed and performed at remote nodes; they include application-specific software for cluster computing and management software for system monitoring. The routing agents are either forwarder agents or navigator agents. They do not have any application-specific tasks. Instead, they are designed for particular sub-networks and can forward or carry task agents in the sub-networks they cover. They allow task agents to efficiently migrate among multiple destinations in a sub-network, because they are statically optimized for the topology of their target sub-networks.

For example, a task agent for dynamically installing application-specific procedures at remote nodes contains the procedures and is migrated from node to node by its navigators or forwarder agents. Whenever the task agent visits a remote node, it deploys the procedures at the node. A task agent for monitoring network traffic loads performs its task at each node that it visits as shown in Figure 7. Although the system itself is independent of any network management protocols, a task agent was constructed that can access SNMP data from a small stationary agent at each node visited. The stationary agent allows the visiting task agent to access the MIB of its node through interagent communication. Since the task agent contains code to perform both information retrieval and filtering, it can carry only relevant information. The system also has three other task agents for monitoring computational resources at processor nodes. They are designed to collect information on the use of CPU, memory, and disks by using performance monitoring systems at the nodes.

Each task agent selects a navigator or forwarder agent according to its requirement. The management system can deploy a repository database, called Agent Pool, of forwarder and navigator agents at a special node. The database records the capabilities of the navigator and forwarder agents at all the nodes on the system. Navigator agents carry more than one task agent and returns to the database node after completing their navigation tasks to wait for the next task. The database can be viewed as a pool for storing idle navigator agents. When a task agent arrives at a sub-network, if it knows the topology of the sub-network, it traverses over the sub-network according to its own itinerary. Otherwise it migrates itself to the database node of the sub-network to find a suitable routing agent.[7]

The total size of a navigator agent containing one task agent is about 4 KB (zip-compressed), which is only 20 percent greater than the size of a self-contained task agent that controls its own itinerary. This is a small increase in size taking into account the amount of data such agents can collect from nodes. Although the cost of collecting information about computational resources at each of the nodes depends on the amount of information, it is less than a few ten milliseconds within this system.

Early experience with this system suggests that the framework presented here enables each task agent to be built independently of any sub-network and to move efficiently among multiple nodes by using navigator agents. That is, the task agents can be reused in other networks, and their programs are relatively simple because they contain no specific knowledge about sub-networks. This is important in the management of cluster computing because the topology of a cluster computing system changes often as computers are added to or removed from the system. On the other hand, it is difficult for a mobile agent to dynamically generate an efficient itinerary among multiple nodes, while a mobile agent optimized for particular networks can efficiently travel among

[7]The selection mechanism of the current implementation compares the reachable nodes of all the forwarder and navigator agents stored in the database with the list of the nodes that the task agent must visit. This mechanism is beyond the scope of this paper and described in the author's previous paper [15].

nodes on its target networks but cannot be reused in other networks.

There have been several attempts to apply mobile agent technology to cluster computing. Most of them allow mobile agents for system and network management to travel among nodes on a cluster computing system as in the system described here. However, their mobile agents need to be transmitted by their runtime systems whose agent migration protocols are statically embedded into the systems. Therefore, they can customize their network processing to the network environment of the target cluster computing systems. On the other hand, our framework can easily and naturally configure agent migration protocols according to the requirements of the applications and the network environments, even while the agents are moving.

6 Related Work

Many mobile agent systems have been developed over the last few years, for example, Aglets [8], Telescript [19], and Voyager [11]. To our knowledge, none of them can dynamically extend and adapt their network processing for agent migration to the characteristics of current networks and to the requirements of respective visiting agents, although mobile agents must be used in heterogeneous and dynamic network environments, for example, in personal mobile communication, wireless networks, and active networks. This is because their agent migration protocols are statically embedded in their systems.

The framework presented in this paper changes network processing by combining it with active network technology [18]. There have been many attempts to apply mobile agent technology to the development of active networks [2, 3] because mobile agents can be considered a special case in mobile code technology, which is the basis of existing active network technologies. For example, the Grasshopper system offers an active network platform consisting of stationary and mobile agents as service entities for telecommunication. In contrast, the framework presented in this paper applies active network technology to mobile agent technology.

Moreover, there have been many reported attempts to customize processing in the literature of meta-level and self-reflective architectures, instead of using mobile agent technology. However, their customization mechanisms are often so complex that it is difficult to construct them and make them accessible and secure. We need to construct a simple and natural approach to configuring and adapting network processing for agent migration. To satisfy this requirement, the approach presented in this paper uses agent migration as a mechanism for deploying and managing agent migration protocols, because agent migration is a key mechanism in mobile agent computing. Several studies attempted to build smart mobile agents that can dynamically learn the topology of networks, (see, for example [9]). However, most of these studies explicitly and implicitly assume to be performed on only simulated networks. Even if the studies could be performed in a real system, the costs of generating efficient routes tend to be large.

There have been a few approaches for building configurable protocols for agent migration rather than for data transmission. A mobile agent, which visits multiple hosts to perform its task, must have an application specific itinerary. For example, a mobile agent may roam over more than one host without making any detours or may have to return to its home host after each hop instead of proceeding to another destination. Also, a network-dependent itinerary is often needed for a mobile agent to travel to multiple hosts efficiently. However, it is difficult to determine such an itinerary at the time the agent is designed or instantiated because the network topology cannot always be known. Also, even if the itinerary of a mobile agent were optimized

for a particular network to travel to multiple hosts efficiently, it might not be reusable in another network. To overcome this problem, ADK [7] separates the travel itinerary of an agent from its behavior by building a mobile agent from a set of component categories: navigational components responsible for a travel itinerary and performer components responsible for executing one or more management tasks on each node. Aglets [8] introduces the notion of an itinerary pattern, which is similar to design patterns in software engineering, to shift the responsibility for navigation from an application-specific agent to a framework library [1]. Both approaches allow us to design the application-specific itinerary for an agent independent of the logical behavior of the agent, but the itinerary parts must be statically and manually embedded in the agent. Consequently, the agent cannot dynamically change its itinerary and cannot travel beyond its familiar networks, unlike our framework. Also, the approaches cannot configure the protocols that transmit mobile agents between two nodes, where our framework can dynamically customize such protocols by using our transmitter agents and forwarder agents.

A portable and extensible mobile agent system MobileSpaces [12] serves as the basis for the framework presented in this paper. It can dynamically adapt its functions and structures to changes in the environments. Also, we presented a architecture for building several agent migration protocols in [13, 14]. That architecture is hierarchically organized like a protocol stack in existing data transmission protocols, and serves as the basis for the framework presented in this paper. While the previous papers are to propose the architecture itself, this paper is to propose mobile agent-based protocols for agent migration over a network, including the Internet. This paper also describes refinement of several protocols, which are useful in the management of cluster computing.

7 Conclusion

We have presented a framework for building a self-configuring infrastructure for agent migration over the Internet, in particular cluster computing environments on the Internet. The framework provides a layered architecture for adaptive protocols for mobile agents. These network protocols for agent migration can be naturally implemented within mobile agents and thus can be dynamically added to and removed from the system by migrating the corresponding agents, according to the requirements of visiting agents and changes in the environment. We presented several mobile agent-based protocols based on standard protocols in the Internet. Our prototype implementation built on a Java-based mobile agent system, called MobileSpaces, allowed us to experiment with the construction and deployment of these protocols. This experience strongly suggests that the use of active network technologies in mobile agents holds considerable promise and that our framework can dynamically and flexibly customize network processing for agent migration, without any limitation on the reusability of application-specific agents.

Finally, we would like to mention further issues. Our early performance measurements indicate that the performance of the adaptive protocols is reasonable for a high-level prototype and fast enough for experimenting with application-specific protocols. However, the performance of the current implementation is not yet satisfactory. We plan to improve the performance. We are interested in developing various agent migration protocols for the Internet, in addition to the examples presented in this paper. Also, the adaptive protocols do not always depend on our framework and thus should be applicable to other active network infrastructure.

References

[1] Y. Aridor, and D.B. Lange, “Agent Design Patterns: Elements of Agent Application Design”, in Proc. Second International Conference on Autonomous Agents (Agents ’98), ACM Press, pp. 108-115. 1998.

[2] C. Baumer, and T. Magedanz, “The Grasshopper Mobile Agent Platform Enabling Short-Term Active Broadband Intelligent Network Implementation”, Proceedings of International Working Conference on Active Networks, pp.109–116, LNCS, Vol.1653, Springer, 1999.

[3] I. Busse, S. Covaci, and A. Leichsenring, “Autonomy and Decentralization in Active Networks: A Case Study for Mobile Agents”, Proceedings of Working Conference on Active Networks, pp.165–179, LNCS, Vol.1653, Springer, 1999.

[4] D. P. Friedman, M. Wand, and C. T. Haynes, “Essentials of Programming Languages”, MIT Press, 1992.

[5] A. Fuggetta, G. P. Picco, and G. Vigna, “Understanding Code Mobility”, IEEE Transactions on Software Engineering, 24(5), 1998.

[6] R. S. Gray, “Agent Tcl: A Transportable Agent System”, CIKM Workshop on Intelligent Information Agents, 1995.

[7] T. Gschwind, M. Feridun, and S. Pleisch, “ADK: Building Mobile Agents for Network and System Management from Reusable Components”, Technical University of Vienna, TUV-1841-99-10, 1999.

[8] B. D. Lange and M. Oshima, “Programming and Deploying Java Mobile Agents with Aglets”, Addison-Wesley, 1998.

[9] N. Minar, K. H. Kramer, P. Maes, Cooperating Mobile Agents for Dynamic Network Routing, in Software Agents for Future Communication Systems, pp.287-304, Springer, 1999.

[10] D. S. Milojicic, W. LaForge, and D. Chauhan, “Mobile Objects and Agents (MOA)”, Proceedings of USENIX Conference on Object Oriented Technologies and Systems, April 1998.

[11] ObjectSpace Inc, “ObjectSpace Voyager Technical Overview”, ObjectSpace, Inc. 1997.

[12] I. Satoh, “MobileSpaces: A Framework for Building Adaptive Distributed Applications Using a Hierarchical Mobile Agent System”, Proceedings of International Conference on Distributed Computing Systems (ICDCS’2000), pp.161-168, IEEE Computer Society, April, 2000.

[13] I. Satoh, “Network Processing of Mobile Agents, by Mobile Agents, for Mobile Agents”, Proceedings of Workshop on Mobile Agents for Telecommunication Applications (MATA’2001), LNCS, Vol.2164, pp.81-92, Springer, August, 2001.

[14] I. Satoh, “Dynamic Configuration of Agent Migration Protocols for the Internet”, Proceedings of International Symposium on Applications and the Internet (SAINT’2002), IEEE Computer Society, pp.119-126, January, 2002.

[15] I. Satoh, “A Framework for Building Reusable Mobile Agents for Network Management” Proceedings of Network Operations and Managements Symposium (NOMS’2002), pp.51-64, IEEE Communication Society, April, 2002.

[16] M. Strasser and J. Baumann, and F. Hole, “Mole: A Java Based Mobile Agent System”, Proceedings of ECOOP Workshop on Mobile Objects, 1996.

[17] J. G. Steiner, B. Clifford Neuman, and J.I. Schiller, “Kerberos: An Authentication Service for Open Network Systems”, Proceedings of the Winter 1988 Usenix Conference. pp.191–201, February, 1988.

[18] D. L. Tennenhouse et al., “A Survey of Active Network Research”, IEEE Communication Magazine, vol. 35, no. 1, 1997.

[19] J. E. White, “Telescript Technology: Mobile Agents”, General Magic, 1995.

Appendix

Suppose an agent migrates between two nodes by using transmitter agents as described in Section 6. The following code fragment is for the `TCPTransmitter` class, which defines simple transmitter agents on these nodes. `TCPTransmitter` agents can exchange agents with each other via their own communication protocol. Since these `TCPTransmitter` agents are mobile agents, they be created and allocated on nodes dynamically.

```
 1: public class TCPTransmitter extends TransmitterAgent implements AgentEventListener {
 2:   public TCPTransmitter() {
 3:     addAgentListener(this);        // registering itself as a listener
 4:     addChildrenContext(new BaseContext());  // offering a context
 5:     registryAs("transmitter");       // registering itself as one of transmitters
 6:   }
 7:   public void add(AgentEvent evt) {    // invoked when an agent arriving
 8:     Message msg = new Message("serialize");
 9:     msg.setArg(evt.getSourceURL());  // url specifies the arriving agent
10:     byte[] data = (byte[])getService(msg); // serializing the arriving agent
11:     AgentURL dst = url.getTarget();  // dst specifies the original destination
12:     send_agent(data, dst);           // transmitting the serialized agent to dst
13:   }
14:   private send_agent(byte[] data, AgentURL dst) {
15:     // sending the serialized agent (data) to the destination (dst)
16:     ...
17:   }
18:   private receive_agent(byte[] data, AgentURL dst) {
19:     // invoked at receiving data for a remote Transfer
20:     Message msg = new Message("deserialize");
21:     msg.setArg(data);  // data is a serialized agent
22:     msg.setArg(dst);   // dst specifies the destination agent
23:     AgentURL url = (byte[])getService(msg); // deserializing data at dst
24:     ...
25:   }
26:   ...
27: }
```

The MobileSpaces system has an event mechanism based on the delegation-based event model introduced in the Abstract Window Toolkit of JDK 1.1 or later. Each agent can be informed indications of such life-cycle state changes because each agent can have one or more listener objects. A listener object implements a specific listener interface extended from the generic `AgentEventListener` interface, which defines callback methods that should be invoked by the

core system before or after the lifecycle state of the agent changes. For example, the `create()` method is invoked after creation, the `destroy()` method is invoked before termination, the `add()` method is invoked after accepting an inner agent, the `remove()` method is invoked before removing an inner agent, the `arrive()` method is invoked after arriving at the destination, and the `remove()` method is invoked before moving to the destination. The following code fragment defines the `SimpleNavigator` class, which is a simple implementation of the navigator agent presented in Section 5.

```
 1: public class SimpleNavigator extends NavigatorAgent implements AgentEventListener {
 2:   Vector route;
 3:   int i = 0;
 4:   public SimpleNavigator() {
 5:     addChildrenContext(new BaseContext()); // offering a context
 6:     addDefaultListener(this);  // registering itself as a listener
 7:     route = new Vector();      // making a list structure
 8:     route.addElement("first.place.com");  // the 1st destination
 9:     route.addElement("second.place.com"); // the 2nd destination
10:     route.addElement("third.place.com");  // the 3rd destination
11:   }
12:   public void create(AgentEvent evt) {...} // invoked after creation
13:   public void arrive(AgentEvent evt) { // invoked after arriving
14:     Message = new Message("do");       // making a message named "do"
15:     msg.setArg(evt.getCurrentURL());   // its argument is the current address
16:     dispatch(msg);   // invoking the method of its inner agents
17:     System.out.println("moving to the next place");
18:     moveToNextHop(); // trying to move to the next place
19:   }
20:   public void leave(AgentURL url) {...} // invoked before migration
21:   public void add(AgentEvent evt) {  // invoked after accepting an agent
22:     moveToNextHop();
23:   }
24:   public void remove(AgentEvent evt) {...} // invoked before removing an agent
25:   private void moveToNextHop() {
26:     String host = route.elementAt(i++); // i-th element of the route list
27:     try {
28:       // requesting a tranmitter agent to migrate to "host"
29:       go(new AgentURL("transmitter://"+host));
30:     } catch (MalformedURLException e) {...}
31:   }
32:   ...
33: }
```

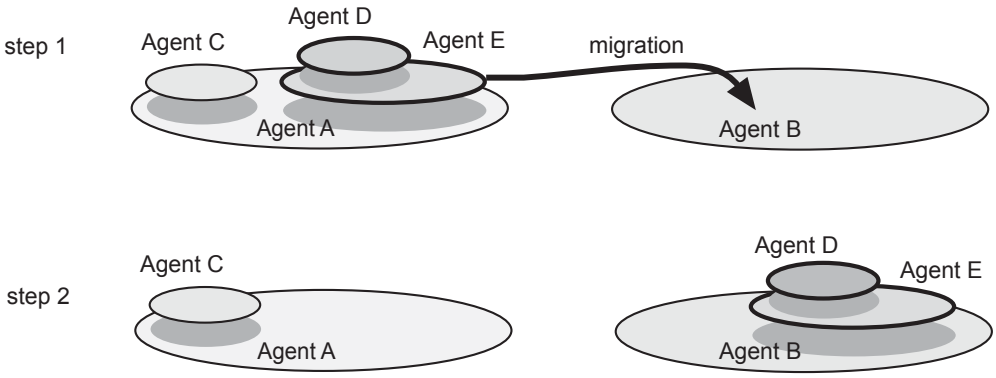

Figure 1: Agent hierarchy and inter-agent migration.

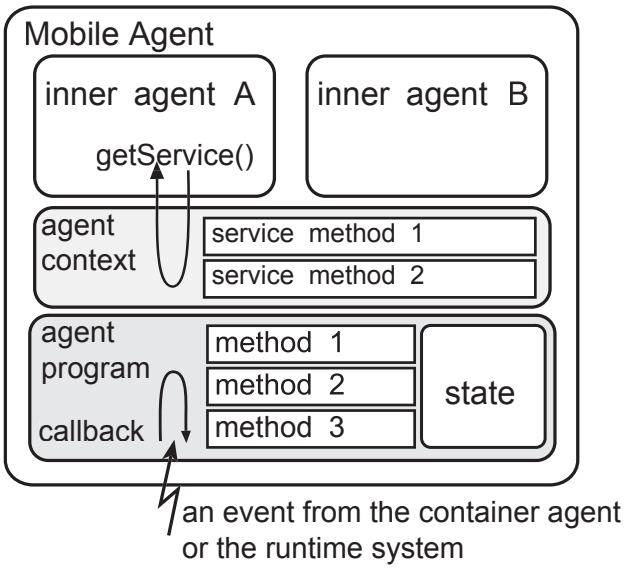

Figure 2: Structure of hierarchical mobile agent.

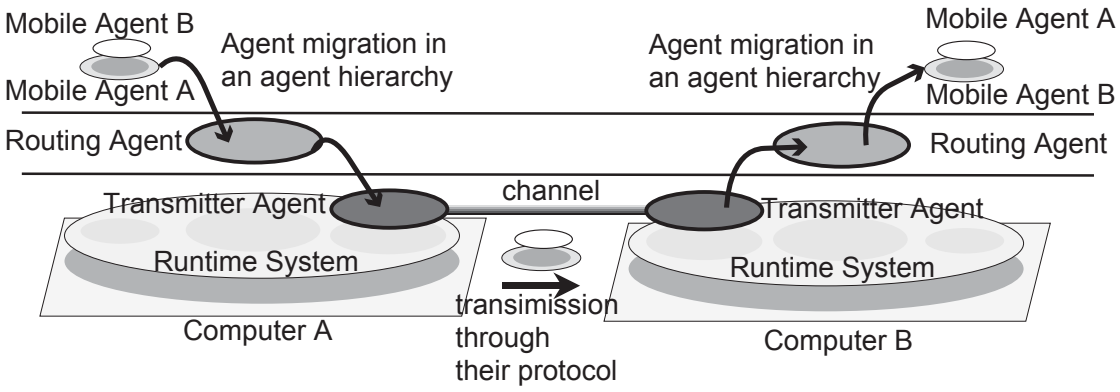

Figure 3: Transmitter mobile agents for establishing channels between nodes.

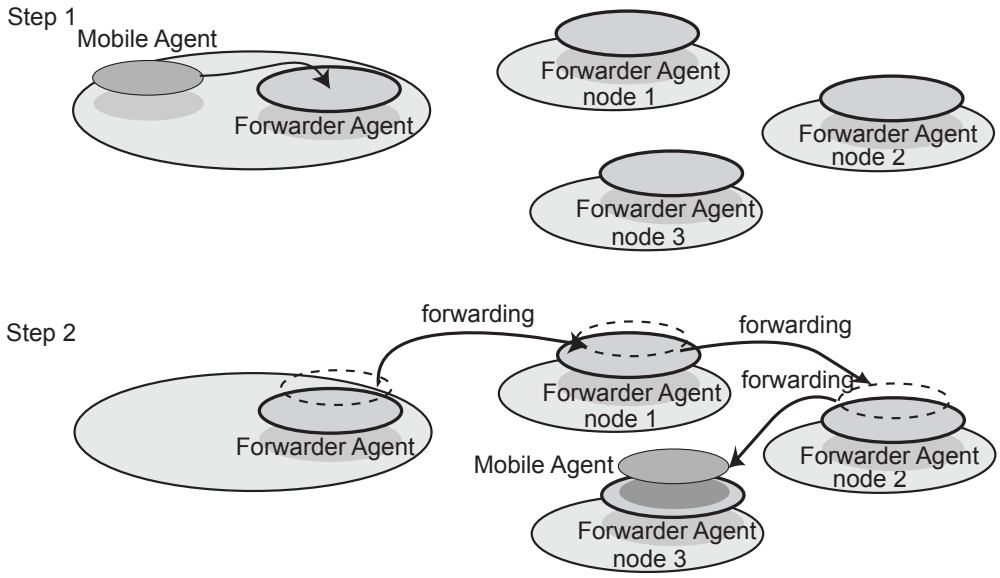

Figure 4: Routing agents for forwarding another agent to the next node.

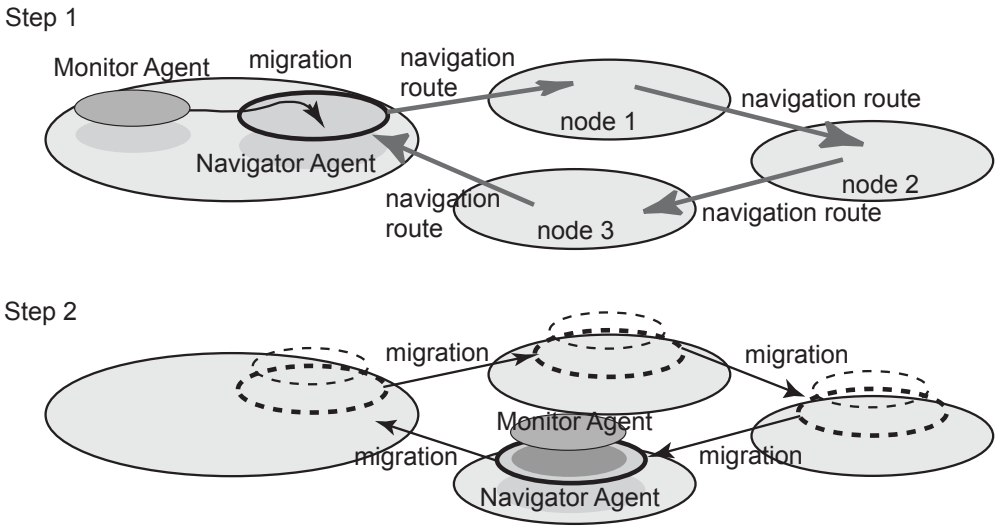

Figure 5: Navigator agent for conveying among nodes with its inner agent.

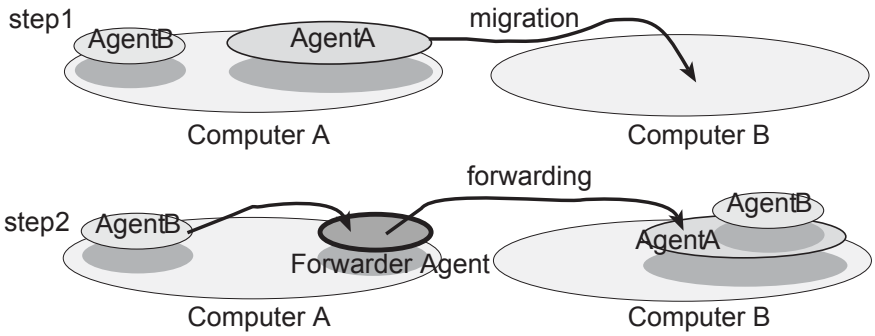

Figure 6: Forwarder agents for locating moving agents.

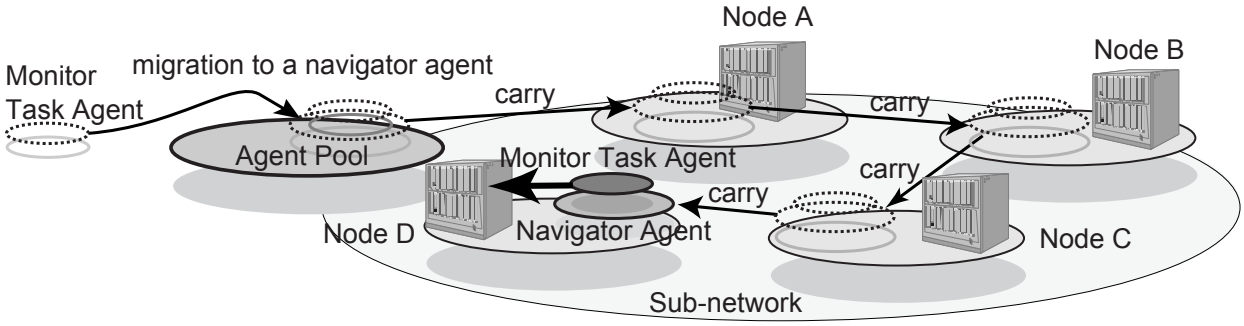

Figure 7: Mobile agent-based management system.

Table 1: Performance of agent migration

Mobile agent-based protocol	Latency (msec)	Throughput (agents/sec)
Transmitter agent (one-hop)	25	7.2
Forwarder agent (per-hop)	38	6.0
Navigator agent (per-hop)	42	5.6